



DS-06-2017: Cybersecurity PPP: Cryptography

PRIViLEDGE
Privacy-Enhancing Cryptography in Distributed Ledgers


D3.1 – State of the Art of Cryptographic Ledgers

Due date of deliverable: 31 August 2018

Actual submission date: 30 August 2018

Grant agreement number: 780477
Start date of project: 1 January 2018
Revision 1.0

Lead contractor: Guardtime AS
Duration: 36 months

	Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020
Dissemination Level	
PU = Public, fully open	X
CO = Confidential, restricted under conditions set out in the Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC	

D3.1

State of the Art of Cryptographic Ledgers

Editor

Aggelos Kiayias, Michele Ciampi (UEDIN)

Contributors

Behzad Abdolmaleki (UT)

Karim Baghery (UT)

Christian Cachin (IBM)

Janno Siim (UT)

Luisa Siniscalchi (UNISA)

Björn Tackmann (IBM)

Ivan Visconti (UNISA)

Niels de Vreede (TUE)

Michał Zajac (UT)

Reviewers

Peter Gaži (IOHK)

Luisa Siniscalchi (UNISA)

Ivan Visconti (UNISA)

30 August 2018

Revision 1.0

The work described in this document has been conducted within the project PRIViLEDGE, started in January 2018. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477.

The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

©Copyright by the PRIViLEDGE Consortium

Contents

1	Introduction	1
2	Execution Model and Basic Assumptions	2
2.1	Model and Definitions	2
2.1.1	Protocol Execution	2
2.1.2	The Consensus Problem	4
2.2	Network Assumptions	5
2.2.1	Communication Primitives	5
2.2.2	Synchrony	7
2.3	Setup Assumptions	8
2.3.1	No Setup	8
2.3.2	Public-State Setup	8
2.3.3	Private-State Setup	9
2.4	Computational Assumptions	9
2.4.1	Information-Theoretic Security	9
2.4.2	Computational Security	9
3	Theory Results: Consensus	11
3.1	Consensus in the Standard Setting	11
3.1.1	Number of Parties	11
3.1.2	Round Complexity	12
3.1.3	Trusted Setup	13
3.1.4	Communication Cost	13
3.1.5	Beyond Synchrony	13
3.1.6	Property- vs. Simulation-Based Proofs	13
3.2	Consensus in the Peer-to-Peer Setting	14
3.2.1	Number of Parties	14
3.2.2	Running Time	15
3.2.3	Trusted Setup	16
3.2.4	Communication Cost	16
3.2.5	Property- vs. simulation-based proofs	16
3.3	Ledger Consensus	16
3.3.1	Number of Parties	17
3.3.2	Transaction Processing Time	18
3.3.3	Trusted Setup	18
3.3.4	Beyond Synchrony	19

4	Notable Systems	20
4.1	PoW-based Ledgers	20
4.1.1	Bitcoin	20
4.1.2	Ethereum	21
4.2	Ledgers Based on Byzantine-Fault Tolerance (BFT)	22
4.2.1	BFT Consensus Protocols	22
4.2.2	Blockchain systems with BFT and BFT-like consensus	22
4.3	Proof of Stake Ledgers	25
4.3.1	Proof-of-Stake Protocols	25
4.3.2	Proof-of-Stake Blockchain Systems.	25
5	Privacy-Preserving Techniques and Systems	27
5.1	Confidential Assets	27
5.2	Monero	28
5.2.1	Preliminaries	28
5.2.2	Description of Ring CT	28
5.2.3	Privacy of Ring CT	29
5.2.4	Vulnerabilities	29
5.3	Zerocash	29
5.3.1	Preliminaries	30
5.3.2	Payment Mechanism Ingredients	31
5.3.3	Transferring Coins – Pour Transaction	32
5.3.4	Privacy of Zerocash	32
5.4	Privacy-preserving Auditing for Distributed Ledgers	33
6	The Relation Between Distributed Ledger Technology and Secure Multiparty Computation	35
6.1	Fairness	35
6.2	Miscellaneous Enhancements to Secure Computation	37
6.3	Privacy and Secure Storage	37
7	Open Problems	39

Chapter 1

Introduction

The aim of this deliverable is to provide a preliminary analysis of the state of the art on cryptographic aspects of distributed ledgers. In Chapter 2 we introduce some terminologies and definitions that are used across this document. Chapter 3 focuses on *Consensus*; (a.k.a. Byzantine agreement) which arguably one of the most fundamental problems in distributed systems, playing also an important role in the area of cryptographic protocols as the enabler of a (secure) broadcast functionality. While the problem has a long and rich history and has been analysed from many different perspectives, recently, with the advent of blockchain protocols like Bitcoin, it has experienced renewed interest from a much wider community of researchers and has seen its application expand to various novel settings. One of the main issues in consensus research is the many different variants of the problem that exist as well as the various ways the problem behaves when different setup, computational assumption and network models are considered. This part of the document hence, aims to perform a systematisation of knowledge in the landscape of consensus research starting with the original formulation in the early 80's up to the present blockchain-based new class of consensus protocols. Moreover we study the consensus problem under its many guises, classifying the way it operates in many settings and highlighting the exciting new applications that have emerged in the blockchain era.

In Chapter 4 we classify the cryptographic ledgers in three main categories: Proof of Work (PoW), Byzantine-Fault Tolerant (BFT) and Proof of Stake (PoS) based ledgers, and describe the main aspects of those categories together with a summary on the state of the art. In Chapter 5 we also focus on privacy-preserving techniques and systems that provide privacy to the users of the blockchain. In more details, we describe the major techniques used to hide information on a blockchain (e.g. the amount of coins that has been transferred in a transaction when the blockchain implements a cryptocurrency). Moreover, a description of some of the well known systems that use privacy-preserving techniques is provided.

Chapter 6 focuses on the relation between distributed ledgers and cryptographic protocols. For example, one of aspect that is explored in this part of the document is how distributed ledgers can improve the security of multi-party computation protocols. Interestingly, the use of distributed ledgers makes possible to achieve novel results in this area.

In the last chapter we propose some open questions that are related to the content of this document and that could be interesting to answer. Part of these open questions arose during the writing of this work document.

Chapter 2

Execution Model and Basic Assumptions

2.1 Model and Definitions

2.1.1 Protocol Execution

In order to provide a formal description of protocols and their executions it is useful to consider a formal model of computation. We choose the Interactive Turing Machine (ITM) based resource bounded model put forth by [Gol01, Can01]. An ITM is like a Turing Machine but with the addition of an incoming and an outgoing communication tape as well as an identity tape and a “subroutine” tape. When an instance of an ITM is generated (we henceforth call this an ITI, standing for interactive Turing Machine Instance), the identity tape is initialised to a specific value that remains constant throughout the instance’s execution. The ITI may communicate with other ITIs by writing to its outgoing communication tape.

Let us consider a protocol Π that is modelled as an ITM. Ideally, we would like to consider the execution of this protocol in an arbitrary setting, i.e., with an arbitrary set of parties and arbitrary configuration. A common way to model this in cryptography and distributed systems is to consider that a certain program, thought of as an adversary, produces this configuration and therefore the properties of the protocol should hold for any possible choice of that program, potentially with some restrictions that are explicitly defined. The advantage of this particular modelling approach is that it obviates the need to quantify over all the details that concern the protocol (and substitutes them with a single universal quantification over all such “environments”).

Suppose now that we have a protocol Π that is specified as an ITM and we would like to consider all possible executions of this protocol in the presence of an adversary \mathcal{A} , that is also modelled as an ITM. We capture this by specifying a pair of ITM’s (\mathcal{Z}, C) , called the “environment” and the “control program” respectively.

The environment \mathcal{Z} is given some input which may be trivial (like a security parameter 1^k) and is allowed to “spawn” new ITIs using the programs of Π and \mathcal{A} . By convention, only a single instance of \mathcal{A} is allowed. Spawning such new instances is achieved by writing a single message on \mathcal{Z} outgoing tape which is read by C . The control program is responsible for approving such spawning requests by \mathcal{Z} . Subsequently, all communication of the instances that are created is routed via C , i.e., C receives the instances’ outgoing messages and checks whether they can be forwarded to the receiving parties’ incoming tape. Note that this may be used to simulate the existence of point to point channels, nevertheless we take a more general approach. Specifically, the control function C by definition only permits outgoing messages of running ITIs to be sent to the adversary \mathcal{A} (with instructions for further delivery). This captures the fact that the network cannot be assumed to be safe for the instances that are communicating during the protocol execution. Beyond writing messages that are routed through \mathcal{A} , ITIs can also spawn additional ITIs as prescribed by the rules hardcoded in C . This enables instances of a protocol Π to invoke subroutines that can assist in its execution. These subroutines can be sub-protocols or instances of “ideal functionalities” that may be accessible by more than a single running instance (more details on ideal functionalities follow in this section).

Given these features, the above approach provides a comprehensive framework for reasoning about protocol executions. Nevertheless, some care needs to be applied to ensure that the total execution runtime of the (\mathcal{Z}, C)

system remains polynomial: it is easy to see that even if all ITIs are assumed to be polynomially bounded, the total execution runtime may not be. By introducing a more refined time-keeping capability for ITIs, it was proven in [Can01] that as long as the ITM \mathcal{Z} is “locally polynomially bounded” (see [Can01] for more details on polynomial time ITMs) and C is a polynomial-time computable function, the execution of (\mathcal{Z}, C) as described above when \mathcal{Z} is given input κ is polynomial-time bounded in κ (cf. Proposition 3 in [Can01]). From this point on we assume always that \mathcal{Z} is locally polynomial bounded and, for brevity, we just state that it is polynomial bounded.

We use $\text{poly}(\cdot)$ to indicate a generic polynomial function. Let κ be the security parameter, and assume that any function, set size or running time implicitly depends on this parameter (especially when we write negl to describe a negligible function in κ —i.e., $\text{negl} < 1/\text{poly}(\kappa)$ for large enough κ). For any ϵ , we say that two distribution ensembles $\{X_\kappa\}_{\kappa \in \mathbb{N}}$, $\{Y_\kappa\}_{\kappa \in \mathbb{N}}$ are ϵ -*indistinguishable*, denoted $\{X_\kappa\} \approx_\epsilon \{Y_\kappa\}$, if for any probabilistic polynomial-time (PPT) algorithm \mathcal{C} , for large enough κ ,

$$|\Pr[\mathcal{C}(1^\kappa, X_\kappa) = 1] - \Pr[\mathcal{C}(1^\kappa, Y_\kappa) = 1]| < \epsilon + \text{negl}(\kappa).$$

We say that X and Y are *computationally indistinguishable* and denote $\{X_\kappa\} \approx \{Y_\kappa\}$ if they are ϵ -indistinguishable with $\epsilon = 0$ under a certain computational assumption.

Functionalities. We next need to specify the “resources” that may be available to the instances running protocol Π . For instance, access to a reliable point-to-point channel or a “diffuse” channel (see below). To allow for the most general way to specify such resources we follow the approach of describing them as “ideal functionalities” in the terminology of [Can01]. In simple terms, an ideal functionality is another ITM that may interact with instances running in parallel in the protocol execution. The critical feature of ideal functionalities though is that they can be spawned by ITIs running protocol Π . In such case, the protocol Π is defined with respect to the functionality \mathcal{F} . The ideal functionality may interact with the adversary \mathcal{A} as well as other ITIs running the protocol Π . One main advantage of using the concept of an ideal functionality in our setting is that we can capture various different communication resources that may be available to the participants running the protocol. For instance, a secure channel functionality may be spawned to transmit a message between two instances of Π that only leak the length of the message to the adversary. As another example, a message passing functionality may ensure that all parties are activated prior to advancing to the next communication round.

Execution of multiparty protocols. When protocol instances are spawned by \mathcal{Z} they are initialised with an identity which is available to the protocol program’s code as well as, possibly, with the identities of other instances that may run in parallel (this is at the discretion of the environment program \mathcal{Z}). The identities themselves may be useful to the program instance, as they may be used by the instance to address other instances running in parallel. We use the notation $\text{VIEW}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa)$ denote an *execution* of the protocol Π with an adversary \mathcal{A} and an environment \mathcal{Z} . The execution is a string that is formed by the concatenation of all messages and all ITI states at each step of the execution of the system (\mathcal{Z}, C) . The total length of the execution is polynomial in κ . The parties’ inputs are provided by the environment \mathcal{Z} which also receives the parties’ outputs. Parties that receive no input from the environment remain inactive. The environment may provide input to a party at any round and may also modify that input from round to round. We denote by INPUT the input tape of each party.

We note that by adopting the resource bounded computation modelling of systems of ITMs by [Can01] we obviate the need of imposing a strict upper bound on the number of messages that may be transmitted by the adversary in each activation. In our setting, honest parties, at the discretion of the environment, are given sufficient time to process all messages delivered by any communication functionality given to them as a resource. It follows that denial of service attacks cannot be used to the adversary’s advantage in the analysis (they are out of scope from our perspective of studying the consensus problem).

Properties. In this document we talk about the *properties* of protocols Π . Such properties are defined as predicates over the random variable $\text{VIEW}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa)$ by quantifying over all possible adversaries \mathcal{A} and environments \mathcal{Z} . Note that protocols may only satisfy some properties with a small probability of error in κ as well as in potentially other parameters. The probability space is determined by the private coins of all participants and the functionalities they employ. Formally we have the following.

Definition 1. Given a predicate Q we say that *the protocol Π satisfies property Q* provided that for all polynomial-time \mathcal{Z}, \mathcal{A} the probability that $Q(\text{VIEW}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa))$ is false is negligible in κ .

Note that we only consider properties that are polynomial-time computable predicates. Furthermore, the predicates that we consider conform to the following rule: given an execution E distributed according to $\text{VIEW}_{\Pi, \mathcal{A}, \mathcal{Z}}(\kappa)$, if E' is a session of protocol Π that unfolds in the course of E , it holds that Q is defined over E' and $Q(E) \rightarrow Q(E')$.

Asynchronous vs. synchronous execution. Our description above captures both asynchronous and synchronous models of execution. Specifically the asynchronous setting can be captured by a communication functionality that accepts messages and enables the adversary to delay message delivery arbitrarily. Regarding synchronous communication we can easily incorporate it in the model, following [KMTZ13], and incorporate via a functionality a mechanism that keeps track of parties' activations and ensures that parties do not advance to the next "round" until all parties have been activated by the environment.

Static vs. dynamic environments. The model we present captures both static and dynamic environments. Specifically, it is suitable for protocols that run with a fixed number of parties that should be known to all participants in advance but it can also allow protocols for which the number of participants is not known a-priori and, in fact, it may not even be known during the course of the execution. Note that in order to allow for proper ITI intercommunication we need to assume that the total set of parties is known, nevertheless, only a small subset of them may be active in a particular moment during the protocol execution.

Setup assumptions. In a number of protocols, there is a need to have some pre-existing configuration (such as the knowledge of a common random string (CRS), or a public-key infrastructure [PKI]). Such setup assumptions can also be easily captured as separate functionalities \mathcal{F} that are available to the running protocol ITIs.

Permissioned vs. permissionless networks. In the context of the consensus problem, this terminology became popular with the advent of blockchain protocols. The Bitcoin blockchain protocol is the prototypical "permissionless" protocol where read access to the ledger is unrestricted and write access (in the form of posting transactions) can be obtained by anyone that possesses bitcoin (which may be acquired, in principle, by anyone that is running the Bitcoin client and invests computational power solving proofs of work). On the other hand, a permissioned protocol imposes more stringent access control on the read and write operations that are available as well as with respect to who can participate in the protocol. Extrapolating from the terminology as applied in the ledger setting, a permissionless consensus protocol would enable any party to participate and contribute input for consideration of the other parties. With this in mind, the traditional setting of consensus is permissioned, since only specific parties are allowed to participate; on the other hand, consensus in the blockchain setting can be either permissioned or permissionless.

Cryptographic primitives. Some standard cryptographic primitives will be useful in the description of the consensus protocols. We overview them below. A digital signature scheme consists of three PPT algorithms (Gen, Sign, Verify) such that $(vk, sk) \leftarrow \text{Gen}(1^\kappa)$ generates a public-key/secret-key pair, $\sigma \leftarrow \text{Sign}(sk, m)$ signs a message m , $\text{Verify}(vk, m, \sigma)$ returns 1 if and only if σ is a valid signature for m given vk . A digital signature scheme is existentially unforgeable, if for any PPT adversary \mathcal{A} , that has access to a $\text{Sign}(sk, \cdot)$ oracle, the event that \mathcal{A} returns some (m, σ) such that $\text{Verify}(vk, m, \sigma) = 1$ and \mathcal{A} did not query the oracle with m , has measure $\text{negl}(\kappa)$, where the probability is taken over the coin tosses of the algorithms. A collision resistant (keyed) hash function family $\{H_k\}_{k \in K}$ has the property that $H_k : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$, it is efficiently computable and the probability to produce $x \neq y$ with $H_k(x) = H_k(y)$ given k is $\text{negl}(\kappa)$.

2.1.2 The Consensus Problem

Consensus (aka *Byzantine agreement*), formulated by Shostak, Pease and Lamport [PSL80, LSP82], is one of the fundamental problems in the areas of fault-tolerant distributed computing and cryptographic protocols (in particular secure multi-party computation [Yao82, GMW86, BGW88, CCD87]). In the consensus problem, n parties attempt to reach agreement on a value from some fixed domain V , despite the malicious behavior of up to t of them. More specifically, every party P_i starts the consensus protocol with an initial value $v \in V$, and every

run of the protocol must satisfy (except possibly for some negligible probability) the following conditions:

- *Termination*: All honest parties decide on a value.
- *Agreement*: If two honest parties decide on v and w , respectively, then $v = w$.
- *Validity*: If all honest parties have the same initial value v , then all honest parties decide on v .

The domain V can be arbitrary, but typically it is enough to consider the case $V = \{0, 1\}$, as there exists an efficient transformation of binary agreement protocols to the multi-valued case [TC84].

There exist various measures of quality of a consensus protocol: its *resiliency*, expressed as the fraction $\frac{t}{n}$ of misbehaving parties a protocol can tolerate; its running time—worst number of rounds by which honest parties terminate; and its communication complexity—worst total number of bits/messages communicated during a protocol run. In the consensus problem, all the parties start with an initial value. A closely related variant is the single-source version of the problem (aka the *Byzantine Generals* problem [LSP82], or simply (reliable or secure) “broadcast”), where only a distinguished party—the *sender*—has an input. In this variant, both the Termination and Agreement conditions remain the same, and Validity becomes:

- *Validity*: If the sender is honest and has initial value v , then all honest parties decide on v .

A stronger, albeit natural, version of the consensus problem requires the output value to be one of the honest parties’ inputs, a distinction that is only important in the case of non-binary inputs. In this version, called *strong consensus* [Nei94], the Validity condition becomes:

- *Strong Validity*: If the honest parties decide on v , then v is the input of some honest party.

Note that the resiliency bounds for this version also depend on $|V|$ (see Section 3.1).

Finally, we point out that, traditionally, consensus problems have been specified as above, in a *property-based* manner. Protocols for the problem are then proven secure/correct by showing how the properties (e.g., the Agreement, Validity and Termination conditions) are met. Nowadays, however, it is widely accepted to formulate the security of a protocol via the “trusted-party paradigm,” cf. [GMW86, Gol01], where the protocol execution is compared with an ideal process where the outputs are computed by a trusted party that sees all the inputs. A protocol is then said to securely carry out the task if running the protocol with a realistic adversary amounts to “emulating” the ideal process with the appropriate trusted party. One advantage of such a simulation-based approach is that it simultaneously captures *all* the properties that are guaranteed by the ideal world, without having to enumerate some list of desired properties. Simulation-based definitions are also useful for applying *composition theorems* (e.g., [Can00, Can01]) that enable proving the security of protocols that use other protocols as sub-routines, which typically would be the case for consensus and/or broadcast protocols.

The above captures the classical definition of the consensus problem. A related and recently extensively studied version of the problem is state-machine replication or “ledger consensus” that we treat in Section 3.3.

2.2 Network Assumptions

2.2.1 Communication Primitives

Consensus protocols are described with respect to a network layer that enables parties to send messages to each other. An important distinction we make is between point-to-point connectivity vs. message “diffusion” as it manifests in a peer-to-peer communication setting.

Point-to-point channels. In this setting, also known as reliable message transmission (RMT), parties are connected with pairwise reliable and authentic channels. When a party sends a message it specifies its recipient as well as the message contents and it is guaranteed that the recipient receives it. The recipient can identify the sender as the source of the message. In such fixed connectivity setting, all parties are aware of the set of parties running the protocol. Full connectivity has been the standard communication setting for consensus protocols, see [LSP82], although sparse connectivity has also been considered (cf. [DPPU88, Upf92]).

The ideal functionality that captures RMT is presented in Figure 2.1. We remark that the functionality captures a synchronous operation.

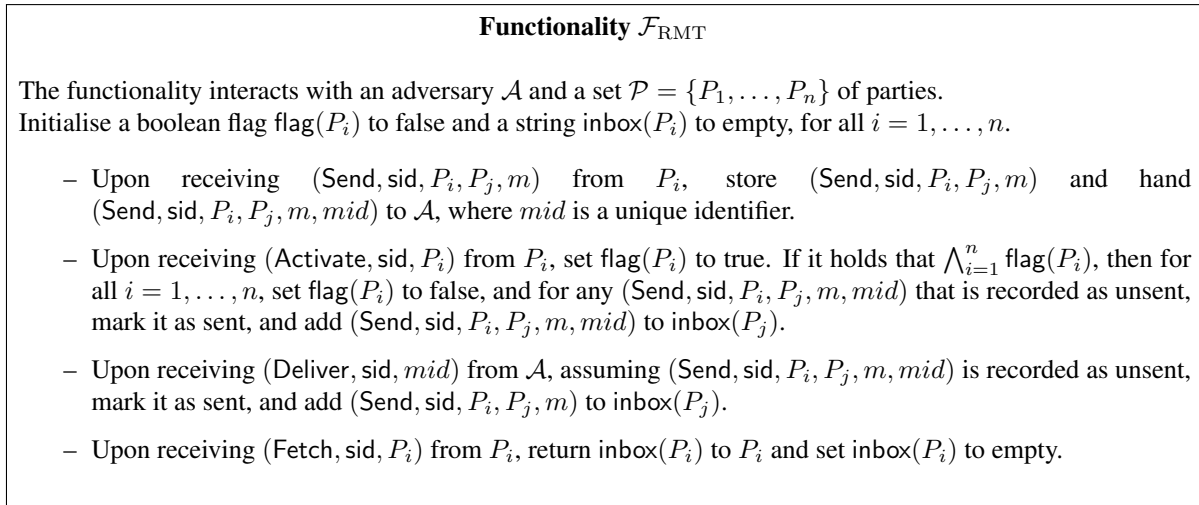


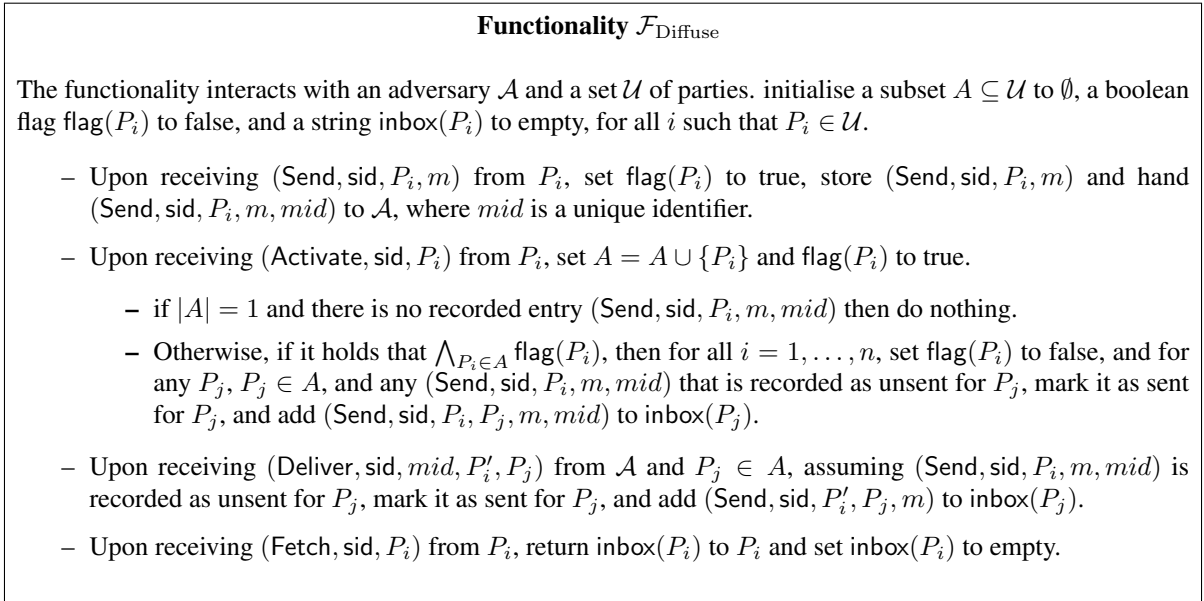
Figure 2.1: *The reliable message transmission ideal functionality.*

In terms of measuring communication costs in this model we use the (maximum) total number of messages in a protocol run, rather than the total number of communicated bits, assuming a suitable message size. See, e.g., [Fit03, Chapter 3] for a detailed account of the communication complexity of consensus (and broadcast) protocols.

Peer-to-peer diffusion. In peer-to-peer the message transmission happens via “gossiping,” i.e., messages received by a party are passed along on to the party’s peers. We refer to this basic message passing operation as “Diffuse.” Message transmission is not authenticated, non-atomic and it does not preserve the order of messages in the views of different parties. When a message is diffused by an honest party, there is no specific recipient and it is guaranteed that all active parties receive the same message. Nonetheless, the source of the message may be “spoofed” and thus the recipient may not reliably identify the source of the message,¹ and when the sender is malicious not everyone is guaranteed to receive the same message. Contrary to the point-to-point channels setting, parties may neither be aware of the other parties that are running the protocol nor their precise number. The ideal functionality capturing the diffuse operation is presented in Figure 2.2 assuming synchronous network operation. A salient feature of protocols running in the $\mathcal{F}_{\text{Diffuse}}$ setting is that the session id may just provide an abbreviation of the universe of parties $\mathcal{P} = \{P_1, \dots, P_n\}$, of which only a subset may be activated.

In order to measure the total communication costs of peer-to-peer diffusion, one needs to take into account the underlying network graph. The typical deployment setting is a sparse constant-degree graph for which it holds that the number of edges equals $O(n)$. In such setting, each invocation of the primitive requires $O(n)$ messages to be transmitted in the network.

¹Note that in contrast to a sender-anonymous channel (cf. [Cha81]), a diffuse channel leaks the identity of the sender to the adversary.

Figure 2.2: *The peer-to-peer diffuse ideal functionality.*

Relation between the communication primitives It is easy to see that given RMT, there is a straightforward, albeit inefficient, protocol that simulates Diffuse; given a message to be diffused, the protocol using RMT, sends the message to each party in the set of parties running the protocol. On the other hand, it is not hard to establish that no protocol can simulate RMT given Diffuse. The argument is as follows, and it works no matter how the protocol using Diffuse may operate. When a party A transmits a message M to party B , it is possible for the adversary in the Diffuse setting to simulate a “fake” party A that sends a message $M' \neq M$ to B concurrently. Invariably, this will result in a setting where B has to decide which one is the correct message to output and will produce the wrong message with non-negligible probability. It follows that Diffuse is a weaker communication primitive: one would not be able to substitute Diffuse for RMT in a protocol setting.

Other models The above models may be extended in a number of ways to capture various real-world considerations in message passing. For instance, in point to point channels, the communication graph may change over the course of protocol execution with edges being added or removed adversarially, something that may also result in temporary network partitions. Another intermediate model between point to point channels and diffusion is to have a diffusion channel with “port awareness”, i.e., the setting where messages from the same source are linkable, or without port awareness, but where each party is restricted to sending one message per round (see Section 2.2.2 for the notion of round) and their total number is known, see [Oku05a].

2.2.2 Synchrony

The ability of the parties to synchronize in protocol execution is an important aspect in the design of consensus protocols. Synchrony in message passing can be captured by dividing the protocol execution in rounds where parties are activated in some sequence and each one of them has the opportunity to send messages which are received by the recipients at the onset of the next round. This reflects the fact that in some scenarios, real world networks messages are delivered most of the time in a timely fashion and thus parties can synchronise the protocol execution in discrete rounds.

A first important relaxation to the synchronous model is to allow a “rushing adversary” [Can00], i.e., allow the adversary to control the activation of parties so that it acts last in each round having access to all messages sent by honest participants before it decides on the actions of the adversarial participants and the ordering of message

delivery for the honest parties in the next round. This is captured by the functionalities in Figures 2.1 and 2.2. A second relaxation is to impose a time bound on message delivery that is not known to the protocol participants. We shall refer to this as the “partially synchronous setting” [DLS88]. The partial synchronous setting is easy to capture by the functionalities in Figures 2.1 and 2.2 as follows: a parameter $\Delta \in \mathbb{N}$ is introduced in each functionality that determines the maximum time a message can remain “in limbo”. For each message that is sent, a counter is introduced that is initially 0 and counts the number of rounds that have passed since its transmission. When this counter reaches Δ the message is copied to the $\text{inbox}(\cdot)$ strings for the active participants.

In [DLS88] considers also the *eventual synchrony model*. In this the synchrony is assumed to hold after a point in time unknown to the protocol participants. This means that such protocols must not violate safety due to violations of any timing assumption (i.e., during asynchronous periods), but their liveness is only guaranteed when the network becomes synchronous.

An even weaker setting than partial synchrony is that of message transmission with eventual message delivery, where all messages between honest parties are guaranteed to be delivered but there is no specific time bound that mandates their delivery in the course of the protocol execution. Note that it is proven that no deterministic consensus protocol exists in this setting [FLP85], and the impossibility can be overcome by randomization [Ben83, Rab83, FM97, KK06]. Solutions of more practical relevance for Byzantine consensus rely on distributed cryptography [CKS05] as prototyped by SINTRA [CP02] or, much more recently, HoneyBadger [MXC⁺16].

Finally, in the “fully asynchronous setting,” where messages may be arbitrarily delayed or dropped, consensus is impossible. A noteworthy consideration with respect to the network models of Section 2.2.1 is that in the Diffuse setting the adversary may flood the honest participants’ incoming communication queues/buffers and thus induce a denial of service attack (the same issue is much easier to deal with in the point-to-point setting, since parties that overutilize their communication link can be ignored). In order to preserve the ability of the honest parties to process all messages as required in the message delivery model, a provision to either “dilate” the round processing time should be allowed as in [GKL15], or impose an *ad-hoc* bound on the messages sent by the adversary per round should be made as in [AD15].

2.3 Setup Assumptions

In the context of protocol design, a setup assumption refers to information that can be available at the onset of the protocol to each protocol participant. Consensus protocols are designed with respect to a number of different setup assumptions that we outline below.

2.3.1 No Setup

In this setting we consider protocols that parties do not utilize any setup functionality beyond the existence of the communication functionality. Note that the communication functionality may already provide some information to the participants about the environment of the protocol; nevertheless, this setting is distinguished from other more thorough setup assumptions that are described below.

2.3.2 Public-State Setup

A public-state setup is parameterized by a probability ensemble \mathcal{D} . For each input size κ , the ensemble \mathcal{D} specifies a probability distribution that is sampled a single time at the onset of the protocol execution to produce a string denoted by s that is of length polynomial in κ . All protocol parties, including adversarial ones, are assumed to have access to s . In this setting, the consensus protocol is designed for a specific ensemble \mathcal{D} .

The concept of a public-state setup can be further relaxed in a model that has been called “sun-spots” [CPS07], where the ensemble is further parameterized by an index a . The definition is the same as above but now the protocol execution is taken for some arbitrary choice of a . Intuitively, the parameter a can be thought as an adversarial

influence in the choice of the public string s . In this setting, the consensus protocol is designed with respect to the ensemble class $\{\mathcal{D}_a\}_a$.

2.3.3 Private-State Setup

As in the public state case, a private state setup is parameterized by an ensemble \mathcal{D} . For each inputs size κ and number of parties n , \mathcal{D} specifies a probability distribution that is sampled a single time to produce a sequence of values (s_1, \dots, s_n) . The length of each value s_i is polynomial in κ . At the onset of the protocol execution, the ensemble is sampled once and each protocol participant receives one of the values s_i following some predetermined order. The critical feature of this setting is that each party has private access to s_i . Observe that trivially the setting of private-state setup subsumes the setting of public-state setup.

As in the case of a public-state setup, it is important to consider the relaxation where the ensemble \mathcal{D} is parameterised by a . As before the sampling from \mathcal{D}_a is performed from some arbitrary choice of a . It is in this sense where private-state setup has been most useful. In particular, we can use it to express the concept of a public-key infrastructure (PKI). In this setting the ensemble \mathcal{D} employs a digital signature algorithm (Gen, Sign, Verify) and samples a value $(pk_i, sk_i) \leftarrow \text{Gen}(1^\kappa)$ independently for each honest participant. For each participant which is assumed to be adversarial at the onset of the execution, its public and secret key pair is set to a predetermined value that is extracted from a . The private input s_i for the i -th protocol participant is equal to $(pk_1, \dots, pk_n, sk_i)$, thus giving access to all parties' public keys and its own private key.

One may consider more complicated interactive setups, such as for example the adversary choosing a some how adversarially based on public information available about (s_1, \dots, s_n) , but we refrain from considering those here.

2.4 Computational Assumptions

The assumptions used to prove the properties of consensus protocols can be divided into two broad categories. In the information-theoretic (aka “unconditional”) setting, the adversary is assumed to be unbounded in terms of computational resources. In the computational setting, on the other hand, a polynomial-time bound is assumed.

2.4.1 Information-Theoretic Security

In the information-theoretic setting the adversarial running time is unbounded. It follows that the adversary may take arbitrary time to operate in each invocation. Note that the protocol execution may continue to proceed in synchronous rounds, nevertheless the running time of the adversary within each round dilates sufficiently to accomodate its complete operation. When proving the consensus properties in this setting we can further consider two variations: perfect and statistical. When a property, Agreement for example, is perfectly satisfied this means that in all possible executions the honest parties never disagree on their outputs. On the other hand, in the statistical variant, there are certain executions where the honest parties are allowed to disagree. Nevertheless, these executions have an exponentially (or, alternatively, negligible) density in a security parameter κ among all executions. We observe that the statistical setting is only meaningful for a probabilistic consensus protocol, where the honest parties may be “unlucky” in their choices of coins.

2.4.2 Computational Security

In the computational setting the adversarial running time, and/or the computational model within which the adversary (and the parties running the protocol) are expressed becomes restricted. We distinguish the following variants.

One-way functions A standard computational assumption is the existence of one-way functions. A one-way function is a function $f : X \rightarrow Y$ for which it holds that f is polynomial-time computable, but the probability $\mathcal{A}(1^{|x|}, f(x)) \in f^{-1}(f(x))$ is negligible in $|x|$ for any polynomial time bounded program \mathcal{A} for x uniformly random. One-way functions, albeit quite basic, are a powerful primitive that enables the construction of more complex cryptographic algorithms that include symmetric-key encryption and digital signatures [NY89].

Proof of Work A proof of work (PoW) is a cryptographic primitive that enables a verifier to be convinced that certain amount of computational effort has been invested with respect to a certain context (e.g., a plaintext message or a nonce that the verifier/prover has provided). A number of properties have been identified as important for the application of the primitive including amortization resistance, sampleability, fast verification, hardness against tampering and message attacks, and almost k -wise independence [GKP17].

Some variants of PoWs have been shown to imply one-way functions [BGJ⁺16].

Chapter 3

Theory Results: Consensus

3.1 Consensus in the Standard Setting

In the traditional network model of point-to-point reliable channels between every pair of parties, the problem was formulated in [LSP82] considering the information-theoretic setting and the computational (also called *cryptographic*, or *authenticated*) setting. As mentioned above, in the former no assumptions are made about the adversary's computational power, while the latter relies on the hardness of computational problems such as factoring large integers or computing discrete logs (therefore the adversary's computational power needs to be restricted), and requires a trusted setup in the form of a PKI. Depending on the setting, some of the bounds on the problems' quality measures differ.

3.1.1 Number of Parties

Let n be the total number of parties involved in the execution of a protocol and let t be the number of dishonest parties. For the information-theoretic setting, Lamport *et al.* [LSP82] showed that $n > 3t$ is both necessary and sufficient for the problem to have a solution. The necessary condition is presented in [LSP82] for the broadcast problem (see [FLM86] for the consensus version of the impossibility result), as the special case of 3 parties ("generals"), having to agree on two values ('attack', 'retreat'), with one of them being dishonest. As in the information-theoretic setting (with no additional setup) the parties are not able to forward messages in an authenticated manner, it is easily shown that an honest receiver cannot distinguish between a run where the sender is dishonest and sends conflicting messages, and a run where a receiver is dishonest and claims to have received the opposite message, which leads to the violation of the problem's conditions (Agreement and Validity, respectively). The general case (arbitrary values of n) reduces to the 3-party case. The protocol presented in [LSP82] matches this bound ($n > 3t$), and essentially consists of recursively echoing messages received in a round while excluding the messages' senders. (In the first round, only the sender sends messages.) This is done for $t + 1$ rounds, at which point the parties take majority of the values received, and go back up the recursion. $t + 1$ rounds were later shown to be optimal (see below), but the protocol requires exponential (in n) computation and communication.

Lamport *et al.* [LSP82] also formulated the problem in the computational setting, where, specifically, there is a trusted private-state setup (of a PKI), and the parties have access to a digital signature scheme. This version of the problem has been referred to as *authenticated Byzantine agreement*. In contrast to the information-theoretic setting, in the computational setting with a trusted setup the bounds for broadcast and consensus differ: $n > t$ [LSP82] and $n > 2t$ (e.g., [Fit03]), respectively. The protocol presented in [LSP82] runs in $t + 1$ rounds but, as in the information-theoretic setting, is also exponential; an efficient (polynomial-time) was presented early on by Dolev and Strong [DS83], which we now briefly describe. In this protocol in the first round the sender digitally signs and sends his message to all the other parties, while in subsequent rounds parties append their signatures and forward the result. If any party ever observes valid signatures of the sender on two *different* messages, then

that party forwards both signatures to all other parties and disqualifies the sender (and all parties output some default message). This simple protocol is a popular building block in the area of cryptographic protocols (refer, however, to Section 3.1.6).

In the computational setting, the original formulation of the problem assumes a PKI. In [Bor96], Borderding considered the situation where no PKI is available, which he refers to as “local authentication,” meaning that no agreement on the parties’ keys is provided, as each party distributes its verification key by itself. Borderding shows that in this case, as in the information-theoretic setting above, broadcast and consensus are not possible if $n \leq 3t$, even though this setting is strictly stronger, as a dishonest party cannot forge messages sent by honest parties.

Regarding the “strong” version of the problem (the decision value must be one of the honest parties’ input values), Fitzi and Garay [FG03] showed that the problem has a solution if and only if $n > \max(3, |V|)t$ in the unconditional setting¹, where V is the domain of input/output values, and $n > |V|t$ in the computational setting with a trusted setup, giving resiliency-optimal and polynomial-time protocols that run in $t + 1$ rounds.

3.1.2 Round Complexity

Regarding the running time of consensus protocols, a lower bound of $t + 1$ rounds for deterministic protocols was established by Fischer and Lynch [FL82]; the lower bound was shown for the case of benign (“crash”) failures, and extended to malicious failures by Dolev and Strong [DS83]. As mentioned above, the original protocols by Lamport *et al.* already achieved this bound, but required exponential computation and communication. In contrast to the computational setting, where a polynomial-time resiliency- and round-optimal were found relatively soon [DS83], in the information-theoretic setting this took quite a bit longer, and was achieved by Garay and Moses [GM98]. In a nutshell, the [GM98] protocol builds on the “unraveled” version of the original protocol, presented and called *Exponential Information Gathering* by Bar-Noy *et al.* [BDDS92], applying a suite of “early-stopping” and fault-detection techniques to prune the tree data structure to polynomial size.

The above $t + 1$ -round lower bound applies to deterministic protocols. A major breakthrough in fault-tolerant distributed algorithms was the introduction of randomization to the field by Ben-Or [Ben83] and Rabin [Rab83], which, effectively, showed how to circumvent the above limitation by using randomization. Rabin [Rab83], in particular, showed that linearly resilient consensus protocols in expected *constant* rounds were possible, provided that all parties have access to a “common coin” (i.e., a common source of randomness). Essentially, the value of the coin can be adopted by the honest parties in case disagreement at any given round is detected, a process that is repeated multiple times. This line of research culminated with the work of Feldman and Micali [FM97], who showed how to obtain a shared random coin with constant probability from “scratch,” yielding a probabilistic consensus protocol tolerating the maximum number of misbehaving parties ($t < n/3$) that runs in expected constant number of rounds. The [FM97] protocol works in the information-theoretic setting; these results were later extended to the computational setting by Katz and Koo [KK06], who showed that assuming a PKI and digital signatures there exists an (expected-)constant-round consensus protocol tolerating $t < n/2$ corruptions.

Recall that deterministic broadcast protocols in the computational setting with setup tolerate an arbitrary number (i.e., $n > t$) of dishonest parties; in contrast, the protocol in [KK06] assumes $n > 2t$ (as it is based on VSS—*verifiable secret sharing* [CGMA85]). In [GKKO07], Garay *et al.* consider the case of a dishonest majority (i.e., $n \leq 2t$), presenting an expected-constant-round protocol for $t = \frac{n}{2} + O(1)$ dishonest parties (more generally, expected $O(k^2)$ running time when $t = \frac{n}{2} + k$), and showing the impossibility of expected-constant-round broadcast protocols when $n - t = o(n)$.

Regarding strong consensus, the $t + 1$ -round lower bound also applies to this version of the problem, which the protocols by Fitzi and Garay [FG03] achieve (as well as being polynomial-time and resiliency-optimal).

¹The lower bound was in fact shown by Neiger, who formulated this version of the problem [Nei94].

3.1.3 Trusted Setup

We already covered this aspect above while describing the protocols achieving the different bounds on the number of parties; here we briefly summarise it. There is no trusted setup in the unconditional setting, although in the case of randomized protocols there is the additional requirement of the point-to-point channels being private in addition to reliable, while the “authenticated” consensus protocols assume a PKI. Related to a trusted setup assumption, we remark that if a *pre-computation* phase is allowed in the information-theoretic setting where reliable broadcast is guaranteed, then Baum-Waidner, Pfitzmann and Waidner showed that broadcast and consensus are achievable with the same bounds on the number of parties as in the computational setting, using a tool known as “pseudo-signatures” [BPW91].

3.1.4 Communication Cost

A lower bound of $\Omega(n^2)$ on the number of messages (in fact, $\Omega(nt)$) was shown by Dolev and Reischuk for consensus for both information-theoretic and computational security [DR85]; for the latter, what they showed was that the number of signatures that are required by any protocol is $\Omega(nt)$, resulting in an $\Omega(nt|\sigma|)$ bit complexity (for a constant-size domain), where $|\sigma|$ represents the maximum signature size. The first information-theoretically secure protocols to match this bound were given by Berman *et al.* [BGP92] and independently by Coan and Welch [CW89]; regarding computational security, the protocol presented by Dolev and Strong [DS83] requires that many messages.

3.1.5 Beyond Synchrony

The case of partial synchrony introduced in [DLS88], considers the existence of an unknown bound Δ that determines the maximum delay of a message that is unknown to the protocol participants.² As shown in [DLS88], the resiliency bounds remain unaltered.

In the eventual delivery setting, as mentioned above, deterministic consensus is impossible but it is still feasible to obtain protocols with probabilistic guarantees. Furthermore, note that it is not possible in this setting to account for all the honest parties’ inputs since parties cannot afford to wait for all the parties to engage (since corrupt parties may never transmit their messages and it is impossible to set a correct time-out). In more detail, without a setup in the information-theoretic setting, it is possible to adapt [FM97] and achieve $n/4$ resiliency cf. [FM88]. Moreover, by allowing the protocol not to terminate with negligible probability it is possible to bring the resiliency down to $n/3$, [CR93, ADH08]. In the private-setup setting, assuming one-way functions, it is possible to obtain an always terminating protocol with $n/3$ resiliency, cf. [FM88]. We note that it is infeasible to go beyond $n/3$ resiliency (cf. [Can96] where they argue this bound for fail-stop failures) and thus the above results are optimal in this sense.

3.1.6 Property- vs. Simulation-Based Proofs

As mentioned in Section 2.1.2, consensus and broadcast protocols have been typically proven secure/correct following a *property-based* approach. It turns out, as pointed out by Hirt and Zikas [HZ10] (see also [GKKZ11]), that in the case of *adaptive* adversaries who can choose which parties to corrupt dynamically, during the course of the protocol execution (cf. [CFGN96]), most existing broadcast and consensus protocols cannot be proven secure in a simulation-based manner. The reason, at a high level, is that when the adversary (having corrupted a party) receives a message from an honest party, can corrupt that party and make him change his message to other parties. This creates an inconsistency with the ideal process, where the party has already provided his input to the trusted party/ideal functionality. To be amenable to a simulation-based proof, instead of sending its initial message “in the clear,” the sender in a broadcast protocol sends a *commitment* to the message, allowing

²In [DLS88] partial synchrony between the clocks of the processors is also considered as a separate relaxation to the model. In the present treatment we only focus on partial synchrony with respect to message passing.

the simulator in the ideal process to “equivocate” when the committed value becomes known in case the party has been corrupted and the initial value changed [HZ10, GKKZ11].

3.2 Consensus in the Peer-to-Peer Setting

Consensus in the peer-to-peer setting refers to the consensus problem when the available communication model is peer-to-peer diffusion (cf. Section 2.2.1), a weaker communication primitive compared to point-to-point channels. This setting arose with the advent of the Bitcoin blockchain protocol, and was formally studied for the first time in [GKL15]. It is epitomised by an unauthenticated model of communication where no correlation of message sources across rounds takes place and the total number of parties that participate in the protocol may be unknown to the protocol participants. Moreover, since the adversary may inject messages in the network, an honest party cannot infer the number of participants from a message count. Because of this uncertainty, the consensus properties of termination, agreement and validity hold for honest parties that are participating in the protocol.

We note that in a precursor model, where there is no correlation of message sources, but the point-to-point structure is still in place albeit without authentication, Okun showed that deterministic consensus algorithms are impossible for even a single failure [Oku05b, Oku05a], but that probabilistic consensus is still feasible by suitably adapting the protocols of [Ben83, FM97]³; the protocol, however, takes exponentially many rounds.

The consensus problem in the peer-to-peer setting has mostly been considered in the computational setting utilising one-way functions and the proof-of-work primitive.

The first suggestion for a solution was informally described in [AJK05], where it was suggested that PoW can be used as an identity assignment tool, which subsequently can be used to bootstrap a standard consensus protocol like [DS83]. Nevertheless, the viability of this plan was never fully analysed until an alternative approach to the problem was informally described by Nakamoto in an email exchange [Nak08b], where he argued that the “Byzantine Generals” problem can be solved by a blockchain/PoW approach tolerating a number of misbehaving parties strictly below $n/2$. As independently observed in [ML14, GKL15], however, with overwhelming probability the Validity property is not satisfied by Nakamoto’s informal suggestion.

The blockchain approach suggests to string PoWs together in a hash chain and obtain agreement using a rule that favors higher concentrations of computational effort as reflected in the resulting hashchains. The inputs to the consensus are entangled within the PoWs themselves and the final output results from a processing of the hash chain. The approach was first formalised in [GKL15] where also two constructions were provided that satisfy all properties assuming a public setup.

Without access to a public setup, it is also possible to obtain a construction based on the results of [AD15] who were the first to formalise the [AJK05] informal approach of using PoWs to identity assignment. Moreover, a blockchain-based approach is also possible as shown in [GKLP18].

Using a private setup, it becomes feasible to use primitives such as digital signatures and verifiable random functions and obtain even more efficient constructions such as the consensus sub-protocol of [Mic16].

3.2.1 Number of Parties

In the convention introduced in [GKL15], each party has a fixed quota of hashing queries that is allowed per round. As a result, the number of parties is directly proportional to the “computational power” that is present in the system and the total number of PoWs produced by the honest parties collectively would exceed that of the adversary assuming honest majority with very high probability.

The main problem in establishing identities using PoW, is that the set of identities as perceived by the honest participants in the protocol execution might be inconsistent. This was resolved with the protocol of [AD15] where PoWs are used to build a “graded” PKI, where keys have ranks. The graded PKI is an instance of a graded agreement [FM97], or partial consistency problem [CFF⁺05], where honest parties do not disagree by much,

³Hence, consensus in this setting shares a similar profile with consensus in the asynchronous network model [FLP85].

according to some metric. Subsequently, it is possible to morph this graded consistency to global consistency by running multiple instances of [DS83]. This can be used to provide a consensus protocol with resiliency $n/2$ without a trusted setup.

It is unnecessary though for the parties to reach consensus by establishing identities. In the first protocol presented in [GKL15], the parties build a blockchain where each block contains a value that matches the input of the party that produced the block. The protocol continues for a certain number of rounds that ensures that the blockchain has grown to a certain length. In the final round, the parties remove a k -block suffix from their local blockchain, and output the majority bit from the remaining prefix. Based on the property called “common prefix” in [GKL15], it is shown that with overwhelming probability in the security parameter, the parties terminate with the same output, while using the “chain quality” property, it is shown that if all the honest parties start with the same input, the corrupt parties cannot overturn the majority bit, which corresponds to the honest parties’ input. The number of tolerated misbehaving parties in this protocol is strictly below $n/3$, a sub-optimal resiliency due to the low chain quality of the underlying blockchain protocol. We note that the maximum resiliency that can be expected is also $n/2$. Optimal resiliency can be reached by the second consensus protocol of [GKL15] as follows: the protocol substitutes Bitcoin transactions with a type of transactions that are *themselves* based on PoWs, and hence uses PoWs in two distinct ways: for the maintenance of the ledger and for the generation of the transactions themselves. The protocol requires special care in the way it employs PoWs since the adversary should be incapable of “shifting” work between the two PoW tasks that it faces in each round. To solve this problem, a special strategy for PoW-based protocol composition is introduced in [GKL15] called “2-for-1 PoWs.” In the second solution presented in [GKL15] the number of tolerated misbehaving parties is strictly below $n/2$.

We note that all these protocols come with a hard-coded difficulty level for PoWs which is assumed to be correlated to the number of parties n . If f is the probability that at least one honest party will produce a PoW in a round of protocol execution, it holds that f approaches 0 for small n while it approaches 1 for large n . It follows that the choice of PoW difficulty results in an operational range of values $[n_L, n_R]$, so that when if $n < n_L$, the protocol cannot be guaranteed to satisfy validity with high probability while if $n > n_R$, the protocol cannot be guaranteed to produce agreement with high probability.

Using digital signatures and verifiable random functions (VRFs), or just digital signatures and a hash function modeled as a random oracle, it is possible to implement the second consensus protocol [GKL15] over an underlying blockchain protocol that uses a public-key infrastructure as opposed to PoW, and allows for arbitrary participation such as [PS17] for optimal resiliency of $n/2$. The idea is as follows: one can use VRFs for each participant to enable a random subset of elected transaction issuers in each round. The ledger will then incorporate such transactions within a window of time following the same technique and counting argument as in the second consensus protocol of [GKL15].

3.2.2 Running Time

In order to measure the running time that the protocols require in the peer-to-peer setting assuming PoW, one will have to also take into account that periods of silence, i.e., rounds without any message passing, may also be required for ensuring the required properties with high probability in κ , a security parameter.

The consensus protocol derived by [AD15] requires $O(n)$ rounds where n is the number of parties. This can be improved to $O(\kappa)$, by e.g., using a blockchain-based approach [GKLP18]. In the public-setup setting, assuming that the number of parties fall within the operational range, the protocols of [GKL15] run in time $O(\kappa)$.

It is worth noting the contrast to the approach used in randomized solutions to the problem in the standard setting (cf. Section 3.1.2), where achieving consensus is reduced to (the construction of) a shared random coin, and comparable guarantees are obtained after a poly-logarithmic number of rounds in the number of parties. The probabilistic aspect in the blockchain setting stems from the parties’ likelihood of being able to provide proofs of work.

In the private setup setting it is possible to improve the running time to expected constant, either using the protocol of [KK06] for optimal resiliency, or for an even smaller constant deploy the consensus sub-protocol of Algorand [Mic16] for $1/3$ resiliency (while it is claimed that $n/2$ resiliency is also attainable).

3.2.3 Trusted Setup

The relevant trusted setup assumption in the above protocols include a fresh random string, that can be incorporated as part of a “genesis block” in the blockchain protocol setting, or in general as part of the PoWs⁴. The objective of this public setup is to prevent a pre-computation attack by the adversary that will violate the relative superiority of honest parties which would be derived by the honest majority assumption.

Note that protocols that require no trusted setup such as [AD15,GKLP18] take advantage of a special randomness exchange phase prior to PoW calculation that facilitates freshness without the need of a common random string.

It is worth to emphasise the fundamental advantage of the PoW setting compared to other computational assumptions that have been used for consensus. Specifically, it is known that without a private setup consensus is not possible with more than $n/3$ corruptions [Bor96] even assuming digital signatures. The $n/3$ impossibility result does not apply since, for instance, proof of work can make it infeasible for the adversary to present diverging protocol transcripts without investing effort for distinct PoW calculations.

3.2.4 Communication Cost

The total number of transmitted messages in the consensus protocols described above is, in expectation, $O(n^2\kappa)$ for the case of [AD15,GKLP18] counting each invocation of the diffuse channel as costing $O(n)$ messages. For the two protocols of [GKL15] the number of messages is $O(n\kappa)$ in the public setup setting. Assuming a private setup, [Mic16] can bring this further down to $O(n)$.

We recall that an important difference with randomized consensus protocols in the standard setting is parties send messages in every round, while in the PoW setting parties only communicate in a round when they are able to produce a proof of work; otherwise, they remain silent. This also suggests that there may be honest parties that never diffuse a message and thus it is feasible to drop communication cost to below n^2 (with a probabilistic guarantee), cf. Section 3.1.4.

3.2.5 Property- vs. simulation-based proofs

To our knowledge, there is no simulation-based treatment of consensus in the peer-to-peer setting, however it is easy to infer a functionality abstracting the problem. The only essential difference is that the actual number of parties involved in the execution are to be decided on the fly and will be unknown to the protocol participants.

3.3 Ledger Consensus

Ledger consensus (a.k.a. “Nakamoto consensus”) is the problem where a set of servers (or nodes) operate continuously accepting inputs that are called transactions and incorporate them in a public data structure called the *ledger*. Clients are able to read the ledger and submit transactions to be added to it. The purpose of ledger consensus is to provide a unique view to the ledger for the clients.

The properties that a ledger consensus protocol must satisfy are as follows:

- *Consistency*: This property mandates that if a client queries an honest node’s ledger at round t_1 and receives the response \mathcal{L}_1 , then a client querying an honest node’s ledger at round $t_2 \geq t_1$ receives a response \mathcal{L}_2 that satisfies $\mathcal{L}_1 \preceq \mathcal{L}_2$, where \preceq denotes the standard prefix operation.
- *Liveness*: If a transaction tx is given as input to all honest nodes continuously for a certain number of rounds denoted by u , and a client queries any honest node’s ledger after these u rounds, then the node responds with a value \mathcal{L} that includes tx .

⁴Alternatively, the protocols would consider as valid any chain that extends the empty chain, and where the adversary is not allowed any pre-computation.

In the traditional distributed systems literature, such a problem is often referred to as *state machine replication* [Sch90]. Consistency ensures that parties have the same view of the log of transactions, while Liveness ensures the quick incorporation of transactions.

Reducing ledger consensus to consensus. Given a consensus protocol it is tempting to apply it in sequential composition in order to solve ledger consensus. The reduction indeed holds but some special care is needed. First let us consider the case where no setup is available. The construction in the synchronous network model is as follows. First, suppose that we have at our disposal a consensus protocol that satisfies Agreement, (Strong) Validity, and Termination after u rounds. The protocol has all nodes collect transactions and then run the consensus protocol with the set of transactions as their input. When the protocol terminates after u rounds, the nodes assign an index to the output (call it the i -th entry to the ledger) and move on to the next consensus instance. It is easy to see that Consistency is satisfied because of Agreement, while Liveness is satisfied with parameter u because of Strong Validity and Termination.

It is worth noting that “plain” Validity by itself is not enough, since a ledger protocol is supposed to run for any given set of transactions and as a result it is possible that no two honest nodes would ever agree on a set of inputs. In this case, Validity might just provide that honest parties’ agree on an adversarial value, which might be the empty string. As a result the ledger would be empty and Liveness would be violated. However it is possible to deal with this problem without resorting to the full power of strong validity. In particular, it is sufficient to consider a variant of consensus where each party has an input set X_i and the joint output set S satisfies that $X_i \subseteq S$. We note that such a “union” consensus protocol can be implied directly by interactive consistency, [PSL80], and it has also recently been considered explicitly as a consensus variant, [DG17].

Let us now comment how the reduction can be performed under different setup and network assumptions. First, if a setup assumption is used, observe that the above reduction requires the availability of the setup every u rounds. Given this might be impractical, one may consider how to emulate the sequence of setups using a single initial setup. This approach is non-black-box on the underlying protocol and may not be straightforward. For instance, when sequentially composing a PoW-based consensus protocol that relies on a public setup, the security of the protocol may non-trivially rely on the unpredictability of the i -th setup, during the execution of the first i instances. Techniques related to sequential composition of a basic building block protocol have appeared in a number of ledger protocols, including [KRDO17, BPS16, GHM⁺17]. Regarding network aspects, we observe that the reduction can proceed in essentially the same way in the peer-to-peer setting as in the point-to-point setting.

3.3.1 Number of Parties

As in the case of the consensus problem, assuming no other identity infrastructure, each party has a fixed quota of hashing queries that it can perform to a hash function per unit of time and thus the number of parties is directly proportional to the total computational or hashing power that is available. In this setting, first [GKL15] showed that ledger consensus can be achieved when the number of corrupted parties is strictly below $n/2$. This bound was also preserved in the partially synchronous setting, as shown by Pass *et al.* [PSS17]. With respect to lower bounds, it is not difficult to see that $n/2$ is a tight bound.

In this context, the relevant concept of “sporadic participation” was put forth in [PS17], which in turn is related to models that have been considered in other protocol settings such as fault-tolerant distributed computation (see e.g., “omission failure” [Lyn96]) and secure multiparty computation [HLP11]. In this scenario in some sense the parties are subject to two types of faults, Byzantine and benign, such as “going to sleep,” but adversarially scheduled. In the latter type, the parties cease participating in the protocol execution. The parties that have been offline can recover and rejoin the protocol (with potentially an outdated state). This type of corruption is cast as “sleep” in [PS17] and the question is posed whether it is possible to perform ledger consensus in a setting where the number of sleep corruptions exceeds $n/2$. Assuming a setting where the adversary gets t Byzantine corruptions and s sleep corruptions it is shown that ledger consensus can be achieved as long as t is strictly bounded by $(n - s)/2$, i.e., the number of sleep corruptions may be larger than $n/2$. It is interesting to point

out that even though the results of [GKL15,PSS17] are not explicitly dealing with sleep corruptions the security arguments can be directly ported to this setting by assuming as before that the number of malicious parties are less than $(n - s)/2$.

With respect to lower bounds, in the case of sleep corruptions the bound can be generalized to $(n - s)/2$; see [PS17] for a formal proof.

The above results refer to a static setting where the number of parties are fixed throughout the execution. The setting where the population of parties running the protocol is dynamically changing over time with the environment introducing new parties and deactivating parties that have participated has been considered for ledger consensus for the first time in [GKL17]. The main result is that ledger consensus can be achieved in the proof-of-work setting, assuming an honest majority; expectedly, honest majority here has to be restated by considering the number of parties as they change over time: assuming n_i are the active parties at time unit i , it holds that the number of adversarial parties is bounded away from $n_i/2$. Even though the sleep corruption setting has not been explicitly considered here, it is expected that it could be applied without major modifications, and in such case, the corresponding bound would be amended to $(n_i - s_i)/2$ where s_i equals the number of parties that have been subjected to a sleep corruption at time unit i . A dynamic setting of parties was also considered in [KRDO17,BPS16,GHM⁺17], providing a similar type of results under the assumption of an initial public-key directory with honest “stake” majority. In [BGK⁺18] the authors consider a more refined, UC-based setting, of dynamic availability (see 4.3.2 for more details).

3.3.2 Transaction Processing Time

Contrary to consensus, the running time of the protocol is not immediately relevant, as a ledger consensus protocol is a protocol that is supposed to be running over an arbitrary, potentially long, period of time. Nevertheless, a related measure is the length of time that it takes for the system to insert a certain transaction in the log maintained by the participants.

This relates to the parameter u introduced as part of the Liveness property, which determines the number of rounds that need to pass in the execution model for a transaction to be included in the log. Observe that Liveness is only provided for transactions that are produced by honest participants or are otherwise unambiguously presented to the honest parties running the protocol.

In this setting we observe that [GKL15] provides a ledger with processing time $O(\kappa)$ rounds of interaction where κ is the security parameter. This result is replicated in the partially synchronous setting, where processing time takes $O(\kappa\Delta)$ rounds, and where Δ is the maximum delay that is imposed on message transmission.

The above results assume the adversarial bounds consistent with honest majority which are tight. Considering a weaker adversarial setting it is possible to improve Liveness, as recently shown in [PS18], where it is shown that processing time can be dropped to $O(1)$ rounds, assuming an honest super-majority (adversary strictly below $n/4$) and a certain specific party called the *accelerator* to be honest.

3.3.3 Trusted Setup

Ledger consensus can be achieved in the public-state or private-state setup setting. The results operating in the former setting are [GKL15,PSS17,GKL17], whereas the results consistent with the latter are [PS17,KRDO17,BPS16,GHM⁺17]. In the absence of any trusted setup it has been shown that it is possible to “bootstrap” a ledger consensus protocol from “scratch,” either directly [GKLP18] or via setting up a public-key directory using proofs of work [AD15]. Note that these protocols work without any setup, nevertheless, they do contain hardcoded information that determines the difficulty level of the underlying proof of work primitive that is employed. In the case that the actual number of parties present in the protocol execution substantially deviates from the estimation that is implied by the difficulty level, the protocol security degrades and it can dissipate entirely.

An important further consideration between public setup and private setup is that, in the peer-to-peer setting, the former represents what typically is consistent with the so-called permissionless setting, while the latter is consistent with the so-called permissioned setting. This follows from the fact that anyone that has access to the

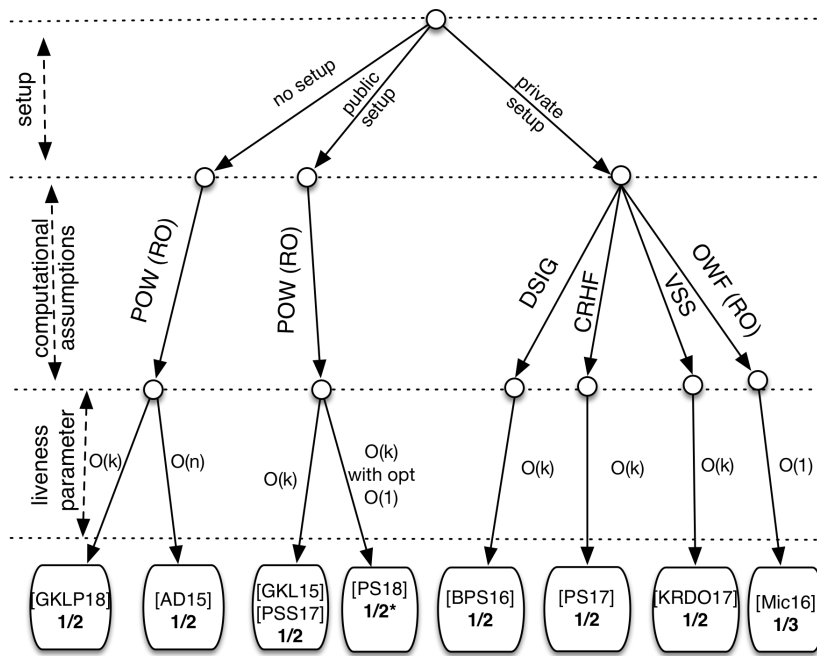


Figure 3.1: Ledger consensus tree of protocols.

peer-to-peer channel is free to participate in the protocol, if no setup or a public setup is assumed. On the other hand, in the private setup setting, a higher level of permissioning is implied: the parties that are eligible to run the protocol need to get authorised either by the setup functionality so they receive the private information that is related to the protocol execution, or, alternatively, interact with the parties that are already part of the protocol execution so they can be inducted.

3.3.4 Beyond Synchrony

The setting of partial synchrony for ledger consensus was first considered in [PSS17]. The analysis of [GKL15] carries to the partial synchronous setting and it can be shown that ledger consensus is achieved in the partial synchronous setting with public setup assuming honest majority in terms of computational power.

Chapter 4

Notable Systems

4.1 PoW-based Ledgers

Chapter 3 discussed Nakamoto’s consensus protocol that relies on proofs of work to guarantee consistency between potentially different local copies of a distributed ledger. In this section we give a high-level overview of famous distributed ledgers that rely on this consensus protocol.

4.1.1 Bitcoin

Bitcoin is a cryptocurrency designed by Satoshi Nakamoto [Nak08a] for peer-to-peer (P2P) networks.

High-level overview. A user in Bitcoin can have multiple unlinked public keys of a signature scheme that correspond to digital wallets. A user makes a payment (called *transaction*) transferring coins from one of his public keys (corresponding to a wallet with enough coins) to other public keys. In order to make such a payment valid the user signs the transaction using the secret key associated to his public key.

Some special users called *miners* are in charge of verifying transactions. Once a transaction is successfully verified by a miner, she adds the transaction to a *block*. As soon as the miner finishes the procedure to generate a new block she announces it in the Bitcoin network hoping that it will be confirmed by other miners. Indeed, miners receiving announcements of new blocks check their correctness and then append them to their local copies of a public decentralized ledger called *blockchain*. A miner that proposes a block that is later on added to the blockchain receives a reward.

The blockchain is therefore a sequence of blocks where each block refers to the previous one, except the very first block that is called the *genesis block*. The consistency between potentially different local copies of a blockchain is implemented using Nakamoto’s consensus protocol (see Chapter 3).

The output of a transaction is the amount of coins sent to a Bitcoin wallet and is called “unspent transaction output” (UTXO). In Bitcoin the balance of a wallet is computed by summing up all UTXOs sent to the wallet that have not been spent yet.

Details on Bitcoin blockchain. A block in Bitcoin contains: 1) a reference to the previous block (i.e., the hash value of the previous block in the blockchain (h_{pb})); 2) a set of transactions; 3) the hash value of the transactions belonging to the block (h_{tx})¹. A solution to a PoW puzzle consists of a value called *nonce* s.t. $H(h_{pb}||h_{tx}||nonce) < target$, where H is a cryptographic hash function² and *target* is a 256-bit number that is known by all the miners of the Bitcoin network. Assuming that the cryptographic hash function behaves as a random oracle, there is no way to solve the puzzle better than brute-force trying all possible values of the nonce

¹To be more precise, the transactions are organized in a Merkle tree and the root of the Merkle tree contributes to form the challenge for the PoW puzzle.

²Bitcoin uses SHA-256 as cryptographic hash function.

until a value that satisfies the above equation is found. The difficulty of finding a solution for a PoW puzzle depends on the target value. Indeed, for a lower value of the target there exists a smaller number of possible solutions of the PoW puzzle, therefore finding a solution becomes more difficult. In Bitcoin every 2016 blocks the users update the target value in order to make the PoW puzzle more or less difficult. In this way it is possible to control the hardness to solve a PoW puzzle and in turn the time to generate a new block. In Bitcoin a new block is generated roughly every 10 minutes and the hardness of the PoW puzzle is adjusted to maintain this time interval while the overall computational power of the miners changes.

Security. The consistency of the blockchain of Bitcoin (and in turn the double-spending prevention) relies on the assumption that honest miners have the majority of the computational power.

In literature various works formalize and study the security of Bitcoin, see [GKL15, PSS17, BMTZ17, GKL17]. Recently it has been shown that a rational miner (i.e., a miner that behaves to maximize his utility) ends up behaving as an honest miner because this strategy gives him the best utility. This was formally proved in [BGM⁺18], that also takes into account previous attacks on Bitcoin (e.g., the selfish-mining attack [ES14]) at backbone level (i.e., assuming that miners are only interested in maximizing their revenues by mining).

DoS attacks are in general possible by generating a huge amount of transactions. The existence of transaction fees strongly mitigates such issue. A transaction fee corresponds to some coins transferred by the user that proposes a transaction to the miner. Notice that at some point the actual reward for miners will consist of transactions fees only since the minting of coins produced by the generation of the blocks reduces over the time.

4.1.2 Ethereum

Ethereum is a decentralized platform based on a P2P network that runs programs called *contracts*. It was proposed by Vitalik Buterin [But13].

High-level overview. A contract is run on every node of the Ethereum network and can compute functions with various purposes (e.g., data storage, coin transfers). More specifically, a contract is a deterministic program and is defined via a Turing-complete bytecode language, called EVM bytecode [But13].

A user requesting the execution of a smart contract must pay some coins called *ethers*. At a very-high level a user in Ethereum can request three possible types of operations, called transactions, through one of her public keys corresponding to digital wallets: (i) creation of a new contract; (ii) execution of a function of a contract; (iii) transfer of ethers to contracts or to other users.

Unlike Bitcoin that adopts UTXO, in Ethereum each public key has a balance that is divided in debits and credits. The difference between credits and debits must always be non-negative, and therefore a transaction that would create a negative balance is refused. To prevent double spending in Ethereum each account is associated to a value called *account nonce*. Every time an user creates a new transaction the corresponding account nonce is incremented by one and added to the transaction. An attempt to spend twice the same coins requires to use twice the same nonce, but in this case only one of two transactions with the same nonce will be accepted, therefore avoiding the double spending.

The UTXO model is more scalable than the balance model since multiple UTXOs can be processed at the same time. Moreover the UTXO model does not require the management of additional information (like account nonce) to prevent double spending. On the other hand the balance model is more intuitive and easier to manage for developers. Furthermore, the balance model is more efficient since a transaction can be validated by checking the balance of the sending account only. This contrasts with the UTXO model where instead all previous transactions concerning the same wallet must be checked.

The blockchain and the consensus protocol. Ethereum inherits from Bitcoin several mechanisms to process a transaction. In particular, transactions are memorized in a block by a miner that at some point announces a block

to other miners who verify its validity and then add it to the blockchain. Similarly to Bitcoin, the consistency is preserved by relying on proofs of work.

Ethereum decreases the time needed to create a new block using the following mechanism. When two blocks are announced by two different miners at the same time just one of them is added in the blockchain. However, unlike Bitcoin, the block that is not successfully added is not discarded, and can be added later. The miners that create or reference this type of block are also rewarded. As a result, in Ethereum a new block is added to the blockchain every 14/15 seconds.

Security. Being based on proofs of work, the unforgeability of the blockchain of Ethereum’s relies on the assumption that the majority of the computational power of the miners is used honestly. Transaction fees defeat DoS attacks and are calculated on the amount of resources (i.e., *Gas*) needed to compute the transaction. In more details, the “gas” is an internal measure of Ethereum that is used to estimate the computational resources, bandwidth and storage required to compute a transaction.

In Ethereum the contracts are published in the blockchain in an immutable way, therefore it can be very difficult to fix possible issues of the contracts. One of the most famous attacks exploiting the vulnerability of contracts is the DAO attack [Sie16]. After this attack an upgraded version of the Ethereum system has been released (hard-fork) in order to nullify the effects of the transactions involved in the attack. The DAO attack was just an example of the attacks that have been mounted to steal coins from the contracts, there are several other cases that can be found in the literature [LJC⁺18, ABC17].

4.2 Ledgers Based on Byzantine-Fault Tolerance (BFT)

4.2.1 BFT Consensus Protocols

Many distributed ledger systems realizing *permissioned* blockchains rely on classical protocols for consensus or state-machine replication. Their defining characteristic is to rely on distinct node identities and a resilience assumption (typically) stated in terms of a number of correct nodes. These protocols are usually referred to as *Byzantine-fault tolerant (BFT) consensus*. The most prominent protocol in this model, and the one that lended it the name, is *Practical Byzantine Fault Tolerance (PBFT)* by Castro and Liskov [CL02]. PBFT can be understood as an extension of the Paxos [Lam98]/Viewstamped Replication [OL88] protocol family for consensus tolerating crashes [Lam01, Lis10, Cac09]. In a system with n nodes, PBFT and its variants tolerate $f < n/3$ Byzantine-faulty nodes, which is optimal. BFT consensus works in the so-called *partial or eventual synchrony model* [DLS88] discussed in Chapter 2, where synchrony is assumed to hold after a point in time unknown to the protocol participants.

After the publication of the PBFT paper in the *computer-systems* research literature, many groups and authors in this field followed up and presented BFT consensus protocols and systems during the years 2000-2010. The comprehensive paper by Aublin et al. [AGK⁺15] summarizes many of them and provides a modular way to understand them and to construct many further protocols.

Only very few actual systems that implement PBFT or one of its variants have been produced before the interest in permissioned blockchains surged around 2015 [Swa15]. *BFT-SMaRt* [BFT18] is the most complete project of this kind and available as open source, initiated by Bessani et al. [BSA14, SB12]. There is widespread agreement today that BFT-SMaRt is the most advanced and most widely tested implementation of a BFT consensus protocol available. Experiments have demonstrated that it can reach a throughput of about 80’000 transactions per second in a LAN with 4 nodes [BSA14], degrading with growing number of nodes to about 55’000 transactions per second for 10 nodes, and provides very low latency overhead in a WAN [SB15].

4.2.2 Blockchain systems with BFT and BFT-like consensus

This section presents a list of blockchain systems that aim at using BFT consensus. We only describe a small, subjective selection of the plethora of protocols and systems that have been proposed by players in the vibrant

blockchain industry.

Hyperledger Fabric. *Hyperledger Fabric* [Hyp18a] is a platform for distributed ledger solutions, written in Golang and with a modular architecture that allows multiple implementations for its components. Following “preview” releases (*v0.5* and *v0.6*) in 2016, whose architecture [Cac16] directly conforms to state-machine replication, a different and more elaborate design was adopted later for version 1. This revised architecture [ABB⁺18] separates the execution of smart-contract transactions (in the sense of validating the inputs and outputs of a program) from ordering transactions for avoiding conflicts (in the sense of an atomic broadcast that ensures consistency).

The consensus protocol up to release *v0.6* was a native implementation of PBFT [CL02]. In version 1.0 (released during 2017) the *ordering service* responsible for conflict-avoidance is provided by an *Apache Kafka* cluster [Apa18]. In the form of an early prototype, the PBFT protocol provided by the BFT-SMaRt toolkit has been integrated in Fabric as an ordering service as well [SBV17]. A native implementation of the ordering service using PBFT is currently under development and targeted for release in 2018.

Tendermint. *Tendermint Core* [Ten18] implements a variant of PBFT [CL02]. In contrast to PBFT, where the client sends a new transaction directly to all nodes, the clients in Tendermint disseminate their transactions to the validating nodes using a gossip protocol. In addition, the protocol entails the continuous rotation of the leader. Namely, the leader is changed after every block, a technique first used in BFT consensus space by the *Spinning* protocol [VCBL09]. This means that Tendermint embeds aspects of PBFT’s view-change mechanism into the common-case pattern.

Hyperledger Indy. Hyperledger Indy is a distributed ledger platform, purpose-built for decentralized identity management [Hyp18b]. It provides tools for managing independent digital identities rooted on blockchains and focuses on the use of privacy-preserving cryptographic technologies.

In its current form, Hyperledger Indy contains the *Plenum Byzantine Fault Tolerant Protocol*, which follows a BFT protocol called *Redundant Byzantine Fault Tolerance (RBFT)* developed by Aublin et al. [AMQ13]. RBFT departs from PBFT in the following way. PBFT uses a special node, called the “primary,” for indicating to other nodes the order in which requests should be processed. In order to avoid that the primary becomes a bottleneck, RBFT executes multiple instances of the protocol simultaneously, a “master” instance and one or more “backup” instances. All the instances order the requests, but only the requests ordered by the master instance are actually executed. Hence the protocol adds redundancy in a way that differs from PBFT.

Kadena/Juno. *Juno* from *kadena* [Kad16] is a platform for running smart contracts that has been developed until about November 2016 according to its website. Juno claims to use a “Byzantine Fault Tolerant Raft” protocol for consensus and appears to address the standard BFT model with n nodes, $f < n/3$ Byzantine faults among them, and eventual synchrony [DLS88] as timing assumption. Later Juno has been deprecated in favor of a “proprietary BFT-consensus protocol” called *ScalableBFT* [Mar16], which is “inspired by the Tangaroa protocol” and optimizes performance compared to Juno and Tangaroa. The whitepaper cites over 7000 transactions per second (tps) throughput on a cluster with size 256 nodes.

The design and implementation of ScalableBFT are proprietary and not available for public review. It is claimed to use Tangaroa, a BFT protocol sketch floating on the web, which is flawed, however [CV17]. The soundness of this protocol therefore remains an open question.

Symbiont Assembly. *Symbiont Assembly* [Sym18] is a proprietary distributed ledger platform. The company that stands behind it, Symbiont, focuses on applications of distributed ledgers in the financial industry, providing automation for modeling and executing complex instruments among institutional market participants.

Assembly implements resilient consensus in its platform based on the BFT-SMaRt toolkit [BSA14]. However, Symbiont uses its own reimplementations of BFT-SMaRt in a different programming language; it reports

performance numbers of 80'000 transactions per second (tps) using a 4-node cluster on a LAN. This matches the throughput expected from BFT-SMaRt and similar results in the research literature on BFT protocols [AGK⁺15].

Chain. The *Chain Core* platform [Cha18] is a generic ledger infrastructure for an institutional consortium to issue and transfer financial assets on permissioned blockchain networks. The *Federated Consensus* [Cha17] protocol of Chain Core is executed by the n nodes that make up the network. One of the nodes is statically configured as “block generator.” This approach, however, provides only a rudimentary form of consensus, much weaker than BFT consensus and much easier to implement. In particular, the static block generator can be seen as the “leader” in a PBFT-style consensus protocol. To the extent visible in the published documents, Chain Core has no provisions for tolerating a faulty block generator, i.e., if that node misbehaves, it could violate safety or liveness of the system and manual restart would be needed.

Chain is mentioned here as only one example of many more protocols that aim at implementing BFT consensus, but fail to do so. Cachin and Vukolic [CV17] discuss some of them.

Corda. *Corda* [Cor18] is a distributed-ledger platform that is targeted at asset transfers and departs significantly from the design of a blockchain in that there is not a single representation of the ledger that is shared between all nodes. Instead, at each point in time, each asset is associated with one *notary* that is required to participate in asset transfers and also prevents double spending; the participants in the protocol for transferring an asset are the old and new owner of the asset and the notary. Notaries can in turn be instantiated by a single node, or a crash-fault tolerant or even Byzantine-fault tolerant protocol, where the implementation of the latter one is based on BFT-SMaRt [BSA14]. Corda natively supports so-called *notary-change transactions*, in which an asset is transferred from the custody of one notary to another, and which correspond to cross-chain transactions in blockchain systems.

Ripple. The Ripple blockchain [Rip18] (i.e., the “XRP Ledger” that holds the corresponding cryptocurrency) and its consensus protocol [SYB17] introduced the idea of flexible trust assumptions, which go beyond the typical BFT model in the sense that each participating node declares a list of nodes that it subjectively trusts. In contrast, the “trust sets” of standard BFT systems are global and must be the same for all nodes. The Ripple protocol itself is not based on the research literature; an independent investigation showed that the original specification was flawed [AKM⁺15]. A recent analysis [CM18], by authors from Ripple, reveals furthermore that it contains unnecessary steps and violates safety whenever the trust sets of nodes diverge. There are many claims in the blockchain industry that the Ripple protocol implements BFT consensus, but these works demonstrate that this view is wrong.

Stellar. The Stellar blockchain [Ste18] grew out of Ripple and uses a protocol with a similar goal [Maz16]: Each node declares on its own which other nodes it trusts, instead of a global assumption on which node collusions the protocol tolerates.

Examples in the documentation and the white paper [Maz16, Fig. 3] suggest the use of hierarchical structures with different groups organized into multiple levels, where a different threshold may exist for each group (but the “threshold should be $2/3$ for the top level”).

At this time, the exact guarantees of the Stellar consensus protocol are not properly understood in the research literature. Determining the similarities and differences between Stellar’s “quorum slices” and the generic Byzantine Quorum Systems of Malkhi and Reiter [MR98] for Byzantine consensus is left open.

Antshares/NEO. The project formerly known as *Antshares* has rebranded itself to *NEO* in 2017 [NEO18]. It claims to use a *delegated Byzantine Fault Tolerance (dBFT)* protocol for consensus. No independent or peer-reviewed description of the protocol is currently available.

4.3 Proof of Stake Ledgers

4.3.1 Proof-of-Stake Protocols

The design of *proof-of-stake* blockchain protocols was identified early on as an important objective in blockchain design; a proof-of-stake blockchain substitutes the costly proof-of-work component in Nakamoto’s blockchain protocol [Nak08b] while still providing similar guarantees in terms of transaction processing in the presence of a dishonest minority of users, where this “minority” is to be understood here in the context of stake rather than computational power.

Proof-of-stake protocols typically possess the following basic characteristics. Based on her local view, a party is capable of deciding, in a publicly verifiable way, whether she is permitted to produce the next block. Assuming the block is valid, other parties update their local views by adopting the block, and proceed in this way continuously. At any moment, the probability of being permitted to issue a block is proportional to the relative stake a player has in the system, as reported by the blockchain itself.

A particularly challenging design aspect is that the above probabilistic mechanism should be designed so that the adversary cannot bias it to its advantage. As the stake shifts, together with the evolving population of stakeholders, so does the honest majority assumption, and hence the function that appoints stakeholders should continuously take the ledger status into account.

The idea of proof-of-stake protocols has been discussed extensively in the bitcoin forum [For]. The manner that a stakeholder determines eligibility to issue a block is always publicly verifiable and the proof of eligibility is either computed publicly (via a calculation that is verifiable by repeating it) or by using a cryptographic mechanism that involves a secret-key computation and a public-key verification.

4.3.2 Proof-of-Stake Blockchain Systems.

The first example of the former approach (where the proof of eligibility is publicly computed) appeared in PP-Coin [KN12], and was followed by others including Ouroboros and Snow White [BGM16, KRDO17, BPS16]; while the first example of the latter approach (that involves a secret-key computation and a public-key verification) appeared in NXT [Com14] and was then also used elsewhere, most notably in Algorand [Mic16]. The virtue of the latter approach is exactly in its potential to control adaptive corruptions: due to the fact that the adversary cannot predict the eligibility of a stakeholder to issue a block prior to corrupting it, she cannot gain an advantage by directing its corruption quota to specific stakeholders. Nevertheless, none of these previous works isolated explicitly the properties of the primitives that are required to provide a full proof of security in the setting of adaptive corruptions. Injecting high quality randomness in the PoS blockchain was proposed by Bentov et al. [BLMR14, BGM16], though their proposal does not have a full formal analysis.

Ouroboros. The Ouroboros proof-of-stake protocol [KRDO17] is provably secure in a corruption model that excludes fully adaptive attacks by imposing a corruption delay on the corruption requests of the adversary.

Snow White. The Snow White proof-of-stake [BPS16] is the first to prove security in the Δ -semi-synchronous model but—as in the case of Ouroboros—adopts a weak adaptive corruption model.

Algorand. This protocol ([GHM⁺17]) provides a proof-of-stake ledger that is adaptively secure. Algorand runs a Byzantine agreement protocol for every block and achieves adaptive-corruption security via a novel, appealing concept of player-replaceability. However, Algorand is only secure against a $1/3$ adversary bound; and while the protocol itself is very efficient, it yields an inherently slower block production rate compared to an “eventual consensus” protocol (like Bitcoin, Snow White, and Ouroboros). In principle, proof-of-stake blockchain protocols can advance at the theoretical maximum speed (of one block per communication round), while protocols relying on Byzantine agreement, like Algorand, would require a larger number of rounds to settle each block.

Sleepy consensus. The sleepy model of consensus [PS17] puts forth a technique for handling adaptive corruptions in a model that also encompasses fail-stop and recover corruptions; however, the protocol can be applied directly only in a static stake (i.e., permissioned) setting.

Ouroboros Praos. In a recent work David et al. propose “Ouroboros Praos” [DGKR18]. In Ouroboros Praos, deciding whether a certain participant of the protocol is eligible to issue a block is decided via a private test that is executed locally using a special verifiable random function (VRF) on the current time-stamp and a nonce that is determined for a period of time known as an “epoch”. A special feature of this VRF primitive is that the VRF must have strong security characteristics even in the setting of malicious key generation: specifically, if provided with an input that has high entropy, the output of the VRF is unpredictable even when an adversary has subverted the key generation procedure. In [DGKR18] such VRF functions is called “VRF with unpredictability under malicious key generation”. In a nutshell, the protocol provided in [DGKR18] represents a “robust transaction ledger”. That is, a ledger that enjoys consistency and liveness (see Sec. 3.3). Moreover this protocol is proved secure in the Δ -semisynchronous setting with full adaptive corruptions.

Ouroboros Genesis. In a follow-up work, Badertscher et al. [BGK⁺18] propose “Ouroboros Genesis”. This work provides a blockchain protocols with a novel chain selection rule. The rule enables new or offline parties to safely (re-)join and bootstrap their blockchain from the genesis block without any trusted advice (such as checkpoints) or assumptions regarding past availability. Indeed such a blockchain protocol can “bootstrap from genesis”. Moreover the authors prove that the proposed protocol is secure in the Global UC model against a fully adaptive adversary controlling less than half of the total stake. The model considered in [BGK⁺18] allows adversarial scheduling of messages in a network with delays and captures the dynamic availability of participants in the worst case. This protocol is effectively independent of both the maximum network delay and the minimum level of availability, both of which are run-time parameters.

Chapter 5

Privacy-Preserving Techniques and Systems

Anonymity in Bitcoin Although Bitcoin users are not required to provide their real-world credentials to perform operations on the ledger, their anonymity is severely limited. Every user has a set of *addresses* picked by herself. This allows for some amount of anonymity – no real names are revealed in transactions as an observer sees only addresses and money flow between them. Unfortunately, thoughtful analysis of transactions allows to (almost fully) deanonymise Bitcoin users, see e.g. [GKRN17, AKR⁺13, RH11, RS13]. To address this issue, alternatives to Bitcoin protocol have been proposed. Here some of them are presented: confidential assets [PBF⁺17], privacy-preserving auditing [NVV17] and two examples of the most popular anonymous cryptocurrencies, i.e. Monero [Noe15] and Zerocash [BCG⁺14].

5.1 Confidential Assets

In the Bitcoin network, each node maintains a public ledger which includes the set of UTXOs and the nodes update the set after each new transaction. Since the set of UTXOs and all their past changes are public, it is possible to monitor and analyze all payments of a target address. However, an address cannot be linked to an identity directly, but by monitoring long history of some addresses, there might be some leakages about owners of addresses which will lead to violate privacy of end-users.

In order to deal with such privacy concerns, there have been various proposals. CoinJoin is one of the proposed solutions which aims to hide structure of transactions [Max15]. Intuitively, CoinJoin allows users to combine the transactions interactively and eliminate links from outputs of transactions to inputs and vice versa. But still in transactions, the values of the transferred coins are revealed which might leak information about users in some cases (unless output of all transactions are equal and indistinguishable from each other).

Recently, Poelstra et al. [PBF⁺17] extended an approach to provide more anonymous transactions by blinding values of UTXOs, but still with possibility to check that the sum of input coins equals to sum of spent coins (outputs plus transaction fee). The proposed approach is called the *confidential transactions* which blinds the value of coins in UTXOs with a homomorphic perfectly hiding commitment. Recall that if a commitment scheme is additively homomorphic, it allows to multiply two commitments (e.g. $C_1 = \text{Com}(m_1; r_1)$ and $C_2 = \text{Com}(m_2; r_2)$) and obtain a new commitment to the sum of the two already committed values (e.g. $C_{new} = C_1 \cdot C_2 = \text{Com}(m_1 + m_2; r_1 + r_2)$). The perfectly hiding property guarantees that no receiver (adversary) learns any information about the committed value from the commitment. For confidential transactions, Pedersen homomorphic commitments are used.

With confidential transactions, each transaction contains a homomorphic commitment of the transferred value which blinds the amount. Since the values of transactions are integers and the commitment scheme is homomorphic over a finite ring there might be an issue of overflow. Poelstra et al. deal with this with a *range proof* to show that the committed amount is in a particular range (e.g. $[A, B]$). Their range proof for commitments over slot $[0, m^n - 1]$ is a slightly modified version of Schoenmakers [Sch05] range proof scheme that is based on bit-decomposition approach which expresses numbers in base m and proves that each digit lies in $[0, m - 1]$.

The concept of confidential transactions is also extended to confidential assets, which supports several type of asset (e.g. Bitcoin, USD) just in one transaction while providing confidentially by blinding amount and type of assets (tag value). In brief, the extension assigns a blind asset tag to each output and Pedersen commitment used in transactions commits to both amount of coins and assets tags.

Briefly, there are two main differences between confidential and standard Bitcoin transactions that can be summarized as follows. First, all plain values in the standard Bitcoin transactions are blinded with Pedersen commitment scheme in confidential transactions. Second, in the confidential transactions, the transaction fee is determined explicitly and using homomorphic property of the commitment scheme it should be checked that the differences of inputs minus outputs commits to the value of transaction fee.

5.2 Monero

Monero [Noe15] is a proof-of-work cryptocurrency that focuses on providing transaction privacy. In particular, it claims to hide the sender, the receiver, and the amount of currency in a transaction. However, some of those properties have been found to be partly vulnerable to network analysis attacks [KFTS17,MSH⁺18]. The original Monero protocol was based on the privacy-preserving CryptoNote [vS13] protocol which used one-time keys and a cryptographic tool called ring signature [RST01] to hide senders and receivers of transactions, but the amount was still public. In the beginning of 2017 Monero adopted the Ring Confidential Transaction (Ring CT) protocol [Noe15] which combines the CryptoNote protocol with ideas from confidential transactions [Max15] used in Bitcoin. In particular, this also allows to hide the amounts in transactions.

To remind, transactions in Bitcoin contain inputs and outputs. Inputs refer to outputs of a previous transactions and outputs specify value and to whom to transfer that value by a public key. In a correct transaction, the sum of the input values is (at least) as big as the sum of output values. Moreover, each transaction needs to be signed by the owner of the input transactions. Clearly, Bitcoin does not guarantee any of three above mentioned privacy properties.

5.2.1 Preliminaries

We introduce the main cryptographic tools in Ring CT protocol.

Multilayered Linkable Spontaneous Anonymous Group (MLSAG) signature A ring signature allows for an individual from a group to provide a signature such that it is impossible to identify which member of that group made the signature. Ring CT protocol uses a ring signature with some additional properties. Their signature scheme is called Multilayered Linkable Spontaneous Anonymous Group (MLSAG) signature, but it should be noted that this is still a ring signature, not a group signature. Beyond the usual ring signature properties, MLSAG has some specialized properties to make it usable for cryptocurrencies. Perhaps most importantly, MLSAG is linkable which means that it is possible to detect if two signatures have been created with the same secret key.

5.2.2 Description of Ring CT

We briefly describe how a transaction looks like in the Ring CT protocol.

Receiver of the transaction always generates a new public key for MLSAG, that is, each key is only used one time. Inputs and outputs in a transaction are not given as a plaintext, but as Pedersen commitments. Using the homomorphic property, outsiders can still verify correctness of the transaction. For example if the input of the transaction is $\text{Com}(a_1; r_1)$ and the outputs are $\text{Com}(a_2; r_2)$, $\text{Com}(a_3; r_3)$, then anyone is able to verify that $\text{Com}(a_2; r_2) \cdot \text{Com}(a_3; r_3) = \text{Com}(a_1; r_1)$ if $a_1 = a_2 + a_3$ and $r_1 = r_2 + r_3$. Sender opens the commitment only to the receiver of the transaction, so that he could verify that the received amount is correct. Since arithmetic is done modulo p , receiver also provides range proofs showing that each committed value is in range $\{0, 1, \dots, k\}$ where $k \ll p$. Finally, the sender picks a random subset of unspent public keys of different users together with his own public key and makes a MLSAG signature of the transaction with those keys.

The Ring CT protocol described here is simplified to some extent (for example homomorphic verification is actually slightly more complicated), but it should give a general intuition of the techniques that are used.

5.2.3 Privacy of Ring CT

Next, we informally argue why Ring CT protocol should be secure.

Hiding the Receiver Since public keys of a single user are independent of each other, it is not possible (at least not as easily as in Bitcoin) to link different transactions to the same user.

Hiding the Sender Properties of ring signature guarantee that an attacker should not be able to tell which of the public keys was used to create the signature and hence the sender of the transaction stays private.

Hiding the Amount Amounts in Ring CT are private since they are hidden using the Pedersen commitment scheme.

Double Spending A malicious sender cannot double spend since this would require signing two messages with the same secret key which can be detected due to linkability of MLSAG.

Range proofs avoid overflow attacks. For example, if input to a transaction is $a_1 = 1$ and outputs are $a_2 = 2$ and $a_3 = -1$, then it still holds that $a_2 + a_3 = a_1$. Of course, negative amount of currency does not make sense and since arithmetic in Pedersen commitment is done modulo some large prime p , then -1 gets interpreted as $p - 1$ which is a very large value. Hence such an attack would allow to illegally produce large amounts of new currency. Range proofs show that each input and output is bounded by some small value k and hence this attack is mitigated.

5.2.4 Vulnerabilities

The traceability of Monero transactions has been studied by Kumar et al. [KFTS17] and Möser et al. [MSH⁺18]. Both works found that transactions were quite easily traceable before the beginning of 2017 when Monero introduced the Ring CT protocol. For example Kumar et al. claim that they could identify the real sender in 87% of cases.

Still, even the Ring CT protocol has not made it impossible to launch various network analysis attacks. Monero is especially vulnerable to time analysis attacks: typically in the MLSAG signature, the correct input is the one that was most recently created. According to Möser et al., even in the current protocol this attack uniquely identifies the sender in 45% of the cases.

5.3 Zerocash

Zerocash was the first scalable decentralized anonymous payment scheme. It is based on Zerocoin, however, contrary to the latter, Zerocash provides strong privacy properties and allows for payments of any values. More precisely, Zerocoin does not hide the value of a transaction and other metadata, it uses coins of fixed denomination and does not allow users to pay each other in “zerocoins” (another DLT, like Bitcoin network, is required to make system work). All these issues were fixed in Zerocash. Due to the use of efficient zk-SNARKs, Zerocash was also the first truly scalable anonymous payment system. The theoretical result was later deployed as a full-fledged cryptocurrency called Zcash.

Overview of the system. Payments in Zerocash are done by transferring coins from one user to another. Essentially, each Zerocash coin comes with a serial number sn , commitment cm and public address $addr_{pk_u}$ of its owner u . Commitment cm is public (and stored in an efficient data structure), while the serial number and public address remain private.

User u , who wants to transfer a coin c to user u' (note, for various reasons it may hold that $u = u'$), performs a zero-knowledge proof that she knows (1) c 's serial number such that there exists a corresponding commitment, (2) a secret value $addr_{sk_u}$ corresponding to u 's public address, known only for the coin owner. Furthermore, the proof assures that the coin was created correctly and has a correct value. Transaction results in coin c being destroyed (its serial number is revealed and nobody accepts a coin with a public serial number) and a new coin c' created. The new coin contains (but does not reveal) the public address of the new owner u' . It also has a new serial number determined by $addr_{sk_u}$, what secures c' from being spend by e.g. u .

Zcash and Monero The closest in privacy-preserving properties to Zcash is Monero [Noe15]. Instead of using zk-SNARKs, the main privacy-preserving tool in Monero is a specialized ring signature. The major differences between Zcash and Monero are:

- Coins are gradually introduced to the system by mining (like in Bitcoin) whereas Zcash requires all the anonymous coins to be minted at the protocol setup. As a consequence Zcash's approach seems more centralized as all the coins are originally controlled by a small number of individuals.
- In Zcash privacy of transaction is optional. In practice, only 4% of all Zcash transactions are private. On the other hand, in all transactions Monero hides the sender, recipient and amount of transferred money.
- Currently, efficient zk-SNARKs require trusted setup.
- Zcash is not known to be vulnerable to timing attacks while Monero is. On the other hand, usage pattern related attacks are known for Zcash [KYMM18].

5.3.1 Preliminaries

Below we briefly introduce some basic cryptographic primitives that are necessary for the security and privacy of Zerocash.

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) Zerocash performance heavily relies on NIZK arguments used therein. Thus, their efficiency is crucial. To that end, Zerocash authors proposed to use zk-SNARKs, which assure that even for very long statements proofs are short – for a statement x and the corresponding witness w the argument length is $\text{poly}(|x| + |w|)^{o(1)}$ (sublinear in the length of the statement), [GW11]. More precisely, the authors describe the statement to prove as an arithmetic circuit (which may be a non-trivial task) and use zk-SNARK (from [BCTV14]) to show that the circuit is satisfied.

Table 5.1 (based on Table 2 in [Gro16]) contains a short comparison of the recent zk-SNARKs for a circuit satisfiability problem. Note that all of them are pairing based and utilizes the CRS model. Furthermore, the arguments have constant size, not only sublinear. The properties taken into comparison are: CRS size, proof size, prover P and verifier V computations and number of pairing equations that has to be performed by the verifier to check the given proof.

Key-private encryption scheme A key-private encryption scheme is a public-key encryption scheme where given a ciphertext c it is impossible (for a PPT adversary) to tell which public key was used to produce c . Key-private encryption scheme is crucial for the privacy of Zerocash – given encrypted coin (see below) it should not be possible to determine its receiver.

	CRS size	Proof size	P comp.	V comp.	PPE
[PHGR16]	$7m + n - 2\ell \mathbf{G}$	$8 \mathbf{G}$	$7m + n - 2\ell \mathbf{E}$	$\ell \mathbf{E}, 11 \mathbf{P}$	5
[Gro16]	$m + 2n \mathbf{G}$	$3 \mathbf{G}$	$m + 3n - \ell \mathbf{E}$	$\ell \mathbf{E}, 3 \mathbf{P}$	1
[BCTV14]	$6m + n + \ell \mathbf{G}_1, m \mathbf{G}_2$	$7 \mathbf{G}_1, 1 \mathbf{G}_2$	$6m + n - \ell \mathbf{E}_1, m \mathbf{E}_2$	$\ell \mathbf{E}_1, 12 \mathbf{P}$	5
[Gro16]	$m + 2n \mathbf{G}_1, n \mathbf{G}_2$	$2 \mathbf{G}_1, 1 \mathbf{G}_1$	$m + 3n - \ell \mathbf{E}_1, n \mathbf{E}_2$	$\ell \mathbf{E}_1, 3 \mathbf{P}$	1

Table 5.1: Comparison of zk-SNARKs for arithmetic circuits satisfiability. The statement has length ℓ , the circuit consists of m wires and n multiplication gates. \mathbf{G} denotes group elements, \mathbf{E} exponentiation of elements in the group and \mathbf{P} pairings. First two entries compare symmetric pairings, while the asymmetric pairings are compared in the last two rows.

One-time signature scheme A one-time signature scheme is a signature scheme for which a secret key, public key pair $(\text{sk}_{\text{Sig}}, \text{pk}_{\text{Sig}})$ is valid for a single (yet arbitrarily picked) message only.

5.3.2 Payment Mechanism Ingredients

The Zerocash payment system consists of the following ingredients.

Users Let u be a Zerocash user with:

- *Payment address*: $(a_{\text{pk}}, a_{\text{sk}})$, which allows to receive funds (a_{pk}) and spend them (a_{sk}) ; each user may have multiple addresses. Public key a_{pk} is derived from the secret one by applying a pseudorandom function PRF^{addr} , i.e. $a_{\text{pk}} = \text{PRF}_{a_{\text{sk}}}^{\text{addr}}(0)$.
- *Key-private encryption scheme keys*: $(\text{pk}_{\text{Enc}}, \text{sk}_{\text{Enc}})$, which allow to privately pass a coin to another user. More precisely, to pass a coin \mathbf{c} to receiver u' , user u encrypts \mathbf{c} under u' 's public key pk'_{Enc} . Receiver u' uses her secret key sk'_{Enc} to decrypt \mathbf{c} .
We write addr_{pk} to denote $(a_{\text{pk}}, \text{pk}_{\text{Enc}})$ and addr_{sk} to denote $(a_{\text{sk}}, \text{sk}_{\text{Enc}})$.
- *One-time signature scheme keys*: $(\text{pk}_{\text{Sig}}, \text{sk}_{\text{Sig}})$, that are generated for every transaction separately and prevent malleability of transactions.

Coins A coin \mathbf{c} is a tuple of elements: $\mathbf{c} = (\text{addr}_{\text{pk}} = (a_{\text{pk}}, \text{pk}_{\text{Enc}}), v, \rho, r, s, \text{cm})$. Entries a_{pk} and pk_{Enc} are a public address and a key picked by user u possessing \mathbf{c} ; value v denotes the value of the coin; cm is a coin commitment, which uses randomness s ; ρ and r are random numbers that contribute to the value cm commits to. The coins are secret, the only part publicly available is the coin commitment cm , and address addr_{pk} .

Serial number A serial number sn identifies the coin. Intuitively, transactions consists of sn and a (zero-knowledge) proof that the spender knows a commitment cm corresponding to sn . Serial number is computed by user u as follows: u picks randomness ρ and assigns $\text{sn} \leftarrow \text{PRF}_{a_{\text{sk}}}^{\text{sn}}(\rho)$.

Coin commitments Coin commitments are one of few public parts of a coin. For efficiency reasons, all commitments are stored in a Merkle tree. In Zerocash the tree is full and consists of 2^{64} leaves, what means that at most 2^{64} coins can be *ever* created in the system.

For sake of privacy and security, coin commitments are computed as follows: (1) User u picks randomness r and computes commitment $k = \text{Com}(a_{\text{pk}}, \rho; r)$, for ρ used to compute serial number sn ; (2) The commitment computed above is nested into next-level commitment cm ; more precisely, $\text{cm} \leftarrow \text{Com}(v, k; s)$.

5.3.3 Transferring Coins – Pour Transaction

The main building block to move the coins around is a Pour transaction that is used to transfer funds from one address to another, to split coins and to withdraw coins from the system. Given coins $\mathbf{c}_1^{old}, \mathbf{c}_2^{old}$, Pour transaction allows to create two new coins $\mathbf{c}_1^{new}, \mathbf{c}_2^{new}$ such that the value $\mathbf{c}_1^{old}.v + \mathbf{c}_2^{old}.v$ is equal to $\mathbf{c}_1^{new}.v + \mathbf{c}_2^{new}.v + v_{pub}$, where v_{pub} is a public value redeemed from the old coins $\mathbf{c}_1^{old}, \mathbf{c}_2^{old}$ (i.e. this value can be used to withdraw some amount of money from the system, e.g. to cover transaction fees). Say that the receivers of coins \mathbf{c}_1^{new} and \mathbf{c}_2^{new} are the addresses $\mathbf{c}_1^{new}.addr_{pk}$ and $\mathbf{c}_2^{new}.addr_{pk}$ respectively.

The transaction is defined as a tuple

$$\text{Pour} = (\text{rt}, \mathbf{c}_1^{old}.sn, \mathbf{c}_2^{old}.sn, \mathbf{c}_1^{new}.cm, \mathbf{c}_2^{new}.cm, \pi_{\text{Pour}}, \mathbf{c}_1^{new}.c, \mathbf{c}_2^{new}.c, h_{\text{Sig}}, h_1, h_2, \sigma, pk_{\text{Sig}}),$$

such that:

1. Value $\mathbf{c}_1^{old}.sn, \mathbf{c}_2^{old}.sn$ are serial numbers of the redeemed coins,
2. Entries $\mathbf{c}_1^{new}.cm, \mathbf{c}_2^{new}.cm$ are coin commitments to the newly created coins.
3. Ciphertext $\mathbf{c}_1^{new}.c$ encrypts under $\mathbf{c}_1^{new}.pk_{\text{Enc}}$ values $(\mathbf{c}_1^{new}.v, \mathbf{c}_1^{new}.\rho, \mathbf{c}_1^{new}.r, \mathbf{c}_1^{new}.s)$; similarly for $\mathbf{c}_2^{new}.c$.
4. Values h_{Sig}, h_1 and h_2 tie $\mathbf{c}_1^{new}.a_{sk}$ and $\mathbf{c}_2^{new}.a_{sk}$ in the following way: for given collision-resistant hash function H , $h_{\text{Sig}} = H(pk_{\text{Sig}})$, $h_1 = \text{PRF}_{\mathbf{c}_1^{new}.a_{sk}}^{pk_{\text{Sig}}}(h_{\text{Sig}})$ and $h_2 = \text{PRF}_{\mathbf{c}_2^{new}.a_{sk}}^{pk_{\text{Sig}}}(h_{\text{Sig}})$.
5. The next element, σ is a signature done under sk_{Sig} on every previous element of the Pour transaction.
6. For sake of correctness, Merkle tree root rt is also included in the transaction.

Proving that transaction is correct The most important (and computational-power consuming) part of the system is a NIZK proof that a Pour transaction is correct, i.e. user performing it has sufficient funds, there is no double spending, no new coins are created etc. To assure this, the transaction contains a zk-SNARK proof for the following statement:

- Coins are well-formed, i.e. (1) $\mathbf{c}_1^{old}.k = \text{Com}(\mathbf{c}_1^{old}.a_{pk}, \rho; \mathbf{c}.r)$, $\mathbf{c}_1^{old}.cm = \text{Com}(\mathbf{c}_1^{old}.v, \mathbf{c}_1^{old}.k; \mathbf{c}_1^{old}.s)$; similarly for \mathbf{c}_2^{old} . (2) $\mathbf{c}_1^{new}.k = \text{Com}(a_{pk}, \rho; \mathbf{c}_1^{new}.r)$, $\mathbf{c}_1^{new}.cm = \text{Com}(\mathbf{c}_1^{new}.v, \mathbf{c}_1^{new}.k; \mathbf{c}_1^{new}.s)$; (3) $\mathbf{c}_2^{new}.k = \text{Com}(a_{pk}, \rho; \mathbf{c}_2^{new}.r)$, $\mathbf{c}_2^{new}.cm = \text{Com}(\mathbf{c}_2^{new}.v, \mathbf{c}_2^{new}.k; \mathbf{c}_2^{new}.s)$.
- The address secret key matches the address public key, i.e. $\mathbf{c}_1^{old}.a_{pk} = \text{PRF}_{\mathbf{c}_1^{old}.a_{sk}}^{addr}(0)$; similarly for \mathbf{c}_2^{old} .
- The serial number is computed correctly, i.e. $\mathbf{c}_1^{old}.sn = \text{PRF}_{\mathbf{c}_1^{old}.a_{sk}}^{sn}(0)$; similarly for \mathbf{c}_2^{old} .
- Coin commitment $\mathbf{c}_1^{old}.cm$ is a leaf in a Merkle tree of root rt ; similarly for \mathbf{c}_2^{old} .
- The values add up, i.e. $\mathbf{c}_1^{old}.v + \mathbf{c}_2^{old}.v = \mathbf{c}_1^{new}.v + \mathbf{c}_2^{new}.v + v_{pub}$.
- Values h_1 and h_2 have been correctly computed.

Recall, the statement above is written in Zerocash as an arithmetic circuit, such that the statement holds iff the circuit is satisfiable. The transaction as stated above is publicly verifiable assuming π_{Pour} is, which is the case.

5.3.4 Privacy of Zerocash

Similarly to Bitcoin, coins in Zerocash are transferred from one address to another. However, in the latter, both sender and receiver addresses are hidden. What is only revealed are coin commitments of the newly created coins, which are public anyway.

Hiding the sender Pour transaction does not reveal its sender since:

- Coins serial numbers sn are values of pseudorandom functions evaluated at random point with key a_{sk} . From the security of PRFs, this reveals nothing about a_{sk} .
- Coin commitments cm are perfectly hiding. From the security of commitment schemes, cm reveals nothing about the underlying values.
- Proof π_{Pour} is zero-knowledge, thus it hides its witness.
- h_{Sig} is a value of a hash function evaluated on a one-time key pk_{Sig} ; values h_1 and h_2 come from PRFs, as mentioned above, they hide function's key.

D3.1 – State of the Art of Cryptographic Ledgers

- signature σ is done under one-time key pk_{Sig}

Other Pour entries are independent from addresses of the sender.

Hiding the receiver Pour transaction does not reveal its receiver since:

- Ciphertexts c are key-private, thus they do not reveal the public key used in encryption.

Other Pour entries are independent from addresses of the receiver.

Hiding the amount Pour transaction does not reveal moved value since:

- Commitments cm hides committed values.
- Transaction π_{Pour} is zero-knowledge and hides its witness.
- Ciphertexts c hides encrypted values.

Other Pour entries are independent from the value of the transaction.

5.4 Privacy-preserving Auditing for Distributed Ledgers

In [NVV17], Narula et al. presented zkLedger as the first distributed ledger system to provide strong transaction privacy, public verifiability, and practical, useful auditing. zkLedger guarantees strong transaction privacy: an adversary cannot recognize the participants in a transaction or how much money is being transacted. Also an important property of zkLedger is that it does not reveal the transaction graph, or linkages between transactions. In this system the type of asset being transferred and the time of transactions are public and all participants in zkLedger can still verify transactions and an auditor can issue a set of auditing queries to the participants and receive answers that are provably consistent with the ledger. Then, by using a modified version of the *btcec* library [NVV17] and SHA-256 as the cryptographic hash function, they implement a prototype of zkLedger.

One of the important property of zkLedger is preserving privacy such that still it allows to an auditor to verify the data in the ledger. To this aim, zkLedger uses Pedersen commitments [Ped91] to hide the values in ledger and non-interactive zero-knowledge proofs (NIZKs) [BFM88] to make privacy-preserving assertions about payment details. Pedersen commitment is perfectly hiding and can be homomorphically combined, so in zkLedger for instance, a verifier can confirm that the sum of the outputs is less than or equal to the sum of the inputs, conserving assets. Also an auditor can combine commitments to compute linear combinations of values in different rows in the ledger.

The crucial property for such a system is the completeness which means that a participant cannot omit transactions or lie to the auditor. More precisely, since in the zkLedger an auditor cannot determine the participants of transactions, so zkLedger must guaranty that during auditing, a participant cannot leave out transactions to hide assets from the auditor. zkLedger uses a table-construction in the ledger such that a transaction is a row which includes an entry for every participant, and an empty entry is indistinguishable from an entry involving a transfer of assets.

In terms of efficiency, zkLedger implements a number of optimizations as follows:

1. In order to have fast generating assets proofs, every participant and the auditor keep commitment caches, which are rolling products of every participants' column in the ledger. This also makes it fast to answer audits.
2. For the sake of lower communication costs in zkLedger, participants do not have to interact to construct the proofs for the transaction and similar to other blockchain systems the spender can create the transaction alone.

The zkLedger system consists of three main entities, the shared ledger, participants (banks) and auditor. The shared ledger is a table where the participants (banks) send their transactions. In order to check the validity of the submitted transactions, an auditor Auditor queries participants about their contents on the ledger and verifies their transaction. Briefly, zkLedger uses ZK proofs, additively homomorphic commitment schemes and permissioned ledgers to hides participants, the amounts, and links between transactions while it is still a verifiable transaction on the ledger and the Auditor can receive reliable answers from the participants based on its queries.

D3.1 – State of the Art of Cryptographic Ledgers

In the following there is comparison between zkLedger and some other well known systems related to work in auditing or computing on private data and privacy-preserving blockchains,

zkLedger vs Provisions [DBB⁺15] Similar to the zero-knowledge proofs used in zkLedger, Provisions uses Proof of Assets and Proof of Liabilities, and permissionless ledgers for Bitcoin exchanges to prove they are solvent without revealing their total holdings. The Provisions protocol relies on range proofs to prevent an exchange from inserting fake accounts with negative balances. Size of proofs are very large (about tens of GB) and are linear in the number of customers. However, in Provisions, an exchange could borrow private keys from another Bitcoin holder and thus prove assets they do not actually hold; in fact multiple exchanges could share the same assets. Moreover, Provisions does not provide completeness. zkLedger uses a columnar construction with a distributed ledger and achieves completeness.

zkLedger vs Bitcoin In Bitcoin the transaction amounts and the links between transactions are public, although using Confidential Transactions [Max15] and Confidential Assets [PBF⁺17], one can hide the amounts in transactions, but still they reveal the transaction graph and do not provide completeness. zkLedger provides stronger transaction privacy and private auditing, but at the cost of scalability. Transactions in zkLedger are sized order the number of participants in the whole system, requiring more time to produce and verify as the number of participants grows. This makes zkLedger more suitable to ledgers with fewer participants who require more privacy.

zkLedger vs Solidus [PBF⁺17] In Solidus, the transaction graph and transaction amount between bank customers are blinded by using Oblivious RAM but still it can only support auditing by revealing all of the keys used in the system to an auditor, and opening transactions. Note that however Solidus initially targets permissioned ledger model which the ledger maintainers should be a *permissioned* (fixed-entity) group, but in general they also can be an *unpermissioned* (fully decentralized) ledger, or any other trustworthy append-only data structure. zkLedger provides private auditing with the same performance of Solidus.

Chapter 6

The Relation Between Distributed Ledger Technology and Secure Multiparty Computation

Secure multiparty computation (MPC) enables multiple mutually distrusting parties to jointly compute some function of their private inputs without revealing any more about their private data than what is revealed through the result of the function. This chapter summarises the major results of how distributed ledger technology (DLT) can enhance MPC and how MPC techniques can be exploited to provide additional security properties to distributed ledger protocols.

6.1 Fairness

A fundamental issue in MPC is the fairness problem, i.e., the problem of ensuring that if some party receives the results of a computation, then all parties receive that same result. A classical result by Cleve shows that fairness is impossible to achieve if at least half of the parties act maliciously, i.e., they do not follow the protocol [Cle86]. In particular, Cleve's result implies that fair two-party coin tossing is impossible. The problem is that, if at some point during the protocol, one party determines that the outcome of the protocol will be unfavourable (to that party), it can abort the protocol completely.

Although perfect fairness is impossible without honest majority, there are various works showing the feasibility of weaker notions of fairness. This section provides an overview of how DLT can be used to achieve certain notions of fairness for MPC.

Despite the impossibility of fair two-party coin tossing, Back and Bentov [BB14] show that using the blockchain makes fair two-party (Bitcoin) lottery possible. The protocol uses smart contracts to ensure that if one of the parties aborts the coin tossing protocol, the other party wins by default. The key reason why this makes the protocol fair is that the process of awarding the winner some amount of Bitcoin is inseparable of the process of tossing the coin. So even though the outcome of the coin toss portion of the protocol may actually be biased, the lottery protocol as a whole is fair.

Many solutions that use the blockchain as part of their solution to the fairness problem follow the same principle of compensating honest parties in case a corrupted party aborts the protocol. The remainder of this chapter focuses on solutions for secure multiparty computation. Solutions specific to two-party fairness are not included. Additionally, several (early) solutions explicitly refer to, or are implemented on Bitcoin. For these works, Bitcoin is mentioned explicitly, rather than distributed ledger technology in general.

A certain notion of fairness can be achieved using so-called timed commitments, introduced by Boneh and Naor [BN00]. A timed commitment scheme is a commitment scheme which has the additional property that the other parties can force the commitment to be revealed if the committer fails to do so.

Andrychowicz *et al.* [ADMM14] introduce a variant of timed commitments, in which, instead of forcing the commitment to be revealed, instead, the committer is forced to compensate the other parties in the form of a Bitcoin fine. The authors demonstrate the use of their version of the timed commitment protocol and the aspect of parties respecting the outcome of the protocol by implementing a fair multi-party lottery. One downside of this protocol is that the amount of Bitcoin each party needs to set aside for compensation in case they behave dishonestly is quadratic in the total number of parties. The total number of transactions required is also quadratic in the number of parties.

Bentov and Kumaresan [BK14] propose a formal definition of the claim-or-refund functionality. The claim-or-refund functionality is a two-party primitive that allows a sender to conditionally transfer a deposit to a receiver. If the receiver does not uphold the condition, then the sender can reclaim the deposit at some later time. Subsequently, the authors formally define functionalities for secure computation with penalties, and secure lotteries. As with the variant on timed commitments by [ADMM14], secure computation with penalties does not truly achieve fairness, i.e., it is still possible for the adversary to obtain the result of the computation without the honest parties learning the same, but the honest parties will be compensated in the form of an previously agreed upon amount of coins in case this happens. The authors also show how to construct protocols for these functionalities from the claim-or-refund functionality.

In [KB14], Kumaresan and Bentov study how Bitcoin can be used to incentivise correct computation. This work improves on the fairness results of [BK14], by the same authors, by showing a *constant round* fair secure computation protocol.

Kumaresan *et al.* [KMB15] extend the above techniques from secure function evaluation to reactive functionalities, in which computation proceeds in multiple stages and parties can make decisions and input new information at later stages, depending on the output of earlier stages. In particular, this construction allows for stateful computation. As an instantiation of their techniques, the authors present a secure protocol for playing Texas hold 'em poker.

In [KZZ16], Kiayias *et al.* give a new formalisation using two ideal functionalities for a global transaction ledger and a global clock. The advantage of this formalisation is that the parties can be modelled as regular interactive Turing machines, whereas [BK14] required special features to be added. This places the new formalisation within the standard model. The authors also express their framework in the universal composition setting with global setup for the ledger and clock functionalities and provide a composition theorem for MPC protocols with compensation. Finally, the authors introduce a *robust* MPC protocol with compensation. Robustness is a stronger property than fairness. The fair protocols with compensation discussed above ensure that parties that do not receive the outcome of the computation, receive compensation instead, but only if some other party does receive the result. Robustness dictates that, if the protocol successfully concludes an initial deposit phase, then all parties which do not obtain the result receive compensation, regardless of whether any other party has obtained the result.

In [KB16], Kumaresan and Bentov improve the efficiency of claim-or-refund based fair computation through amortisation. The main idea is that parties can use the same deposit for multiple MPC instances.

[KVV16] improves on the script complexity of claim-or-refund transaction, i.e, the amount of work miners put in to verify a contract, from quadratic to linear in the number of parties. It also improves on the protocol of [KMB15] by reducing the number of claim-or-refund transactions from quadratic to linear in the number of parties.

In [CGJ⁺17], Choudhuri *et al.* present a new model for completely fair MPC using a public bulletin board. Bulletin boards can be achieved through DLT. Unlike most results discussed in this chapter, which define fairness as penalising the parties that abort after obtaining the output, this work achieves fairness in the original sense: either every party obtains the output, or no party does. The main idea of this work is that the problem of fair computation can be reduced to the problem of fair decryption. This is possible by using an unfair MPC protocol to compute an encryption of the desired output, rather than the output itself. Fair decryption itself, however, is a complete functionality for fair MPC [GIM⁺10], meaning that it cannot be achieved in the standard model either. The key insight of this work is that fair decryption is possible for a witness encryption scheme using a

bulletin board. A witness encryption scheme is an encryption scheme that allows encryption of a message with a statement. Decryption is only possible using a witness for the statement. Constructing the statement in such a way that the witness is equal to the proof that some special value is posted on the bulletin board ensures that if one party obtains the witness, then every party can do so, due to the public nature of the bulletin board.

[BKM17] follows up on the work of [KB16] and offers several efficiency improvements in a model that allows for stateful smart contracts. In this model, the authors show a constant round (counting rounds of interaction with the cryptocurrency network) setup phase, as opposed to the results of [KB16], which required a number of rounds quadratic in the number of parties. They also reduce the size of the deposits from quadratic to linear in the number of parties.

6.2 Miscellaneous Enhancements to Secure Computation

This section provides an overview of how distributed ledger technology can be exploited to improve various properties relating to secure multiparty computation.

In principle MPC is concerned with securely computing some output, given some input. This does not regard the issues of authenticity of the input, or whether the parties actually honor the output that is computed. In addition to the fairness results discussed above, Andrychowicz *et al.* [ADMM14] also show how to link inputs and outcomes to Bitcoin transactions, solving these issues insofar as they relate to Bitcoin transactions.

Beyond the fairness result of [KB14], the authors also discuss how Bitcoin can be used to incentivise verifiable computation, which is an interesting primitive for the construction of special multiparty computation protocols. A publicly verifiable computation scheme allows a client to outsource computation to a worker. In addition to the computation result, the worker should also produce a proof that allows the client to verify that the claimed result is indeed correct. For such a scheme to be practical, it is important that the client can verify the correctness faster than performing the computation itself. Pinocchio was the first verifiable computation scheme that was claimed to be (nearly) practical [PHGR13]. A verifiable computation scheme is called *publicly* verifiable if, given the necessary keys, any party can verify the correctness of the result. Crucially, public verifiability requires that knowledge of the verification key does not allow the worker to produce an invalid proof.

In [KB14] the authors show how to incentivise computation by integrating verifiable computation with Bitcoin. Despite the relatively efficient verification property of verifiable computation schemes, a naive integration of a verifiable computation scheme with Bitcoin in which the network performs the verification of the result to determine whether payout occurs would still be too expensive. Instead, the authors introduce a variant of the claim-or-refund functionality in which the miners only need to perform the verification algorithm in case the client fails to pay the worker. [KB14] also formalises and proposes candidate realisations for non-interactive bounty mechanisms, in which a client can pose a computational problem and *any* party can present a solution to claim the reward, i.e., any party can act as the worker and the identity of such workers does not have to be known at the time the client poses the problem. Unfortunately, the actual Bitcoin protocol did not support the instructions necessary to implement these protocols at the time of writing.

Electronic voting is a special case of MPC. The problem of private voting using bitcoin was studied by Zhao and Chan [ZC15]. Their protocol makes use of a threshold signature scheme to ensure that all voters agree on the signature needed for the compensation transaction.

6.3 Privacy and Secure Storage

The previous two sections concerned the various ways in which DLT can be used to improve the properties of MPC. This final section summarises results in which MPC techniques are employed to provide data privacy properties to ledgers.

Zyskind *et al.* introduced the secure computation platform Enigma [ZNP15]. Enigma focusses on issues revolving around secure storage, exploiting the ledger to manage access control and serve as a log of events. Data are not stored directly on the ledger, instead, distributed hash tables are used. This way, not all nodes need

to store the data. Storage and computation fees can be arranged using the cryptocurrency underlying the ledger. Storage fees are received by those parties that actually store the data, Computation fees by those parties that carry out computation on behalf of some other party.

The smart contract system Hawk [KMS⁺16] uses MPC techniques to improve privacy properties of distributed ledgers. Hawk offers transactional privacy and private smart contracts. The execution of private smart contracts is facilitated by a minimally trusted manager, a special party that can see user's inputs and is trusted not to disclose them. This manager is not a true trusted party, however, as it cannot affect the correct execution of a contract, nor affect the security of the underlying currency. The authors note that the manager could be replaced by MPC, which would strengthen users' privacy, but have opted not to do so out for efficiency and practicality.

The work of Benhamouda *et al.* [BHH18] investigates the use of MPC techniques for storing private data on the ledger and to evaluating smart contracts based on such private data. The authors implement their system on the permissioned blockchain architecture Hyperledger Fabric. Private data are stored encrypted on the ledger, with the data owner having access to the corresponding keys. This is unlike other systems that use DLT for private data storage, which typically store the data "off-chain" only store the information required to facilitate retrieval of the data. Contracts involving private data can be evaluated using MPC between (at least) the parties whose private data are referred to in the contract.

In [KAS⁺18] Kokoris-Kogias *et al.* introduce the decentralised access control mechanism for private data storage on blockchain systems SCARAB. SCARAB addresses the issues of accountability and revocability. Accountability is provided by atomically logging every request that is handled on the ledger. Revocability is achieved through collectively managed data access policies.

Chapter 7

Open Problems

Illegal content in the blockchain of Bitcoin. A recent work [MHH⁺18] shows that the blockchain of Bitcoin contains illegal content, for instance files of child pornography and others sensible information that can violate the right to be forgotten of a person. In addition, the need of privacy is motivating additional regulations (e.g., GDPR) that require to delete data collected in the past. As a consequence governments could enforce the removal of some content of a blockchain and this can cause to break down of the entire Bitcoin system.

An interesting open question is to find a practical solution that allows to delete data still keeping some sufficient structure of the blockchain that is sufficient for the applications.

Relation stake-based consensus and traditional Byzantine consensus. Many public blockchain networks are currently experimenting with proof-of-stake consensus. Some, like the Cardano Settlement Layer with Ouroboros, use only the pure stake held by its members, in terms of a token or cryptocurrency. Others, like the Casper protocol proposal for Ethereum rely on validators to actively lock up some of their currency holdings before they then can participate in a more traditional BFT voting protocol. This investment is the stake that they might lose if they misbehave during the voting protocol, i.e., through equivocation. Yet others, like Algorand, essentially elect a small committee using a proof-of-work lottery and then execute rounds of BFT voting among the committee.

These competing approaches are currently not well understood. Why is stake used for delegating to BFT-style voting protocols? In which sense do these protocols maintain incentive compatibility and lead to equilibria? How can the differences between these protocols be analyzed formally?

Relaxing time assumptions. One of the most common approaches in the UC treatments of blockchains is to assume a mechanism that allows the parties to be somehow synchronized. The level of synchronization required by the protocols depends on many aspects; in general a more flexible blockchain could require a weaker synchronisation. In the UC treatments of blockchain, such as in [GKL14, BGK⁺18, DGKR18, BMTZ17], the synchronization is modelled as a clock that the parties can have access to. Though, in the real world these “timing” assumptions could be not realistic. An open problem is to better understand what is the minimum level of synchrony required to model a meaningful blockchain.

Recovering from corrupted ledger zkLedger provides publicly verifiable auditing on confidential transactions. It is not clarified that what happens if the distributed ledger would be corrupted by some malicious parties. In short answer all parties need to be gathered to construct old transactions. Addressing this issue precisely could be a prominent improvement in the zkLedger.

Moreover, current version of zkLedger does not provide support for transactions That are committed unwillingly. Presenting a version of zkLedger which address this issue is an interesting research topic in this direction,.

More efficient range proofs for confidential transactions One of the main primitives in confidential transactions, as described in Sec. 5.1, are range proofs. Proposing a more efficient range proof can improve the performance of whole system. In the current version for each commitment there is an individual range proof. Another interesting research direction could be aggregating range proofs, such that one can give a proof for two commitments. Such a proof can be place in Merkle tree to give efficient range proof of the children, but with more efficient verification.

Post-quantum confidential transactions The primitives used in a version of confidential transactions from [PBF⁺17] lie on elliptic-curve discrete-log assumption which is not secure against quantum attacker. Substituting used primitives with post-quantum secure primitives could be a big improvement.

Bibliography

- [ABB⁺18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Rui Oliveira, Pascal Felber, and Y. Charlie Hu, editors, *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 30:1–30:15. ACM, 2018.
- [ABC17] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In Matteo Maffei and Mark Ryan, editors, *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10204 of *Lecture Notes in Computer Science*, pages 164–186. Springer, 2017.
- [AD15] Marcin Andrychowicz and Stefan Dziembowski. Pow-based distributed cryptography with no trusted setup. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 379–399. Springer, 2015.
- [ADH08] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *PODC*, pages 405–414. ACM, 2008.
- [ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 443–458. IEEE Computer Society, 2014.
- [AGK⁺15] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. The next 700 BFT protocols. *ACM Trans. Comput. Syst.*, 32(4):12:1–12:45, 2015.
- [AJK05] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, July 2005.
- [AKM⁺15] Frederik Armknecht, Ghassan O. Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In Mauro Conti, Matthias Schunter, and Ioannis G. Askoxylakis, editors, *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, volume 9229 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 2015.

- [AKR⁺13] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2013.
- [AMQ13] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: redundant byzantine fault tolerance. In *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA*, pages 297–306. IEEE Computer Society, 2013.
- [Apa18] Apache. Apache kafka – a distributed streaming platform. Web site, August 2018. <https://kafka.apache.org/>.
- [BB14] Adam Back and Iddo Bentov. Note on fair coin toss via bitcoin. *CoRR*, abs/1402.3698, 2014.
- [BCG⁺14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.
- [BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796. USENIX Association, 2014.
- [BDDS92] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. *Inf. Comput.*, 97(2):205–233, 1992.
- [Ben83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983.
- [BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 103–112. ACM, 1988.
- [BFT18] BFT-SMaRt project. BFT-SMaRt. GitHub repository, August 2018. <https://github.com/bft-smart/library>.
- [BGJ⁺16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.
- [BGK⁺18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. *IACR Cryptology ePrint Archive*, 2018:378, 2018.
- [BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2016.
- [BGM⁺18] Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2018.

- [BGP92] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. *Bit Optimal Distributed Consensus*, pages 313–321. Springer US, Boston, MA, 1992.
- [BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.
- [BHH18] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. In Abhishek Chandra, Jie Li, Ying Cai, and Tian Guo, editors, *2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018*, pages 357–363. IEEE, 2018.
- [BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.
- [BKM17] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 410–440. Springer, 2017.
- [BLMR14] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract]. *SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.
- [BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.
- [BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.
- [Bor96] Malte Borderding. Levels of authentication in distributed agreement. In Özalp Babaoglu and Keith Marzullo, editors, *Distributed Algorithms, 10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996, Proceedings*, volume 1151 of *Lecture Notes in Computer Science*, pages 40–55. Springer, 1996.
- [BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
- [BPW91] Birgit Baum-Waidner, Birgit Pfitzmann, and Michael Waidner. Unconditional byzantine agreement with good majority. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91, 8th Annual Symposium on Theoretical Aspects of Computer Science, Hamburg, Germany, February 14-16, 1991, Proceedings*, volume 480 of *Lecture Notes in Computer Science*, pages 285–295. Springer, 1991.
- [BSA14] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on*

D3.1 – State of the Art of Cryptographic Ledgers

Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014, pages 355–362. IEEE Computer Society, 2014.

- [But13] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform. <https://github.com/ethereum/wiki/wiki/White-Paper>, 2013.
- [Cac09] Christian Cachin. Yet another visit to Paxos. Research Report RZ 3754, IBM Research, November 2009.
- [Cac16] Christian Cachin. Architecture of the Hyperledger blockchain fabric. Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL 2016), 2016.
- [Can96] Ran Canetti. Studies in secure multiparty computation and applications. *pp73-79, March*, 1996.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
- [CCD87] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, page 462. Springer, 1987.
- [CFF⁺05] Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *J. Cryptology*, 18(3):191–217, 2005.
- [CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996.
- [CGJ⁺17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 719–728. ACM, 2017.
- [CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395. IEEE Computer Society, 1985.
- [Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
- [Cha17] Chain protocol whitepaper. Available online, <https://chain.com/docs/1.2/protocol/papers/whitepaper>, 2017.
- [Cha18] Chain. Chain. Web site, August 2018. <http://chain.com/>.

D3.1 – State of the Art of Cryptographic Ledgers

- [CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptology*, 18(3):219–246, 2005.
- [CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
- [Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369. ACM, 1986.
- [CM18] Brad Chase and Ethan MacBrough. Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242, 2018.
- [Com14] The NXT Community. Nxt whitepaper, 2014. <https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf>.
- [Cor18] Corda. Corda. Web site, August 2018. <https://corda.net/>.
- [CP02] Christian Cachin and Jonathan A. Poritz. Secure intrusion-tolerant replication on the internet. In *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*, pages 167–176. IEEE Computer Society, 2002.
- [CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 249–259. IEEE Computer Society, 2007.
- [CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51. ACM, 1993.
- [CV17] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017.
- [CW89] Brian A. Coan and Jennifer L. Welch. Modular construction of nearly optimal byzantine agreement protocols. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 295–305. ACM, 1989.
- [DBB⁺15] Gaby G Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 720–731. ACM, 2015.
- [DG17] Florian Dold and Christian Grothoff. Byzantine set-union consensus using efficient set reconciliation. *EURASIP J. Information Security*, 2017:14, 2017.
- [DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.
- [DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.
- [DPPU88] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.

- [DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.
- [DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
- [ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014.
- [FC16] Marc Fischlin and Jean-Sébastien Coron, editors. *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*. Springer, 2016.
- [FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 211–220. ACM, 2003.
- [Fit03] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2003.
- [FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.
- [FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.
- [FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
- [FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *STOC*, pages 148–161. ACM, 1988.
- [FM97] Pease Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
- [For] Bitcoin Forum. Refer e.g., to the posts by QuantumMechanic and others from 2011 <https://bitcointalk.org/index.php?topic=27787.0> (Last Accessed 19/09/2017).
- [GHM⁺17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nikolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017.
- [GIM⁺10] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2010.
- [GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 658–668. IEEE Computer Society, 2007.

- [GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 179–186. ACM, 2011.
- [GKL14] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. *IACR Cryptology ePrint Archive*, 2014:765, 2014.
- [GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
- [GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 291–323, 2017.
- [GKLP18] Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 465–495. Springer, 2018.
- [GKP17] Juan A. Garay, Aggelos Kiayias, and Giorgos Panagiotakos. Proofs of work for blockchain protocols. *IACR Cryptology ePrint Archive*, 2017:775, 2017.
- [GKRN17] Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *CoRR*, abs/1708.04748, 2017.
- [GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.
- [GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 174–187. IEEE Computer Society, 1986.
- [Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Fischlin and Coron [FC16], pages 305–326.
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 99–108. ACM, 2011.
- [HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 132–150, 2011.

D3.1 – State of the Art of Cryptographic Ledgers

- [Hyp18a] Hyperledger. Hyperledger fabric. GitHub repository, August 2018. <https://github.com/hyperledger/fabric/>.
- [Hyp18b] Hyperledger. Hyperledger indy. Web site, August 2018. <https://www.hyperledger.org/projects/hyperledger-indy>.
- [HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 466–485. Springer, 2010.
- [Kad16] Kadena. Juno – smart contracts running on a bft hardened raft. GitHub repository, November 2016. <https://github.com/kadena-io/juno>.
- [KAS⁺18] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gailly, Philipp Jovanovic, Linus Gasser, and Bryan Ford. Hidden in plain sight: Storing and managing secrets on a public ledger. *IACR Cryptology ePrint Archive*, 2018:209, 2018.
- [KB14] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 30–41. ACM, 2014.
- [KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Weippl et al. [WKK⁺16], pages 418–429.
- [KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero’s blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, volume 10493 of *Lecture Notes in Computer Science*, pages 153–173. Springer, 2017.
- [KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer, 2006.
- [KMB15] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 195–206. ACM, 2015.
- [KMS⁺16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 839–858. IEEE Computer Society, 2016.
- [KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.
- [KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012. <https://peercoin.net/assets/paper/peercoin-paper.pdf>.

- [KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 357–388, 2017.
- [KVV16] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In Weippl et al. [WKK⁺16], pages 406–417.
- [KYMM18] George Kappos, Haaron Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 463–477. USENIX Association, 2018.
- [KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Fischlin and Coron [FC16], pages 705–734.
- [Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.
- [Lam01] Butler W. Lampson. The abcd’s of paxos. In Ajay D. Kshemkalyani and Nir Shavit, editors, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, page 13. ACM, 2001.
- [Lis10] Barbara Liskov. From viewstamped replication to byzantine fault tolerance. In Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors, *Replication: Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*, pages 121–149. Springer, 2010.
- [LJC⁺18] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *CoRR*, abs/1802.06993, 2018.
- [LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
- [Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
- [Mar16] Will Martino. Kadena — the first scalable, high performance private blockchain. Whitepaper, <http://kadena.io/docs/Kadena-ConsensusWhitePaper-Aug2016.pdf>, 2016.
- [Max15] Greg Maxwell. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt, 2015.
- [Maz16] David Mazières. The Stellar consensus protocol: A federated model for Internet-level consensus. Stellar, available online, <https://www.stellar.org/papers/stellar-consensus-protocol.pdf>, 2016.
- [MHH⁺18] Roman Matzutt, Jens Hiller, Martin Henze, Jan Henrik Ziegeldorf, Dirk Mllmann, Oliver Hohlfeld, and Klaus Wehrle. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin, 2018.
- [Mic16] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.
- [ML14] Andrew Miller and Joseph J. LaViola. Anonymous Byzantine consensus from moderately-hard puzzles: A model for bitcoin. University of Central Florida. Tech Report, CS-TR-14-01, April 2014.
- [MR98] Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

- [MSH⁺18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, 2018.
- [MXC⁺16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Weippl et al. [WKK⁺16], pages 31–42.
- [Nak08a] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2008.
- [Nak08b] Satoshi Nakamoto. “the proof-of-work chain is a solution to the Byzantine Generals’ problem”. The Cryptography Mailing List, <https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html>, November 2008.
- [Nei94] Gil Neiger. Distributed consensus revisited. *Inf. Process. Lett.*, 49(4):195–201, 1994.
- [NEO18] NEO. Neo. Web site, August 2018. <https://neo.org/>.
- [Noe15] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.
- [NVV17] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. *auditing*, 17(34):42, 2017.
- [NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 33–43. ACM, 1989.
- [Oku05a] Michael Okun. Agreement among unacquainted Byzantine generals. In Pierre Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 499–500. Springer, 2005.
- [Oku05b] Michael Okun. Distributed computing among unacquainted processors in the presence of Byzantine failures. Ph.D. Thesis Hebrew University of Jerusalem, 2005.
- [OL88] Brian M. Oki and Barbara Liskov. Viewstamped replication: A general primary copy. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 8–17. ACM, 1988.
- [PBF⁺17] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *Financial Cryptography Bitcoin Workshop*. <https://blockstream.com/bitcoin17-final41.pdf>, 2017.
- [Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.
- [PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013.
- [PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, 2016.
- [PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 380–409, 2017.

- [PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.
- [PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
- [PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.
- [Rab83] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983.
- [RH11] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *SocialCom/PASSAT*, pages 1318–1326. IEEE, 2011.
- [Rip18] Ripple. Ripple. Web site, August 2018. <https://ripple.com/>.
- [RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 6–24. Springer, 2013.
- [RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.
- [SB12] João Sousa and Alysson Neves Bessani. From byzantine consensus to BFT state machine replication: A latency-optimal transformation. In Cristian Constantinescu and Miguel P. Correia, editors, *2012 Ninth European Dependable Computing Conference, Sibiu, Romania, May 8-11, 2012*, pages 37–48. IEEE Computer Society, 2012.
- [SB15] João Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, September 28 - October 1, 2015*, pages 146–155. IEEE Computer Society, 2015.
- [SBV17] João Sousa, Alysson Bessani, and Marko Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. *CoRR*, abs/1709.06921, 2017.
- [Sch90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.
- [Sch05] Berry Schoenmakers. Interval proofs revisited. In *slides presented at International Workshop on Frontiers in Electronic Elections*, 2005.
- [Sie16] David Siegel. Understanding the dao attack. <http://www.coindesk.com/understanding-dao-hack-journalists/>, 2016.
- [Ste18] Stellar. Stellar. Web site, August 2018. <https://www.stellar.org/>.

- [Swa15] Tim Swanson. Consensus-as-a-service: A brief report on the emergence of permissioned, distributed ledger systems. Report, available online, April 2015.
- [SYB17] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple protocol consensus algorithm. Ripple Inc., available online, https://ripple.com/files/ripple_consensus_whitepaper.pdf, 2017.
- [Sym18] Symbiont. Assembly. Web site, August 2018. <https://symbiont.io/technology/assembly>.
- [TC84] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Inf. Process. Lett.*, 18(2):73–76, 1984.
- [Ten18] Tendermint. Tendermint core. GitHub repository, August 2018. <https://github.com/tendermint/tendermint>.
- [Upf92] Eli Upfal. Tolerating linear number of faults in networks of bounded degree. In Norman C. Hutchinson, editor, *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1992*, pages 83–89. ACM, 1992.
- [VCBL09] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin one’s wheels? byzantine fault tolerance with a spinning primary. In *28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009*, pages 135–144. IEEE Computer Society, 2009.
- [vS13] Nicolas van Saberhagen. Cryptonote v 2.0. <https://cryptonote.org/whitepaper.pdf>, 2013.
- [WKK⁺16] Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 2016.
- [Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.
- [ZC15] Zhichao Zhao and T.-H. Hubert Chan. How to vote privately using bitcoin. In Sihan Qing, Eiji Okamoto, Kwangjo Kim, and Dongmei Liu, editors, *Information and Communications Security - 17th International Conference, ICICS 2015, Beijing, China, December 9-11, 2015, Revised Selected Papers*, volume 9543 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2015.
- [ZNP15] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *CoRR*, abs/1506.03471, 2015.