Evaluation of Threshold Cryptography for k-anonymous Dining Cryptographer Networks

Juri Dispan
945106

Bachelor Thesis
VS-B16-2019

Examined by
Prof. Dr.-Ing. Franz J. Hauck

Supervised by
M.Sc. David Mödinger

Institute of Distributed Systems
Faculty of Engineering, Computer Science and Psychology
Ulm University

October 15, 2019

Issued: December 6, 2019

I hereby declare that this thesis titled:

Evaluation of Threshold Cryptography for k-anonymous Dining Cryptographer Networks

is the product of my own independent work and that I have used no sources or materials other than those specified. The passages taken from other works, either verbatim or paraphrased in the spirit of the original quote, are identified in each individual case by indicating the source.
I further declare that all my academic work was written in line with the principles of proper academic research according to the official "Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis" (University Statute for the Safeguarding of Proper Academic Practice).

Ulm, October 15, 2019

Juri Dispan, student number 945106

ABSTRACT

Dining Cryptographer Networks (DCNs) provide strong privacy guarantees for blockchain-based cryptocurrencies, but can easily be jammed. Worse, jamming in DCNs only affects honest clients, while the attacker can keep receiving transmissions. We propose an extension of the DCN protocol that makes such attacks impossible. We use threshold cryptography to force cooperation of at least k network members to read messages delivered over the network. Our system achieves a "either-everyone-or-no-one-can-read" semantic. As a trade-off, performance is typically worse than in classical DCNs. Nevertheless our system achieves transmission rates suitable for use in blockchain applications.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

Current cryptocurrencies are often based on blockchains. Blockchains are append-only databases that record and persist transactions between parties in a public manner. This can be a serious privacy concern, since analysing the public history of transactions can yield private information such as purchasing behaviour or credit balances.

Several current cryptocurrencies have attempted to tackle this problem by employing cryptographic measures like zero-knowledge proofs. Their goals are making individual transactions unlinkable to each other and hiding the involved parties' identities [8, 9]. These measures can preserve privacy against attackers solely having access to the public blockchain, but cannot deter attacks that make use of other sources of information [7, 1]. For example, an attacker might insert malicious nodes into the cryptocurrency's network, enabling them to record parts of the traffic inside the network. This threatens privacy, because a large enough number of nodes under an attacker's control allows them to trace the originator of a message with a high probability. Tracing the physical originator of a message can be utilised to link transactions to IP addresses.

A system proposed by Mödinger et. al. uses Dining Cryptographer Networks (DCNs) to prevent this kind of attack [10]. DCNs are networks in which anonymous broadcasts are possible. They guarantee that both members of the network as well as outside observers can not determine the origin of a broadcast. A weakness of this approach is that such networks can easily be jammed. In DCNs, at most one message can be sent at a time. Jamming the network can be done by constantly broadcasting random bits. Honest members of the DCN are thereby prevented from receiving legitimate broadcasts that are being sent in the network. The attacker on the other hand is unaffected by the jamming and still able to receive legitimate broadcasts.

Depending on the type of information sent, this can give the attacker a significant advantage. That is why we are interested in improving the system in such a way that jamming it for other members while still being able to read other users' broadcasts becomes impossible.

1.2 RELATED WORK

The above mentioned system by Mödinger et. al. splits the process of broadcasting a message in a network in three phases. In this system, the nodes of the network graph are organised in small groups of size s. In the first phase, the sender shares their message with the other nodes in their group using the Dining Cryptographer protocol, which is discussed further in Section 2.1 [2].

In the second phase, a randomly chosen member of the sender's group initiates the transmission of the message to the rest of the network. This is done using an algorithm that uses the topology of the network to reduce an attacker's

chance of guessing the source of the broadcast with the help of statistical methods. Phase three is an ordinary flooding algorithm, which guarantees that the message reaches all members of the network.

Phase two makes it very hard for attackers to determine the source group of a broadcast. Only a strong attacker that controls large parts of the network can theoretically be able to circumvent the second phase's privacy measures. In such a case, phase one still guarantees s-anonymity for the sender. This means that when trying to determine the origin of a message, an attacker can narrow down the set of possible senders to a group of size s, but any findings more precise than this are impossible.

DCNs have previously seen little usage in real-world applications, because they become very inefficient as their number of members rises. The strong anonymity guarantees DCNs offer rely on the fact that each time the DCN protocol is executed, each pair of network members exchanges a message. There have been attempts to circumvent this problem by modifying the protocol in such a way that this is no longer needed. A notable example is "Dissent" [3, 12]. Dissent changes the topology of DCNs to become a client/server architecture. Broadcasting messages is secure as long as at least one server is honest.

Mödinger et. al. solve the Dining Cryptographer protocol's scaling issues by splitting up one DCN into two smaller ones when it becomes too large for efficient message transmission. Transmitting messages between different DCNs is achieved by a more suitable protocol.

The Dining Cryptographer protocol's other problem is its susceptibility to be jammed with the attacker still being able to read transmitted broadcasts. Chaum proposed a method for identifying jammers with an addition to the DCN protocol [2]. In this addition, members must reserve timeslots. In these timeslots, they may transmit a broadcast or lay a trap. If a jammer jams a round in which a trap was laid out, they can be identified. Note that in order to catch cheaters with a high probability, many traps are needed. This reduces the overall efficiency of the protocol. It is also not out of the question that an attacker assumes a new identity when busted, allowing them to continue jamming until they are identified again [6].

1.3 RESULTS

We improved DCNs in such a way that either all or no participants of the network receive broadcasts. In particular, this means that no single attacker is able to jam the network for every other participant. The system we propose has a robustness parameter k. An attacking party must infiltrate the system with at least k network members in order to jam the network while simultaneously receiving broadcasts. The anonymity guarantees of DCNs still apply.

1.4 STRUCTURE OF THIS WORK

In the following Chapter 2 we introduce core concepts that are being used throughout this work and their mathematical basis. We provide an explanation of DCNs as well as an explanation of Shamir-secret-sharing.

In Chapter 3 we set forth our approach to prevent jammers in DCNs from reading broadcast messages. We modify DCNs and combine them with threshold cryptography techniques in such a way that members of a DCN are forced to cooperate in order to be able to read received messages. Any jamming that takes place in such a network has the effect of rendering broadcast messages unreadable for everyone, including the attacker.

In Chapter 4 we present an evaluation of our approach, considering both performance and privacy guarantees.

Chapter 5 provides a short overview of our work.

2 MATHEMATICAL BASICS

In this chapter we discuss some basic mathematical concepts that we use throughout the rest of this work. We introduce the concepts of Dining Cryptographer Networks (2.1) and secret-sharing (2.2). When necessary, we provide examples to ease understanding.

2.1 DINING CRYPTOGRAPHER NETWORKS

Dining Cryptographer Networks are a type of network that allows for anonymous broadcasts. That is, members of the network can broadcast messages to each other without being identified as the sender. A necessary precondition for the protocol to work is that all members of the network can communicate with each other and that every pair of members has the ability to generate shared secrets in a secure manner (for example by performing a Diffie-Hellman Key Exchange [5]).

In the following sections we will discuss the details of DCNs.

2.1.1 THE BASIC CASE

Dining Cryptographer Networks were invented by Chaum in 1988 [2]. The name "Dining Cryptographer Network" comes from the example Chaum used to introduce the concept. It goes as follows:

Three cryptographers are having dinner in a restaurant. The waiter informs them that their meal has been paid for either by the National Security Agency (NSA) or anonymously by one of them. They now wonder if the NSA has paid. The obvious approach to solve this problem is asking each cryptographer whether or not they have paid. In the event that one of them has paid, is would lead to the de-anonymisation of the paying cryptographer, which is unwanted. In order to find out if the NSA has paid and to preserve privacy in case one of them has paid, they execute the following two-stage protocol:

In the first stage, every two cryptographers establish a secret, which can be either 0 or 1. This can be achieved by tossing a coin. Every cryptographer is now in possession of two secret bits, each of which has been established with an other cryptographer.

In the second stage, every cryptographer combines their two secret bits. Every non-paying cryptographer chooses the XOR operation for combining the two secrets. A paying cryptographer on the other hand chooses the XNOR operation, which is the inversion of the XOR operation. Each cryptographer then publishes the result of combining their two secrets.

XORing all published bits yields the answer to the question whether the NSA has paid or not: A 0 signals that none of the cryptographers has paid, which means that the NSA must have paid. A 1 signals that one of the cryptographers has paid.

In case one of the cryptographers has paid, one can't tell which one of the cryptographers did it. Proof of this is given in [2].

Input: Message m_{in}, Group members $g_1, g_2, \ldots, g_n$, Message length ℓ.
Output: Message m_{out} which is the same across all participants.

1. Establish secrets $s_1, \ldots, s_n$, where s_i is shared with g_i and of length ℓ.
2. Compute $M = \bigoplus_{i=1\ldots n} s_i \oplus m_{in}$.
3. Send M to $g_1, g_2, \ldots, g_n$.
4. Receive M_i from $g_i \; \forall i \in \{1 \ldots n\}$.
5. Compute $m_{out} = \bigoplus_{i=1\ldots n} M_i$.

Figure 2.1: One round of the dining cryptographer protocol for a network of size n, as executed by every member of the network separately. Only one member can send a message $m_{in} \neq 0$ per round, every other member has to set $m_{in} = 0$.

2.1.2 GENERALISING DCNS

DCNs are not restricted to only three participants: The above protocol works well for any number of cryptographers. It is also not restricted to single bit messages. In fact, a message can be any size, as long as this size is known to all participants of the network. The generalised version of the protocol is described in Figure 2.1, where $\oplus$ denotes a bitwise XOR operation. Only one participant can send a message at a time, i.e. only one participant can have $m_{in} \neq 0$.

Improving the efficiency of the protocol when executing multiple rounds can be achieved in two ways, as outlined by Chaum in [2]. Essentially, both ideas eliminate the need for every pair of cryptographers to agree on a new secret each time the protocol is executed. The first possibility is to have each pair of cryptographers share a large number of secrets in advance. For example, one cryptographer could generate a random sequence of bits, save it onto a hard drive and hand it to the other cryptographer.

The second possibility is to have each pair of cryptographers agree on a seed for a pseudorandom number generator beforehand. The pseudorandom number generator can then be used to generate shared secret bits as needed. The downside to this approach is that it is vulnerable if the employed number generator is broken.

What a DCN conceptually does is a distributed computation of the bitwise XOR function $\bigoplus_{i=1\ldots n} m_i$ where each participant provides one input value m_i. In case participant g_k is sending a message m_k and every other participant is not, each member computes $m_{out} = \bigoplus_{i=1\ldots n} m_i = 0 \oplus 0 \oplus \cdots \oplus m_k \oplus \cdots \oplus 0 = m_k$. When two or more participants, say $g_{k_1}, g_{k_2}, \ldots$ send messages $m_{k_1}, m_{k_2}, \ldots$ the output of the network becomes $m_{out} = \bigoplus_{i=1\ldots} m_{k_i}$. None of the input messages m_{k_i} can be recovered from m_{out}. This illustrates why in order to transmit broadcasts only one member can send a message at a time.

Example

To ease understanding DCNs, we provide an example illustrating one round of the DCN protocol. The network we look at consists of four members (A, B, C, D) and allows for the transmission of two bits per round. Member A wishes to transmit the following message: $m_A = 01_\text{b}$. No other member wishes to transmit a message, so $m_x = 00_\text{b} \; \forall x \in \{B, C, D\}$.

First of all, every pair of members has to agree on a shared secret of length two. This can, for example, be accomplished be performing a Diffie-Hellman

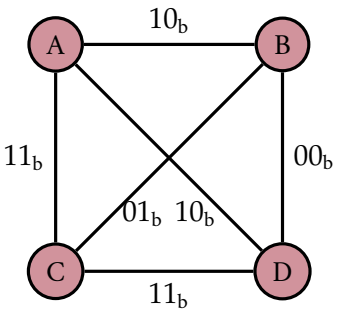
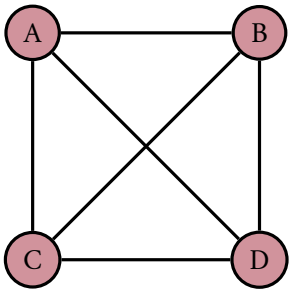


Figure 2.2: DCN after establishing pairwise shared secrets.

$10_b \oplus 10_b \oplus 11_b \oplus 01_b = 10_b$ $10_b \oplus 01_b \oplus 00_b \oplus 00_b = 11_b$



$11_b \oplus 01_b \oplus 11_b \oplus 00_b = 01_b$ $10_b \oplus 00_b \oplus 11_b \oplus 00_b = 01_b$

Figure 2.3: Each member calculates the XOR of their shared secrets and their message (or 00_b if they don't have a message to transmit) and announces the result to the other members XORing all announcements yields the message A wanted to send.

Key Exchange and using the last two bits of the resulting key as the shared secret. Suppose that the shared secrets in our example have been agreed on as seen in Figure 2.2.

Now each member x computes the bitwise XOR of m_x and all of the shared secrets they have with other members. The result is then published, as seen in Figure 2.3. Combining these computation results by XORing them yields the message that A wanted to transmit:

$$10_b \oplus 11_b \oplus 01_b \oplus 01_b = 01_b$$

B, C and D can not detect that it was A who sent the message.

2.1.3 JAMMING DCNS

The network configuration we will focus on in the rest of this Section contains n participants. Participants $g_1, \dots g_{n-2}$ are not sending messages. Participant g_{n-1} is transmitting a message m_{in} with length ℓ and participant g_n is malicious and tries to jam the network. Jamming the network can be achieved by sending a message m_r consisting of ℓ random bits. In this case, every member will compute the output of the network to be $m_{out} = m_{in} \oplus m_r$.

Since $g_1, \dots g_{n-2}$ don't know m_r, they can't recover m_{in} from m_{out}. In general they can't even detect that the network has been jammed, because m_{out} might as well have been produced by a sender sending an encrypted message that seems like random noise.

g_{n-1} of course notices that m_{out} differs from m_{in}. He can therefore conclude that an accidental collision or intentional jamming must have occurred.

g_n knows m_r and can therefore recover m_{in} by computing $m_{out} \oplus m_r = m_{in} \oplus m_r \oplus m_r = m_{in}$.

This is a significant weakness of DCNs: An attacker can prevent other participants from receiving a broadcast while still being able to receive it himself.

2.2 THRESHOLD CRYPTOGRAPHY AND SECRET-SHARING

Threshold cryptography describes a cryptographic system in which decrypting an encrypted message is only possible if a certain number of different parties cooperate. To establish such a system, the key with which the secret is encrypted is typically split up into several shares using a secret-sharing-scheme and distributed among the participants. A certain number of participants can then cooperate to retrieve the key and decrypt the secret.

Secret-sharing describes act of splitting a piece of information into multiple parts in such a way that the original information can only be recovered by combining a certain number of these parts again. A secret-sharing scheme, in which information is split into n pieces and at least k of these pieces are required to restore the original information is called a (n, k) secret-sharing scheme. In such a scheme, no set of $k - 1$ or less pieces can be used to reconstruct the original information, but every set of k or more pieces can.

Both $(n, 1)$ secret-sharing schemes and (n, n) secret-sharing schemes are easily found: A $(n, 1)$ scheme can be accomplished by simply copying the original information $n - 1$ times.

In order to split an information m in a (n, n) scheme one can choose n random numbers $m_1, \dots, m_n$, such that $\sum_{i=1}^{n} m_i = m$.

Producing a (n, k) secret-sharing scheme with $1 < k < n$ on the other hand is more complicated. A famous example is the Shamir-secret-sharing scheme [11].

2.2.1 SHAMIR-SECRET-SHARING

Shamir-secret-sharing is a (n, k) secret-sharing scheme for any k and n with $1 \leq k \leq n$ proposed by Adi Shamir in 1979.

The basic idea of the scheme is as follows: A polynomial $q(x)$ of degree $k-1$ is unambiguously defined by any k points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_i \neq x_j \; \forall i \neq j$ and $q(x_i) = y_i \; \forall i \in \{1 \ldots k\}$.

In order to split some information m, we choose a random polynomial $q(x)$ of degree $k-1$ such that $q(0) = m$. We then evaluate $(i, q(i))$ for every $i \in \{1 \ldots n\}$. The original information m is now split into the resulting points $(1, q(1)), (2, q(2)), \ldots, (n, q(n))$.

Any k or more of these points can be used to restore the $q(x)$ via interpolation and to compute $q(0) = m$, whereas any $k-1$ or less of these points can not.

For security reasons these calculations are not performed using the usual integer arithmetic, but by using finite field arithmetic. We want sets of fewer than k parts to yield no information at all about m. Using real arithmetic $k-1$ parts can restrict the number of possibilities for m, which improves the chances of guessing it. This is not possible with finite field arithmetic.

In practice, that means that we choose a large prime p such that $p > n$ and $p > m$. The random polynomial $q(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ must be generated in such a way that $p > a_i \; \forall i \in \{1 \ldots k-1\}$. Instead of evaluating $(i, q(i))$, we compute $(i, q(i) \mod p)$ for every $i \in \{1 \ldots n\}$, which are the parts usable to reconstruct the original information.

Example

Splitting the information We wish to split the information $m = 1337$ into five parts in such a way that any three of these parts suffice to reconstruct the information. We will employ a (5, 3) Shamir-secret-sharing scheme.

The first step is to choose p. We don't want p to be too small. A small p increases the chance of guessing m, since $m \in [0, p)$. For this example, we'll choose $p = 7919$. We can now generate a random $k-1$-degree polynomial $q(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$. In order to fulfil the requirement that $q(0) = m$, we must choose $a_0 = m = 1337$. $a_1 \ldots a_{k-1}$ can be chosen randomly, a random number generator[1] selected the values $a_1 = 858$ and $a_2 = 387$. The random polynomial is therefore given by $q(x) = 1337 + 858x + 387x^2$. Generating five pieces of m can now be accomplished by computing $(1, q(1) \mod p), \ldots, (5, q(5) \mod p)$:

$$(1, 2582), (2, 4601), (3, 7394), (4, 3042), (5, 7383)$$

Any three of these points can now be used to reconstruct the original information.

Recovering the information We will recover m by using the three points $(x_1, y_1) = (1, 2582)$, $(x_2, y_2) = (2, 4601)$ and $(x_4, y_4) = (4, 3042)$. As a method of interpolation, one can use Lagrange interpolation. We don't need to reconstruct the entire polynomial $q(x)$ in order to restore m. Because $m = q(0) = a_0$ it suffices to calculate the value of a_0. As described in [4], this can be accomplished by computing

$$m = q(0) = (-1)^{k-1} \sum_{s=1}^{k} y_s \prod_{\substack{1 \leq j \leq k \\ j \neq s}} x_j \cdot (x_j - x_s)^{-1} \mod p$$

1 https://www.random.org/

where n^{-1} is the inverse of $n \mod p$. Filling in all variables yields

$$\begin{aligned} m &= \sum_{s \in \{1,2,4\}} y_s \prod_{\substack{j \in \{1,2,4\} \\ j \neq s}} x_j \cdot (x_j - x_s)^{-1} \mod 7919 \\ &= 2582 \cdot 21120 + 4601 \cdot 125421120 + 3042 \cdot 41799122 \mod 7919 \\ &= 1337 \end{aligned}$$

3 APPROACH

3.1 SCENARIO AND ATTACKER MODEL

In this and the following section, we will restrict ourselves to a scenario in which all but one participants are honest. For simplicity, we will assume that only one participant will try to transmit a message in any given round.

The singleton attacker is keen on acquiring some advantage over the other DCN members by preventing them from reading the broadcast message. On the other hand, keeping the ability to read the broadcast has a higher priority for the attacker than the goal of jamming everyone else. Apart from attempting to jam the network, the attacker will not carry out any actions that violate the DCN protocol.

We also require that each pair of participants is able to generate a shared secret.

3.2 IDEA

We extended the DCN protocol in such a way that jamming the network for other participants while still being able to receive broadcasts is no longer possible. The extended DCN protocol has three phases (see Figure 3.1).

The key idea is that a sender no longer broadcasts the message they want to share to every member of the DCN. Instead, they split the message into n parts using a (n, k) secret-sharing scheme (see Section 2.2). We call this the split phase. In the following Distribution phase, each of the n participants of the network then receives a unique share of the secret. After this phase, which can be carried out using a modified version of the DCN protocol, k participants can then cooperate and recover the original broadcast from their k message parts.

Note that the DCN protocol as described in Section 2.1 can only be used to make anonymous broadcasts, but not to send individual messages to certain participants anonymously. We can modify the protocol in such a way that this

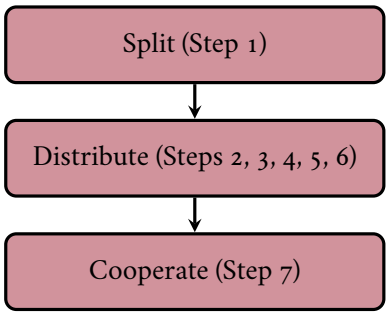


Figure 3.1: The three phases of the modified DCN protocol and their corresponding steps in Figure 3.2. A broadcast gets split up into n shares, which are then distributed to the network's members. Any k members can then cooperate and recover the original message.

Input: Message m_q, Group members $g_1, g_2, \ldots, g_n$, Message length ℓ.
Output: Message m_{out}, the message transmitted to this entity.

1. Split m_q into n parts $m_{q,1}, m_{q,2}, \ldots, m_{q,n}$ using a secret-sharing-scheme.
2. Establish secrets $s_1, \ldots, s_n$, where s_i is shared with g_i and of length ℓ.
3. Compute $M_{send,i} = \bigoplus_{j=1\ldots n} s_j \oplus m_{q,i}\ \forall i \in \{1 \ldots n\}$.
4. Send $M_{send,i}$ to $g_i\ \forall i \in \{1 \ldots n\}$.
5. Receive $M_{rec,i}$ from $g_i\ \forall i \in \{1 \ldots n\}$.
6. Compute $m_{q,out} = \bigoplus_{i=1\ldots n} M_{rec,i}$.
7. Cooperate with $k-1$ other group members in order to restore m_{out}

Figure 3.2: One round of the modified dining cryptographer protocol for a network of size n, as executed by member g_q. g_q provides $m_{q,i}$ as an input for participant i's computation of the distributed XOR function. Only one member in the network can send a message $m_q \neq 0$ in any given round, every other member has to set $m_q = 0$.

becomes possible. See the modified version of the DCN protocol in Figure 3.2. The key modification compared to the original DCN protocol as described by Chaum in [2] and shown in Figure 2.1 is that step 3 no longer makes a broadcast, but transmits individual messages to other participants.

If we understand a DCN as the distributed computation of the XOR function (as described in Section 2.1), the output that participant g_q computes is now no longer $m_{out} = \bigoplus_{i=1\ldots n} m_i$ but rather $m_{q,out} = \bigoplus_{i=1\ldots n} m_{i,q}$. When all participants behave in accordance with the protocol, then only one member g_p inputs messages $m_{p,i} \neq 0$ into the distributed computation. The output for every member g_q thus becomes $m_{q,out} = \bigoplus_{i=1\ldots n} m_{i,q} = 0 \oplus 0 \oplus \cdots \oplus m_{p,q} \oplus \cdots \oplus 0 = m_{p,q}$.

In this scenario, an attacker can either jam the network or read the broadcast message, but not do both at the same time. Jamming the modified DCN by sending random messages is still possible, but instead of receiving the broadcast message, an attacker will only receive one part of the message. When a secure secret-sharing scheme like the Shamir-secret-sharing scheme (see 2.2.1) is used, this does not allow the attacker to draw any conclusions about the message the sender intended to broadcast.

Without jamming attempts, every participant including the sender are in possession of one share of the message. Each member must now collaborate with $k-1$ other members in order to recover the originally sent message. For this protocol, it is not important how exactly this collaboration is organised, as long as a minimal amount of overhead communication is produced. An example for such an organisation scheme that, when executed correctly by each network member, produces a minimal amount of transmitted messages is given in Figure 3.3. Here, a total of $n \cdot (k-1)$ messages are being transmitted.

3.3 MULTIPLE ATTACKERS IN ONE DCN

A single attacker can not jam our modified DCN while still being able to read messages transmitted over it. Multiple attackers in the same network on the other hand, can. At least k malicious participants can cooperate to conduct such an attack in the following way:

Input: Message part m_i, Group members $g_1, g_2, \ldots, g_n$, Amount of required message parts k
Output: Message m_{out}

1. Compute $A_i = \{i + a \mod (n+1) \mid a \in \mathbb{N}, 1 \leq a \leq k-1\}$
2. Send m_i to $g_j \; \forall j \in A_i$
3. Receive m_r from g_r
   $\forall r \in \{x \mid \exists a, 1 \leq a \leq k-1 : x + a \mod (n+1) = i\}$
4. Recover m_{out} from the $k-1$ received messages and m_i.

Figure 3.3: A round of the cooperation phase as executed by network member i. Each participant sends their message part to the members with the $k-1$ next higher indices. When only $s < k-1$ higher indices exist they send their message part to the members with the s next higher indices and the members with the $k-1-s$ lowest indices.

One of the malicious participants jams the network with random bits. Their accomplices can then be provided with those exact random bits and use them to unjam their respective message shares. The attackers can now cooperate and uncover the original broadcast with their k message parts.

To understand why this works, we consider DCNs again as a distributed computation of the XOR function. As explained in Section 3.2, participant g_q computes the output of the modified DCN to be $m_{q,out} = \bigoplus_{i=1\ldots n} m_{i,q} = 0 \oplus 0 \oplus \cdots \oplus m_{p,q} \oplus \cdots \oplus 0 = m_{p,q}$, where $m_{p,q}$ is the only non-zero input into g_q's calculation. When member g_r is jamming the network (i.e. providing random inputs $m_{r,q}$) the output for member g_q becomes $m_{q,out} = m_{p,q} \oplus m_{r,q}$. g_r can provide every g_q that is an accomplice with $m_{r,q}$. Given $m_{r,q}$, g_q can compute $m_{q,out} \oplus m_{r,q} = m_{p,q} \oplus m_{r,q} \oplus m_{r,q} = m_{p,q}$, which is one of the shares needed to reconstruct the original message. When the k or more attackers then combine their message shares, they can reconstruct the original message.

3.4 IMPLEMENTATION

We implemented a prototype that is able to simulate both the original DCN protocol as well as our modified version. The prototype is written in Java, because Java's object oriented style allows us to model the scenario on hand very well. Cryptographers have been modelled as Java Threads. They exchange messages via shared memory and use Java's built-in `Semaphore` and `CyclicBarrier` classes for synchronisation. Every pair of cryptographers can generate shared secret bits as needed by using pseudorandom number generators seeded with the same but random initial value.

For threshold cryptography, we used the open-source library `shamir`[1] in version `0.7.0`, which implements (n, k) Shamir-secret-sharing using the Galois field $\mathrm{GF}(2^8)$ as a finite field.

This library enables us to do secret-sharing by providing the two functions `split()` and `join()`. `split()` takes a message as an argument and produces n message shares. `join()` takes k or more message shares and recovers the original message from them.

1 https://github.com/codahale/shamir

Table 3.1: Complexities of functions provided by the Java library `shamir`, which implements a (n, k) Shamir-secret-sharing scheme. ℓ denotes the length of the original message as well as the message parts.

Function	Complexity
`split()`	$\mathcal{O}(\ell \cdot (n + k))$
`join()`	$\mathcal{O}(\ell \cdot k^2)$

We have determined the computational complexity of these functions by conducting a code review. The results are displayed in Table 3.1. We discovered that the function `split()` is in $\mathcal{O}(\ell \cdot (n + k))$, while `join()` is in $\mathcal{O}(\ell \cdot k^2)$.

4 EVALUATION

We evaluated the modified version of DCNs considering both performance and privacy guarantees. Our version of DCNs offers the same privacy guarantees as the DCNs described by Chaum in [2]. A drawback of our version is the performance, which is worse than that of Chaum's DCNs, but still acceptable for real-world applications like the system described in [10].

4.1 PERFORMANCE

We evaluated the performance of the modified DCN protocol by comparing it to the performance of Chaum's original version. Our particular interest was in investigating the behaviour of the network when scaling in size, i.e. increasing the parameter n. Other parameters that influence the behaviour of the network are k (The number of message parts needed to restore one message) and the number of bytes that get transmitted in each round of the Dining Cryptographer protocol ℓ.

We do not consider the step of having each pair of participants generate a shared secret here. As discussed in subsection 2.1.2, there are ways to establish shared secret bits without the need to repeatedly exchange messages over a network, such as using identically seeded pseudorandom number generators as in our prototype.

Members of the network are modelled as concurrently running Java threads in our prototype. These threads exchange information by writing it into shared memory. We can simulate network latency by adding a delay to reading from shared memory.

If not specified otherwise, the benchmarks we conducted had this delay set to 0, because predicting the communication delay in a real-world system is difficult and having no delay at all shows the performance differences between the original DCN protocol and our system most clearly.

We performed the benchmarks in the following manner: Each benchmark consists of several tests, these tests in turn consist of several runs. One run consists of initialising a DC network, having one member transmit a broadcast and then shutting down the network. What we call "test" is performing a run several times, measuring the duration of the run each time. From these values and the size of the transmitted information we calculate the average and the standard deviation of the throughput. In this context, throughput is calculated as $\frac{\text{size of the transmitted message}}{\text{duration of a run}}$. In each test we ran, $10 - 30$ runs were performed.

A benchmark is running several tests. Across these tests, one network parameter takes on different values. All other values stay constant across all tests in a benchmark.

In order to mitigate distortion of our results due to runtime optimisation attempts by the Java virtual machine, we ran a warm-up phase before each test. In this warm-up phase, 100 runs were performed which we didn't include in our results. By the time these runs were completed, most JVM optimisation has

probably been applied. This guarantees equal conditions across runs that we counted in our results.

4.1.1 FINDING THE OPTIMAL NUMBER OF BYTES PER ROUND

Both original DCNs and our modified DCNs transmit a message of the fixed length ℓ each round. For simplicity, we assume that ℓ can be specified in bytes.

We want to keep ℓ as close as possible to the actual length of the information we want to send. If the information a sender wants to broadcast is longer than ℓ, they need to split this information into multiple parts of size ℓ or smaller and transmit over multiple protocol rounds. Independent of the exact method of transmission, each protocol round produces overhead. When communication is done via a computer network, the duration of one round can not be shorter than the maximum latency between two participants, regardless of the available bandwidth. This latency is added to the total duration every round, which in turn increases as more rounds are carried out. In our prototype's case, where communication is performed using shared memory, each round requires synchronisation between multiple threads, which also produces overhead.

If the information is shorter than ℓ, it can be padded with 0-bytes to make it size ℓ. This leads to the transmission of more data than necessary, producing overhead as well.

This effect can be seen in Figure 4.1 and Figure 4.2, where we benchmarked our prototype with several different values for ℓ ranging from 32 B up to 32 kB. We chose the relevant parameters for this benchmark with regard to the potential use for our proposed system (see Section 1.1) in the field of cryptocurrencies. In an approach as described by Mödinger et. al. in [10], a cryptocurrency's peer-to-peer network is broken down into small groups that perform a form of the Dining Cryptographer protocol. These groups have a size ranging from 4 to 10 participants. We conducted benchmarks with $n = 4$ and $n = 10$.

The transmitted information had a size of 8 kB. As expected, the performance is at its peak when ℓ is roughly equal to the size of the information to transmit, i.e. 8 kB. This seems to be independent on the number of nodes in the network.

4.1.2 SCALING THE NETWORK

In one round of the original Dining Cryptographer protocol (Figure 2.1) each participant computes $M = \bigoplus_{i=1\ldots n} s_i \oplus m_{in}$ and transmits it to every other cryptographer. This produces a total amount of $n \cdot (n-1) \in \mathcal{O}(n^2)$ messages.

A round of the modified version of the Dining Cryptographer protocol (Figure 3.2) produces the same messages. In addition, the cooperation phase (Figure 3.3) requires $n \cdot (k-1)$ messages (see Section 3.2).

When keeping k constant, the modified version of the protocol requires $\mathcal{O}(n)$ more transmissions than the original one. As we see in Figure 4.3, this makes a significant difference for a low number of participants. Because both versions of the protocol are of overall complexity $\mathcal{O}(n^2)$, this linearly growing performance penalty becomes less of a concern when n grows large.

The increased number of sent messages is only one reason for the worse performance of our system. The time that performing one round of the protocol

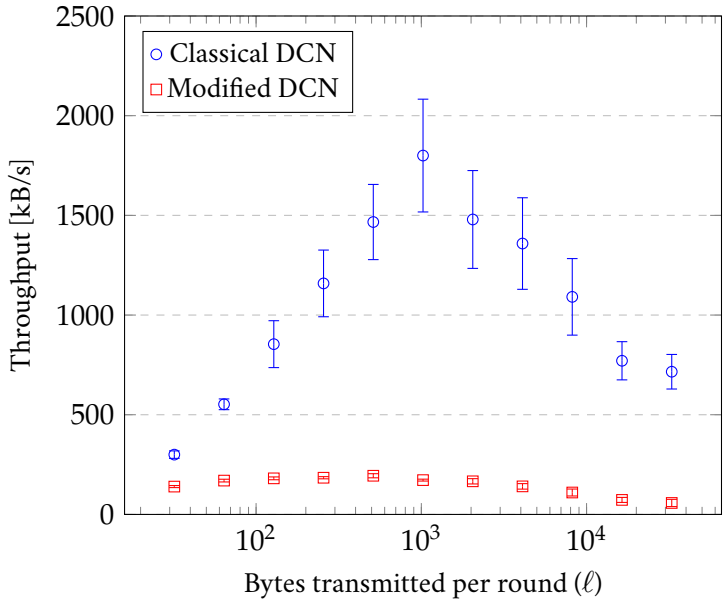


Figure 4.1: The measured throughput and its standard deviation while increasing ℓ and leaving $n = 4$. The transmitted information has size 8 kB. The modified DCN has parameter $k = 3$. Performance reaches its peak when ℓ is about the size of the transmitted information.

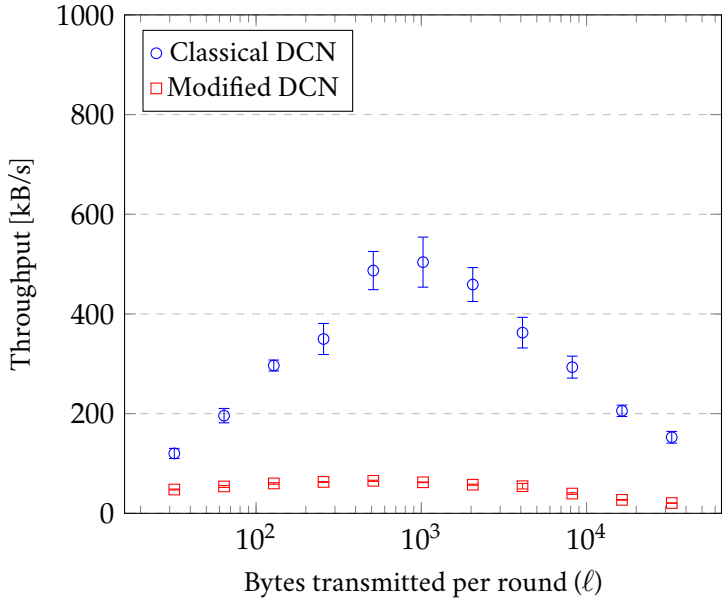


Figure 4.2: The measured throughput under the same parameters as in 4.1, but with $n = 10$. Overall performance is lower that with $n = 4$, but still reaches its peak when ℓ is about the size of the transmitted information.

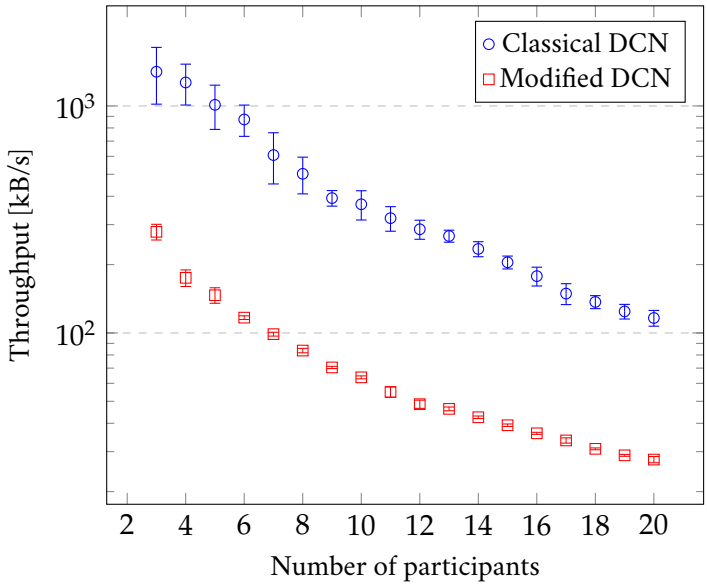


Figure 4.3: Measuring throughput in DCNs of various sizes. Parameter ℓ as well as the size of the transmitted information are 8 kB. In the modified version of DCN we chose $k = 3$.

needs can be divided into two parts. Time spent communicating and time spent doing calculations. The network's participants in our prototype communicate via shared memory and not via a computer network. This is significantly faster and causes the time needed for performing calculations to have a much higher impact on the overall duration of a round. Our system requires a larger amount of computations compared to classical DCNs, because in addition to performing the core DCN functionality it also splits and joins the messages to transmit using a secret-sharing-scheme. That is why the performance difference in Figure 4.3 seems so drastic.

In a scenario where communication is performed over a network, this difference is much smaller. We simulated this in our prototype by adding a delay to reading operations in shared memory. With a delay > 0, the gap between original DCNs and our system is notably smaller. The results of adding a delay of 2 ms, 5 ms and 100 ms are shown in Figure 4.4, Figure 4.5 and Figure 4.6 respectively.

Note that when adding delay, our system only improves relative to the original Dining Cryptographer protocol. The absolute performances of both approaches suffer under message transmission delay.

The key measurements are the throughput rates with $4 \leq n \leq 10$. When employed in a system as described by Mödinger et. al. in [10], that is the typical number of members in a network. A typical value for delay in internet communication is 100 ms, so the data shown in Figure 4.6 might come closest to the performance of our system in a real-world application. The measured throughput rate with delay set to 100 ms is 13.58 kB/s for $n = 4$ and 9.12 kB/s for $n = 10$.

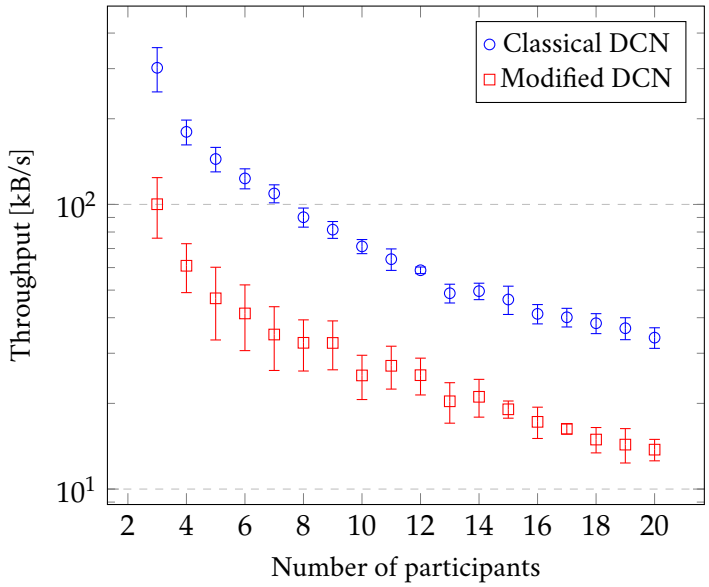


Figure 4.4: Throughput measurement under the same parameters as in Figure 4.3. In this test we set a delay on shared memory write access of 2 ms to simulate delay in network communication.

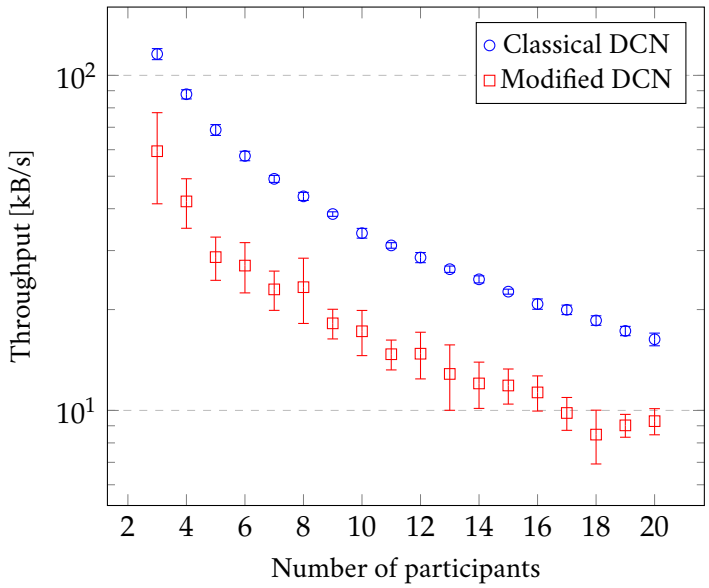


Figure 4.5: The same measurement as in Figure 4.4, but with the shared memory access delay set to 5 ms.

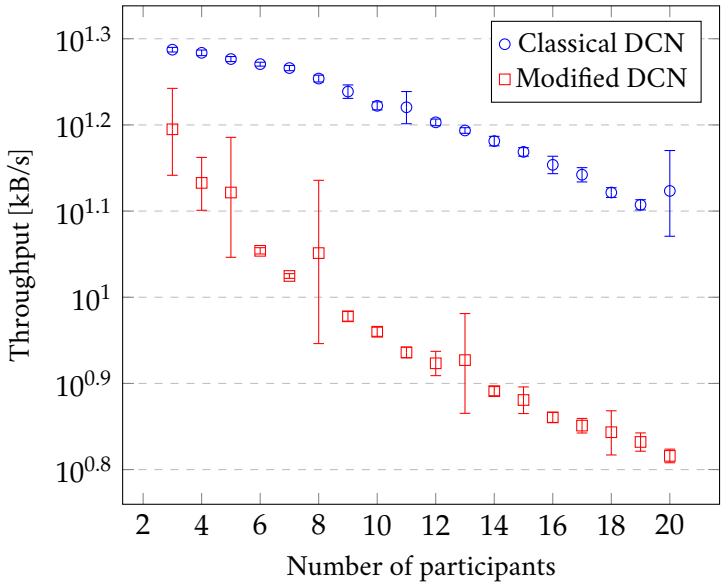


Figure 4.6: The same measurement as in Figure 4.4, but with the shared memory access delay set to 100 ms.

4.1.3 ADJUSTING k

The value of k is the number of message shares needed to restore a message in a (n, k) secret-sharing-scheme. In the original Dining Cryptographer protocol, k is 1, because no message splitting takes place and each participant receives the broadcast in its entirety.

In our system, k impacts performance in two ways. As discussed in subsection 4.1.2, the cooperation phase of the protocol takes an extra $(k-1)$ transmissions for each participant to acquire message shares. Apart from acquiring message shares, each participant has to combine them into the original broadcast. In our prototype we employed Shamir-secret-sharing, which combines message shares using Lagrange interpolation. This interpolation has a complexity of $\mathcal{O}(k^2)$ (see Section 3.4).

Figure 4.7 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.

Benefits of increasing k are that at least k attackers need to collude in order to perform a jamming attack on a network while still being able to receive broadcasts themselves. When k is large, inserting k infiltrators into a given network might be hard, which can prevent attacking parties from doing so.

Choosing k is therefore a trade-off between performance and resistance against such attacks.

4.1.4 CLASSIFYING THE PERFORMANCE

As we have seen, our version of DCNs typically achieves throughput rates between 10 kB/s and 100 kB/s. A real-world application for our system lies in

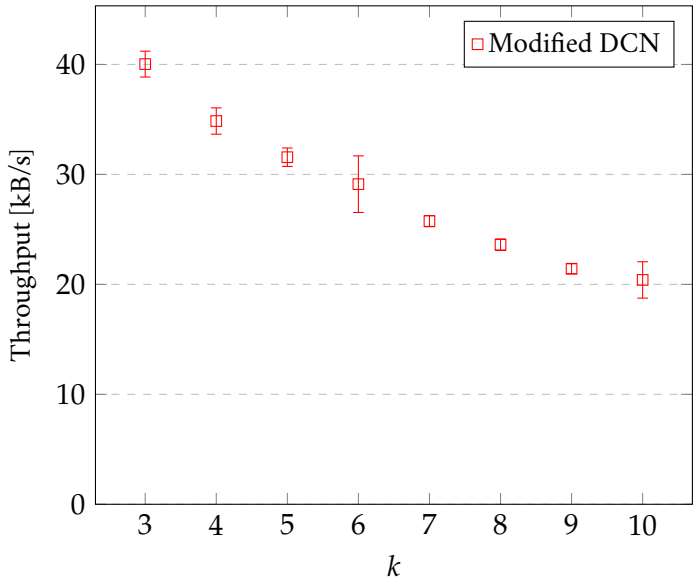


Figure 4.7: Measuring network throughput while varying parameter k. The other parameters are kept constant with $n = 10$ and ℓ as well as the size of the transmitted information as 8 kB.

the anonymous transmission of transaction data for blockchains, e.g. in an environment as the one proposed in [10]. Such transaction data are typically of size < 1 kB, whereas group sizes are between $n = 4$ and $n = 10$ and transmission delay is around 100 ms. The transmission of such data therefore certainly lies well below 1 s. In our estimate, every system that can achieve speeds of > 1 transaction made/s is very much suitable for application in a system as the one proposed in [10]. Our system is therefore probably well-suited for such a task.

4.2 PRIVACY

DCNs as described by Chaum allow anonymous broadcasts for one participant per round. Proof of this is given in [2]. The same guarantee of anonymity holds true in our modified version of DCNs. Both outside observers and network members can not determine the origin of a broadcast.

4.2.1 FROM INSIDE THE NETWORK

For preserving privacy against other participants, step 4 of the original Dining Cryptographer protocol and step 5 of our modified version are crucial. In both cases, each participant can not determine the originator of a broadcast by analysing the messages they receive from the other group members. Both the originator's messages and the messages of non-broadcasting members look like random noise. Only when combining these messages the broadcast can be restored.

4.2.2 FROM OUTSIDE THE NETWORK

An attacker that is not part of a modified DCN but has the ability to read all traffic in the network can not locate the origin of any broadcast. The data that is exchanged between members of the network looks random and can therefore not be used to reach any conclusions about the sender.

Such a powerful attacker could not even read one of the transmitted message parts, even if transmission occurs without any additional encryption layer. For receiving the message part directed to g_q one needs to know $m_{i,q}\ \forall i \in \{1 \dots n\}$. An attacker with wiretapping abilities can read $m_{i,q}\ \forall i \neq q$ as they need to be transmitted between network members. $m_{q,q}$ on the other hand never gets transmitted and thus can't be recorded.

An attacker with access to all transmissions can try to gain access to message parts by eavesdropping on the cooperation phase of our protocol and combining them into the original broadcast. Such an attack can be prevented by having members use an encrypted channel for the cooperation phase, for example by making use of TLS.

All in all we can assert that any outside observers, even when having full access to the modified DCNs communications, can not obtain any information about the sender or the content of any broadcast.

5 CONCLUSION

In this work we showed how malicious entities in Dining Cryptographer Networks can prevent the transmission of a broadcast to all network members except themselves. We proposed a way to modify the existing Dining Cryptographer protocol by applying techniques from the field of threshold cryptography in such a way that attacks of this kind are no longer possible.

We evaluated our system considering both performance and privacy guarantees. Performance was tested by consulting a prototype we implemented. The performance of our system is below the performance of the original Dining Cryptographer protocol. For application in an environment where transmissions of at most a few kilobytes are required, this is sufficient. Privacy-wise both versions are on par.

Our system might be interesting when one is willing to forfeit performance in favour of the guarantee that either all or none of a network's members received a broadcast.

5.1 FUTURE WORK

In order to bring our system into a real-world application, further research is required. Measuring performance of our system with message transmission over a real network can verify suitability for use in a real-world environment.

A feature which our system lacks is the ability to detect and exclude members that try to jam the network. As discussed in Section 1.2, there have been approaches that work in the original DCN protocol. Further work concerning our system could investigate the applicability of these approaches in our version of DCNs.

BIBLIOGRAPHY

[1] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. "Deanonymisation of Clients in Bitcoin P2P Network". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 15–29.

[2] David Chaum. "The dining cryptographers problem: Unconditional sender and recipient untraceability". In: *Journal of Cryptology* 1.1 (Jan. 1988), pp. 65–75. ISSN: 1432-1378.

[3] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: Accountable Anonymous Group Messaging". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: ACM, 2010, pp. 340–350.

[4] Ed Dawson and Diane Donovan. "The breadth of Shamir's secret-sharing scheme". In: *Computers & Security* 13.1 (1994), pp. 69–78. ISSN: 0167-4048.

[5] Whitfield Diffie and Martin Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654.

[6] Philippe Golle and Ari Juels. "Dining Cryptographers Revisited". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 456–473.

[7] Philip Koshy, Diana Koshy, and Patrick D. McDaniel. "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". In: *Financial Cryptography*. 2014.

[8] Ian Miers et al. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *2013 IEEE Symposium on Security and Privacy*. May 2013, pp. 397–411.

[9] Andrew Miller et al. "An Empirical Analysis of Linkability in the Monero Blockchain". In: *CoRR* abs/1704.04299 (2017).

[10] David Mödinger et al. "A Flexible Network Approach to Privacy of Blockchain Transactions". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 1486–1491.

[11] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782.

[12] David Isaac Wolinsky et al. "Dissent in Numbers: Making Strong Anonymity Scale". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI'12. Hollywood, CA, USA: USENIX Association, 2012, pp. 179–192.