TU WIEN Informatics

Privacy Enhancing Technologies for Distributed Ledgers

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Mathias Wolf, BSc
Matrikelnummer 01315079

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Gernot Salzer
Mitwirkung: Ass.Prof. Monika di Angelo

Wien, 13. Juli 2020

Mathias Wolf	Gernot Salzer

TU WIEN Informatics

Privacy Enhancing Technologies for Distributed Ledgers

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Mathias Wolf, BSc
Registration Number 01315079

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Gernot Salzer
Assistance: Ass.Prof. Monika di Angelo

Vienna, 13th July, 2020

Mathias Wolf	Gernot Salzer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Mathias Wolf, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. Juli 2020

Mathias Wolf

Danksagung

Mein besonderer Dank gilt meinen beiden Betreuern, Gernot Salzer und Monika di Angelo, welche mir stets mit Rat beistanden und mir ihr Vertrauen schenkten, die Themen dieser Arbeit weitgehendst selbst zu definieren. Sie ermöglichten es mir stets meinen Interessen vollends nachzugehen, was diese Arbeit für mich zur wunderbaren Entdeckungsreise werden ließ. Weiters danke ich Georg Fuchsbauer, der mir dabei half, eine Vielzahl an Fehlern zu finden, und mir bei Fragen der kryptografischen Sicherheit meiner Konstruktionen zur Seite stand.

Kurzfassung

Die Möglichkeit, Computerberechnungen mit einem Beweis für deren korrekte Ausführung zu hinterlegen, scheint in Zeiten der oft genannten 'Digitalisierung' ein vielversprechendes Mittel zu sein, um es Parteien, die einander nicht trauen, zu ermöglichen sich auszutauschen. Diese Korrektheitsbeweise sind gerade im Themengebiet der Blockchain interessante und vielversprechende Kandidaten, um dortige Probleme zu adressieren. In dieser Arbeit untersuchen wir formal, was eine Blockchain im Sinne Nakamotos auszeichnet, und inwiefern sogenannte 'Privacy Enhancing Technolgies' in diesen dezentralen Systemen Anwendung finden. Das Hauptaugenmerk legen wir dabei auf 'pairing-based preprocessed zero knowledge succinct non interactive arguments of knowledge' (zkSNARKs). Wir untersuchen deren mathematische Grundlage und entwickeln unsere eigene zkSNARK Konstruktion. Diese basiert auf einer Erweiterung der gängigen $\mathbb{F}$-arithmetischen Schaltung, die neben arithmetischen Operationen auch die der skalare Gruppenexponentiation auf elliptischen Kurven unterstützt. Die Implementierung ist in der Sprache Golang verfasst und unter einer GPL3.0 Lizenz auf Github zugänglich gemacht. Das Programm übersetzt eine eigens dafür entwickelte Programmiersprache in die Form eines 'Quadratic Arithmetic Programs' und kann daher über unsere Arbeit hinaus in Beweissystemen basierend auf $\mathbb{F}$-arithmetischen Schaltungen eingesetzt werden.

Abstract

In our digital society, proofs of computation provide new means of interaction among distrusting parties. This thesis investigates why these proof systems are especially desirable tools in the Blockchain space. We first provide a formal definition of a Nakamoto-style-Blockchain, and an overview of state-of-the-art 'privacy enhancing technologies' currently used in these decentralized environments. The main focus lies on 'pairing based preprocessed zero knowledge succinct non interactive arguments of knowledge' (zkSNARKs). This thesis also investigates their mathematical foundations and provides the description and implementation of a zkSNARK construction that is derived from a new type of underlying circuit. This new circuit extends $\mathbb{F}$-arithmetic circuit-based constructions to enable proving knowledge of the discrete logarithm on an elliptic curve with just one gate. The prototype is implemented as Golang code and is publicly accessible via Github under a GPL3.0 license. It includes a compiler that translates a domain specific language designed for this purpose efficiently into the form of a 'quadratic arithmetic program'. This language and its compiler can be used for other types of proof systems based on $\mathbb{F}$-arithmetic circuits as well.

Contents

CHAPTER 1

Introduction

1.1 Motivation

'The Times 03/Jan/2009 Chancellor on brink of second bailout for banks'
Bitcoin Mainnet's Genesis Block coinbase parameter

How can one start a thesis about cryptocurrency technologies other than by quoting what Satoshi Nakamoto wrote into the first block of Bitcoin's main net. The quote was taken from a Financial Times headline [can]. She did not provide a public explanation for this choice, but the year before in 2008, the global economic crisis increased the gap between the poor and the rich. As the ruling neoclassical theory of economy lacked the ability to explain the disaster[Her16], the growing community around Bitcoin soon got an idea what she might have had in mind when choosing this headline. Many people saw Bitcoin as an attempt to shake the foundations of our current monetary system and to rebuild it from scratch. Spread over the world wide web it is a technology that neither stops at national borders nor allows unsecured bank money creation or the targeted exclusion of anyone. Naive, yet, one can argue, also an attempt to shine light on one of the most important challenges we face in the 21st century, namely rethinking what money is, who controls it and how, and what characteristics it should have.

'As the human race gradually draws together to solve problems that are increasingly global in character with the aid of rapid communication and dissemination of information worldwide the associated complexities contradictions and difficulties loom large on the horizon'
Murray Gell-Man, ACM Talk on 'the quality of information', 1997

1.2 Research Questions and Expected Results

In this thesis we address the following research questions.

- **What is a blockchain?**

  Ironically, the scientific community struggles to find consensus on what makes up consensus machineries. The target is a moving one as blockchains emerge from a collective interest. A satisfying definition is therefore very hard, if not impossible, to find. Different distributed ledgers aim to provide different services. Consequently, the techniques vary on how to keep these consensus systems maintainable and decentralized at the same time [GKL14].

  The goal of this analysis is to find and classify elementary properties of a blockchain and investigate how they relate to each other. Another key aim is to provide an algorithmic description of a blockchain protocol that is based on the expense of a resource and in its core aims for unrestricted accessibility as well as censorship resistance.

  The formal definition of a blockchain will be stated in terms of an interactive message exchange protocol between an honest full node and an adversary that tries to fork the chain. Such interactive protocols provide a common way to specify security requirements in cryptographic protocols.

- **Which state-of-the-art privacy enhancing techniques are used in distributed ledger systems?**
  As blockchains are publicly accessible message transcripts, their use cases are tightly bound to the availability of so called privacy enhancing techniques (PET).

  We will provide an overview of some of the currently implemented PETs for distributed ledgers and will describe their fields of application, advantages and drawbacks.

- **What are zkSNARKs?**

  zkSNARKs provide arguments of knowledge for arbitrary statements within the problem class NP. They are statistical knowledge sound, verifiable in time linear in the input size and are constant in size, which makes them promising candidates to address the scalability issue as well as privacy concerns in the blockchain space.

  Understanding the underlying assumptions is essential before developers can integrate SNARKs into their ecosystem. We will discuss the mile stone papers in this area in order to understand how and why these constructions work.

  The expected result is a complete description of all steps needed to create a zkSNARK, accompanied by a publicly accessible implementation that enables the compilation of code into arithmetic circuits, from there into the rank one constraint system (R1CS) language, and from R1CS into quadratic arithmetic programs, which form the basis of a SNARK and zkSNARK.

- **Can we extend zkSNARKs?**

  When researching zkSNARKs, the question arose whether it is possible to create a SNARK system for a language that combines algebraic with arithmetic statements. Such systems are highly desirable and the subject of ongoing research[AGM18]. They promise huge efficiency gains on the prover side in certain scenarios and could consequently enable low-energy devices to create SNARKs.

  This work's contribution is a pairing-based, preprocessed, however only very limited statistical knowledge sound, SNARK for composed statement satisfiability. In addition to proving arithmetic circuit satisfiability, this SNARK proves knowledge of a discrete logarithm on an elliptic curve with a single algebraic gate, instead of hundreds of arithmetic gates as in current state-of-the-art constructions. The construction is publicly verifiable. A trusted third-party generates a proving key and a verification key. Afterwards anyone can use the proving key to generate non-interactive proofs for adaptively-chosen NP statements, and the proofs can be verified by anyone using the verification key. This SNARK works with asymmetric pairings for higher efficiency and a proof consists of four group elements only. To verify a proof we need just four pairings. In addition, this SNARK can be turned into zero-knowledge, which means that it does not reveal anything about the inputs the prover used to create the proof. On the theoretical side, we prove the completeness of the construction and analyze its soundness. On the practical side, we provide a an implementation publicly available at [Wol].

CHAPTER 2

Public Distributed Ledger

A public distributed ledger **PDL** is a publicly accessible self-organizing raid mirror system with a dynamic number of mirrors. The data which is mirrored therefore is highly persistent if the number of mirrors is high and if they are under independent control. There are several aspects on the basis of which PDLs can be categorized and distinguished. Their key properties and dependencies amongst each other are shown in figure 2.1.

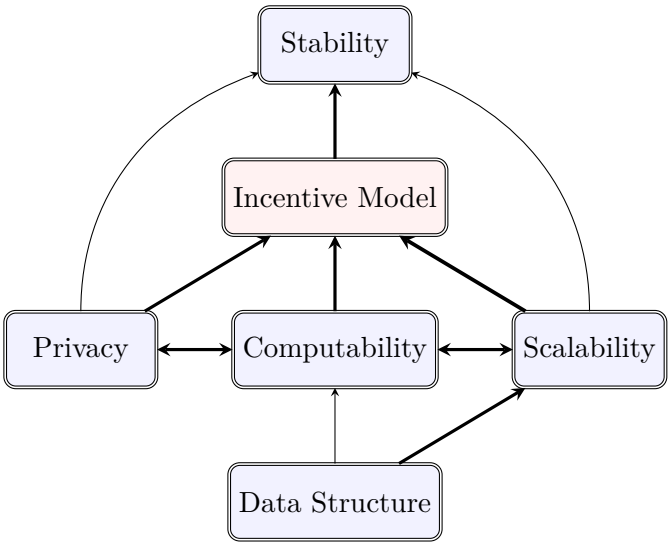


Figure 2.1: Dependencies among PDL properties. The direction of an arrow indicates impact. For example: Privacy affects the Incentive Model, but not vice versa.

- With the term **stability** of a PDL system, we refer to its degree of decentralization and independence regarding the protocol's continuity. A PDL might be decen-

tralized, yet unstable. Blockstream's version of Bitcoin, for example, is currently unstable since in case the four biggest mining-pools stop their participation, the protocol is will likely require a hardfork[1], since the rapid decline in hash rate would cause the average block creation time to become unbearably high until the next difficulty adaptation (max 2016 blocks) takes place. For a big part, the stability is a consequence of the incentive model. Privacy on the network layer increases stability as well since DoS attacks on miners and full nodes are harder to conduct.

- The **incentive model** has the task to ensure that the self organizing system does not die out but preferably grows or stagnates instead. It is the primary source of a PDLs stability, since it is in some sense the sum of all possible answers to the question: "Why should I run a full node?". A wide range of answers to that question is the goal. It is usually achieved by a mixture of top down and bottom up approaches. The top down approach is: the developers of the PDL try to answer the question by design in advance. The bottom up approach is: provide strong privacy, scalability, computability and data structure (as shown in 2.1), in order to increase the chance of having the user find its own answer to the question. The most commonly chosen top down measure is the introduction of a token system. If the tokens are exchangeable for goods, it is likely that a wide range of independent users is attracted. The token creation mechanism then can be used as a strong incentive and enables the developers to apply game-theoretic considerations in order to maximize stability.

- The **data structure** determines if the database updates (messages) have a total order (blockchain) or a partial order (directed acyclic graph 'DAG'). It affects how concurrently appearing ledger updates can be dealt with, and is tightly connected to the ledger's ability to scale. In practice so far, DAG based ledgers could not prove to support higher transaction rates than blockchains with comparable rates of stability, as it was claimed in [SZ13]. The data structure influences the ledgers growth of the required storage space, since it enables different pruning techniques. DAG based PDLs have strong limitation in their ability to support 'smart contracts', since there is no (or only limited) consensus on the ordering of messages in time.

- The property whether interaction with the ledger preserves **privacy** is often referred to as a feature. Yet privacy is arguably more then a feature, since it requires consideration starting from the base layer of the blockchain design down to depths of the network layer to avoid IP tracing [kov]. As an analogy: Tor is not a Firefox addon. Applied privacy enhancing technologies have severe impacts on a PDL's real world use case. It encourages or even forces the user to run a full node instead of a light client, which in turn affects the networks stability.

- The category **computability** covers the complexity and therefore computational effort a single transaction (message) can pose on a validating full node. In every

[1] At this point it is hard to name any cryptocurrency that meets all the requirements of stability in our sense.

PDL, a transaction has minimum requirements to be valid. Additionally it can carry a set of rules (code) whose execution during validation poses additional computational effort on the network. Computability is therefore in close connection with scalability, as the maximum number of messages a network is able to verify depends on the average computational cost of a message. If a PDL has advanced computability, codes can contain cryptographic computations and therefor enable privacy preserving interactions such as (zero) knowledge proofs. Systems which allow succinct verification of complex rule sets are an active field of research and will be treated in chapter 4. To support the execution of code that depends on consensus on the order of transactions, the choice of a proper data structure is decisive. Finally: If computability is high, the expressibility of code attached to a message is high as well. Consequently the users are more likely to strengthen the incentive model from bottom up since they will find use cases the developers probably have not thought of.

- **Scalability** addresses how the stability of a PDL is affected when the average number of computations the network performs per unit of time increases. Computability affects scalability since it enables off-chain interactions on layer 2 as well as sharding techniques where not every node is required to fully conduct each computation. Conversely scalable PDLs support the execution of more complex codes for a lower fee.

2.1 Informal Description of a Satoshi Blockchain

A blockchain, as described by Satoshi Nakamoto[N+08], is a realization of a PDL and therefore a self-organizing system where each agent is supposedly aware of every valid message broadcast and shared within the system. The agents are called 'full nodes' when they keep and store the entire transcript, or 'light nodes' when they only memorize the most recent state. Achieving consensus on the most recent state among the agents is the main objective of the self-organizing system. Consensus arises from continuously receiving, validating and passing on of self-approved messages. Messages are collected and bundled periodically in order to cause network-wide database updates. Messages that make it into such a bundle (aka. block) have a chance of becoming persistently stored by each node. The next question naturally arising is about the systems bundling mechanism, particularly how it creates such a bundle and under what circumstances. In Bitcoin, those who create bundles are called 'miners'. Each block except the first one, references its predecessor. The miners compete against each other as they try to be the first one to publish a block referencing the most recent block. If a miner succeeds in creating and publishing a block before his competitors do, they can draw from some predetermined self assigned reward once their block has been accepted by the majority and it became a link in the chain. This competition has two sides. Loosely speaking, it gives rise to the system's security but the moment two or even more blocks reference the same predecessor, the system is weakened as consensus has to be reached on which

block to keep. To maximize the constructive aspects of competition and minimize the destructive ones, mining is tied to some computationally hard task which terminates with a certain probability per computation step. As the probability dynamically adapts to the average mining activity of the network to keep the block creation time constant, the likelihood of having two colliding blocks is reduced. This leads us to the following observations: A blockchain is a communication network where peers only trust and propagate what their own reasoning lets them conclude. Therefore there is no need for authority or hierarchy since only consistent argumentation leads to acceptance. Second: A mining mechanism coupled to computational power implies that the one with the lowest electricity costs, easiest access to mining hardware and least concerns about or constraints on environmental damage has a competitive advantage. The tools required to realize the world's first blockchain, were already there and just waiting for someone like Satoshi to combine and reuse them in a new, unforeseen fashion. The information theoretical problem she thereby tried to solve is a much older one, namely, how to achieve consensus in an adversarial environment without a common source of trust. Commonly one finds references to the Byzantine generals problem, where generals of the Byzantine army camp around an enemy city with the intent to conquer it. The generals must find a way to agree upon a battle plan by exchanging messages, knowing that some generals might be traitors trying to ruin their attack by holding messages back or even spreading false ones [LSP82]. John von Neuman's work on 'how to build a reliable system from unreliable components'[Pip90] addresses the same issue but in a different context, considering adversaries as dysfunctional components in the ecosystem. One major difference between Satoshi's work and previous work in this area is that her protocol does not need to know its participants in advance and that it dynamically adapts to allow for variations in the number of users. The 'only' requirement to participate is a PC and Internet access. Since it is a tool that aims to achieve consensus on the ordering of events in time without a common source of trust, a Satoshi-blockchain can be interpreted as a mapping of the physical world's arrow of time into a system of connected computers. Nowadays almost every general-purpose computer has an inbuilt clock but it has no 'understanding' of the concept of time. All it does is periodically incrementing a counter. The device cannot 'tell' the difference if we change its local time or manipulate the frequency of its clock. Likewise, an isolated person in a darkened room tends to lose his sense of time very quickly but usually keeps his/her internalized concept of growing entropy. Consequently, a person who lost his/her sense of current time and date is still able to order events chronologically if the growth of entropy is evident, like most people would label a broken vase as 'after' and the unbroken one as 'before'. A blockchain in that sense is quite similar to a series of pictures where the growth of entropy is almost as immediately detectable for a computer as it is for us humans with the broken and unbroken vase. Trying to convince a node about a state that has less 'entropy' than the current one who is believed to be correct is very hard, like it is hard to convince someone that a vase is intact, after the person has seen it smashed into pieces. Another masterpiece of Satoshi's work is her argument about the system's security. In its core, it is heavily based on human greed, even though it is commonly not phrased this way. The preferred wording is rather

'game theoretic arguments based on the rational behavior model'. The rational behavior model in principle states that the majority of humans considers more profit to be better than less profit and will therefore make decisions in favor of profit maximization. Since this hypothesis appears plausible enough for economists to build their theories upon, blockchain enthusiasts quickly skip over it and end up arguing about the cryptographic security assumptions. It should be noted, however, that these assumptions and their implications are very different. Cryptographic assumptions are basically mathematical problems which are easy in one direction but assumed to be disproportionately difficult to revert for anybody in the other direction. They are called assumptions since there is no mathematical proof guaranteeing their difficulty. We rely on them solely because so far no one was able to disprove them although they were subject to scientific research over years or even decades. The day may come when a mathematician refutes one of our difficulty assumptions and thereby forces us to migrate to a different one. In contrast, the rational behavior model is a sociological statistic argument with outcomes varying with culture, era and region. Since everybody can break it by acting irrationally, nobody can disprove it. Satoshi introduced this statistical argument into computer science. Besides the cryptographic assumptions, Bitcoin's security model relies on a variety of statistical requirements and assumptions. The technical ones are:

- The network is reliable and its topology has no bottlenecks.
- The probability of mining a block is directly proportional to the amount of computations performed per unit of time.

whereas the economic ones are:

- Every participant has the same purchasing power per mined Bitcoin, in order to circumvent an intrinsic trend towards centralization.
- Participants predominantly adhere to the rational behavior model.

These prerequisites are not restricted to Bitcoin. This thesis claims that every distributed ledger based on proof of work that aims for decentralization and censorship resistance needs to address all four of them. Network reliability and topology is rather unambiguous and therefore will not be discusses. Mining probability will be investigated in more detail in the following section. Concerning the 'purchasing power per Bitcoin': it is tied to political decisions concerning taxation of cryptocurrencies as well as availability and accessibility of exchange markets. Topics which are likewise beyond the scope of this paper. What remains to analyze is why the rational-behavior model should be applicable and why it influences security. The update logic promises convergence towards a fixed amount of tokens. In Blockstream's Bitcoin (formerly Satoshi's Bitcoin), the sooner one starts to participate, the bigger one's share. Due to the tokens' limited abundance, copy resistance, transferability etc. they will be understood and treated as a

good[2]. In capitalist markets they are exchangeable for goods, services and other objects of subjective value (gold, Pokemon cards, fiat etc.), and therefore the rational behavior model becomes applicable. Having more tokens is better than having fewer. Honest users are encouraged to cooperate as their tokens become worthless if they play by individual rules and thereby lose market acceptance. Likewise, it disincentivizes attackers. Because of the unrestricted accessibility, a rational attacker takes into account that the time and energy spent attacking could be used to participate instead and thereby create bigger profits. In other words, cryptocurrencies create an environment where symbiosis of the weak outweighs parasitism. And like with parasites, an attacker who dominates the system might create short-term profits but will kill its host in the long term. Of course, this analysis barely scratched the surface of a highly intricate system, but by now it is hopefully clear why distributed ledgers are worth studying. We need to understand the inherent trade-offs between efficiency and decentralization and consider political, sociological, economical and environmental aspects. Currently the carbon footprint of these systems is enormous (estimates range from 21 to 53 $MtCO_2$ in 2018 for Bitcoin alone[SKG19, LLP+19]), while the degree of decentralization is rather an illusion as Bruce Schneier pointed out[Sch19]. Greg Maxwell (BTC) called it *"probably the most insanely inefficient mode of communication ever devised by man"* [Max], while Rick Dudley (ETH) doubts the promise of a world computer and Vlad Zamfir (ETH) argues why encoded law and blockchain governance can lead to very undesirable societies as well [Dud, Zam]. To summarize this chapter: Trust forms the foundation of every society. Trust gives rise to simplicity. From simplicity arises efficiency. A world without trust we call blockchain.

2.2 Formal Definition of a Satoshi Blockchain

Definition: A state $\mathcal{S}$ is a data-structure similar to a directed rooted tree, where $V(\mathcal{S})$ gives the set of all nodes (aka. blocks), and $\mathsf{predecessor}(a)$ gives the predecessor node of $a \in V(\mathcal{S})$. The root vertex or node base we call *genesis* block $\mathcal{G}$, where $\mathsf{predecessor}(\mathcal{G}) := \mathcal{G}$. A state supports the following functions:

- $\mathsf{Chains}(\mathcal{S}) \to \{\mathcal{S}_i\}$ takes a state as input, and returns the set of all simple-directed-chains from each of the trees leafs that have a path to the *genesis* block $\mathcal{G}$.
- $\mathsf{value}_{\mathsf{Block}}(b) \to n$ is a function that takes a single block b as input and returns an estimate on the amount of spent resources $n \in \mathbb{N}$ in order to create the block b. The feasibility of this function al is essential and will be treated in greater detail below.
- $\mathsf{value}_{\mathsf{Chain}}(\mathcal{S}) \to n$, gives an estimate on the amount of spent resources $n \in \mathbb{N}$ in order to create the state $\mathcal{S}$. $\mathsf{value}_{\mathsf{Chain}}(\mathcal{S}) := \sum_{v \in V(S)} \mathsf{value}_{\mathsf{Block}}(v)$

[2]Digital gold is a frequent analogy to describe Bitcoin, even though it should be used with care as the element gold and its abundance is probably not a subject of what humans or computers agree upon.

- $\mathsf{Lead}(\mathcal{S}_A) \to \mathcal{S}_B$ takes state $\mathcal{S}_A$ and returns the branch (simple directed chain) where most resources were spent on via: $\mathsf{Lead}(\mathcal{S}) := \max_{\mathsf{value}_{\mathsf{Chain}}(a)} \{a \in \mathsf{Chains}(\mathcal{S})\}$.
- $\mathsf{Value}(\mathcal{S}) := \mathsf{value}_{\mathsf{Chain}}(\mathsf{Lead}(\mathcal{S}))$.
- The disagreement-factor $\mathsf{df}(a, b) \to r$ gives the ratio $r \in [0, 1]$ on how much two states a and b differ via:

$$df(a, b) := \frac{|(V(a) \setminus V(b)) \cup (V(b) \setminus V(a)|}{|V(a)| + |V(b)|} = \frac{|(V(a) \triangle V(b))|}{|V(a)| + |V(b)|}. \tag{2.1}$$

Definition: A blockchain client $\mathcal{C}$ is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Ver}, \mathsf{Ins})$ with:

- $\mathsf{Gen}(\mathcal{S}_A, \mathcal{S}_B) \to \mathcal{T}_{\mathcal{S}_A \rightsquigarrow \mathcal{S}_B}(\cdot)$, is a probabilistic algorithm that takes two states as input and outputs a transition function with

$$\mathcal{T}_{\mathcal{S}_A \rightsquigarrow \mathcal{S}_B}(\mathcal{S}_x) \quad = \begin{cases} \mathcal{S}_B & \text{iff } V(\mathcal{S}_x) \supseteq V(\mathcal{S}_A) \\ \perp & \text{otw.} \end{cases}$$

  Running this algorithm will be referred to as *mining*. Its not a necessity that Gen terminates at all.

- $\mathsf{Ver}\,(\mathcal{S}_a, \mathcal{T}_{\mathcal{S}_b \rightsquigarrow \mathcal{S}_c}(\cdot)) \to 0/1$, returns 1 iff the call of the given transition function $\mathcal{T}_{\mathcal{S}_b \rightsquigarrow \mathcal{S}_c}(\mathcal{S}_a) \neq \perp$.
- $\mathsf{Ins}\,(\mathcal{S}_a, \mathcal{T}_{\mathcal{S}_b \rightsquigarrow \mathcal{S}_c}(\cdot)) \to \mathcal{S}_x$, takes a state and a transition function and returns

$$\mathcal{S}_x \quad = \begin{cases} \mathcal{S}_c & \text{if } \mathsf{Ver}(\mathcal{S}_a, \mathcal{T}_{\mathcal{S}_b \rightsquigarrow \mathcal{S}_c}(\cdot)) = 1 \wedge \mathsf{Value}(\mathcal{S}_a) < \mathsf{Value}(\mathcal{S}_c) \\ \mathcal{S}_a & \text{otw.} \end{cases}$$

  .

Definition: An agent is a quadruple $(\mathcal{C}, \mathcal{S}, N, R(t))$, where $N \subseteq \mathcal{B}$ describes the agents neighborhood e.g. its directly connected peers (other agents), which are a subset of all agents $\mathcal{B}$. We note that an agent shall not be connected with itself. $\mathcal{C}$ is the blockchain client he uses and $\mathcal{S}$ the current agents state. $R(t) \in \mathbb{N}$ is the resource number. It tells how much resources at time t an agent spends in order to mine. If we do not not write the time dependence explicitly we mean the present time (right now). $R = x$ means, the agent is just about to spend x resources in order to mine. At this point we do not dare to define what a resource is and how an order relation can be assigned onto them for the comparability requirement. We rather rely on the readers intuition and note that a resource is probably something finite or unequally distributed such as computational power, chocolate, storage space etc.. Knowledge is not a considerable resource in this sense as it is basically cost-free to replicate and cannot be expended. We exclude the possibility that for example Bob with $R = 42$ knows a secret signing key, Alice with

$R = 7$ does not know. Therefore our following blockchain definition will not, and cannot, cover proof of stake protocols.

Definition: A Blockchain $\mathcal{B}$ is a set of $p \in \mathbb{N}_{>1}$ distinct agents $\mathcal{B} = \{(\mathcal{C}, \mathcal{S}_i, N_i, R_i)\}_{i=1}^{p}$. The agents share the same blockchain client $\mathcal{C}$, genesis block as root node and order relation over R e.g. they expend the same type of resource in order to mine. $\bigcup N_i = \mathcal{B}$ forms a fully connected graph/network.

As explained in the beginning of section 2.1, we want to minimize destructive competition among miners while maximizing constructive competition. Satoshi tried to achieve this by turning Gen into an algorithm that terminates with a certain probability after on unit of time. She aimed for the following: Let $\mathcal{S}_X, \mathcal{S}_Y$ be two different states where $V(\mathcal{S}_X) \subset V(\mathcal{S}_Y)$ and $\mathsf{Lead}(\mathcal{S}_X) < \mathsf{Lead}(\mathcal{S}_Y)$. The probability that $\mathsf{Gen}(\mathcal{S}_X, \mathcal{S}_Y)$ terminates, if run by agent j after one unit of time is

$$\mathbb{P}_j[\mathsf{Gen}(\mathcal{S}_X, \mathcal{S}_Y) \downarrow] \propto R_j.$$

Since

$$\sum_i \mathbb{P}_i[\mathsf{Gen}(\mathcal{S}_X, \mathcal{S}_Y) \downarrow] = c \in [0,1] \implies$$

$$\mathbb{P}_j[\mathsf{Gen}(\mathcal{S}_X, \mathcal{S}_Y) \downarrow] \propto R_j \left(\sum_{(\cdot,\cdot,\cdot,R_i)\in\mathcal{B}} R_i \right)^{-1},$$

the probability that agent j mines one or more blocks is proportional to the fraction of resources j is spending relative to what all other miners are spending. We observe that the probability for $\mathsf{Gen}(\mathcal{S}_X, \mathcal{S}_Y)$ to terminate ($\downarrow$ indicates convergence/termination, $\uparrow$ divergence of a program), depends on the summation over all resource numbers in the network. A number which cannot be known for several reasons. First: the resource numbers as well as number of miners are not static. Miners come and go and their willingness to spend resources vary. Second: knowledge about this number as stated above would only be accessible for someone with a global view, which supposedly should not exist in a decentralized network. So the question is now, how to approximate this number with a deterministic algorithm s.t. every miner knows his chance of mining a block by only looking at his local leading state. Nakamoto solved this issue by attaching a proof to each block v s.t. if K resources where spend in order to create v then $\mathsf{value}_{\mathsf{Block}}(v)$ returns a value converging to K as K grows. In the Bitcoin protocol by Nakamoto he uses 'Proof of Work' **PoW** (see [DN92, RSW96, B$^+$02]) to establish this. This PoW we call **useless PoW** if no market with demands for such solved PoW instances beyond the corresponding blockchain infrastructure exists[3]. A PoW we call useful, if it is not useless.

[3]We do not claim that useless PoW exists. Hash Based PoW (as in BTC,ETH,LTC..) yields huge profits for those who sell the mining equipment (GPUs, ASICs,..). Therefore a miner who owns and/or has interests/stakes/shares in mining-equipment-production kills two birds with one stone. They have

The uselessness of this PoW guarantees in some sense that two miners with equivalent R have the same profit on the long therm and therefore equivalent growth potential. Useful proof of work constructions, such as 'proof of protein folding', might lead to different long therm profits since miners with interests in the pharmaceutical industry potentially benefit more from the PoW result then those without. Arguing about the blockchains stability (defined in section 2) is far simpler if the PoW is useless. If such a proof of work is attached to each mined block, the past lets an agent conclude how much resources were spent on average. The verification Ver now can be extended to return 0 in case the value of a new block deviates from some expectation value, and the miner now knows his chances of mining since he can compute the amount of resources spent over some time interval starting from the least leading block and compare it with his own resource number. An attacker who manages to cut of all neighbors from an honest node for a short period of time, still has to spend as many resources in order influence the targets state, as if the target was not disconnected at all. Furthermore, since the chance for Gen to terminate, depends linearly on the expend of a resource, it implies that sybil attacks conducted in order to increases chances in mining a block (an attacker controlling multiply seemingly independent nodes) are not useful, since the attacker will have to distribute his resources amongst his sybil identities. Attacks on the network topology however remain untreated in our simplified description of a blockchain protocol. An attacker who manages to subdivide the network, subdivides the resources and therefore increases his probability to fork.

For an honest agent $a \in \mathcal{B}$ with his state $\mathcal{S}$, it always holds that:

$$\mathsf{Ins}(\mathcal{S}, \mathsf{Gen}(\mathcal{S}, \mathcal{S}_x)) = \mathcal{S}_x \implies \mathsf{Ver}(\mathcal{S}, \mathsf{Gen}(\mathcal{S}, \mathcal{S}_x)) = 1 \tag{2.2}$$

In order to define a Satoshi blockchain, we introduce the *fork-game* $\Pi^{\mathcal{A}}_{fork}$, where an adversary $\mathcal{A}$ tries to fork (e.g. alter the past) of an honest users state $\mathcal{S}$, as:

cheap access to mining-equipment. They dominate the networks hash-rate. If others want to keep up, they need to by the equipment from their competing miners who control the price of the equipment. During the gold rush those wo sold the shovels got rich, not those who mined[Lor].

Π_{fork}

Adversary $\mathcal{A}$ with resource number $R_{\mathcal{A}}$		honest Agent $\mathcal{H} \in \mathcal{B}$ with state $\mathcal{S}; i = 0$
connect to $\mathcal{H}$	$\xrightarrow{\mathcal{A}}$	add neighbor: $N := N \cup \mathcal{A}$
cut of $k \in \mathbb{N}^+$ leading blocks from s_0, create $\mathcal{S}_{\mathcal{A}}$ s.t. $\mathsf{df}(\mathsf{Lead}(\mathcal{S}_A), s_0) > 0$	$\xleftarrow{s_0}$	send best current state $s_0 \leftarrow \mathsf{Lead}(\mathcal{S})$
compute and send $b \leftarrow \mathsf{Gen}(s_0 - k, \mathcal{S}_{\mathcal{A}})$	$\circlearrowleft \begin{cases} \xrightarrow{\mathrm{b}} \\ \xleftarrow{s_i} \end{cases}$	calls $\mathcal{S} := \mathsf{Ins}(\mathcal{S}, b), i := i + 1$ sends best current state $s_i \leftarrow \mathsf{Lead}(\mathcal{S})$

The attacker wins (successfully forks the chain) if: $(\mathsf{Lead}(\mathcal{S}_{\mathcal{A}}) = s_i) \wedge (s_i \neq s_0)$ where:

$$s_i = \begin{cases} \mathsf{Lead}(\mathcal{S}_{\mathcal{A}}) & \text{if } \mathsf{Ins}(\mathcal{S}, b) = \mathcal{S}_{\mathcal{A}} \\ \mathsf{Lead}(\mathcal{S}) & \text{if } \mathsf{Ins}(\mathcal{S}, b) = \mathcal{S} \wedge \mathsf{Ver}(\mathcal{S}, b) = 1. \\ \perp & \text{otw.: no response} \end{cases} \tag{2.3}$$

The probability for an adversary $\mathcal{A}$ to win the fork game Π_{fork} is described by a function f whose behavior depending on its input arguments is decisive for our intent to classify:

$$\mathbb{P}[\Pi^{\mathcal{A}}_{fork} = 1] = f(R_{\mathcal{A}}, s_{i-1}, \mathcal{S}_{\mathcal{A}}) \tag{2.4}$$

Note that the hones agent during the game is allowed to receive updates from his remaining neighborhood. Hence f in equation 2.4, depends on s_{i-1}, since this is the actual state, the adversary has to fork. If $N_{\mathcal{H}} = \{\mathcal{A}\} \wedge R_{\mathcal{H}} = 0$, then $s_0 = s_{i-1}$. Assuming that all states are valid, we observe and define:

- $\forall r \in \mathbb{N}, \mathcal{S}_A, \mathcal{S}_B : \mathsf{Value}(\mathcal{S}_A) > \mathsf{Value}(\mathcal{S}_B) : f(r, \mathcal{S}_A, \mathcal{S}_B) = 0$. Forks can only succeed if they increase the chain value.
- if $\exists \mathcal{S}_A \forall \mathcal{S}_B : \mathsf{Value}(\mathcal{S}_B) > \mathsf{Value}(\mathcal{S}_A) : \forall r \in \mathbb{N} : f(r, \mathcal{S}_A, \mathcal{S}_B) = 0$ the blockchain has stable states or so called block-(height)-finality, where we call $\mathcal{S}_A$ stable state or **milestone**.
- if $\forall \mathcal{S}_A, \mathcal{S}_B : \mathsf{Value}(\mathcal{S}_B) > \mathsf{Value}(\mathcal{S}_A) : \forall a, b \in \mathbb{N}, r \in \mathbb{Q} : (a \geq b) \wedge (a = b \cdot r) : |f(a, \mathcal{S}_A, \mathcal{S}_B) - r \cdot f(b, \mathcal{S}_A, \mathcal{S}_B)| < \mathsf{negl}\Big(\mathsf{value}_{\mathsf{chain}}(\mathsf{Lead}(\mathcal{S}_B) \setminus \mathsf{Lead}(\mathcal{S}_A))\Big)$, then the blockchain is **linear-balanced**. This means, if peer A has r times bigger resource number then B then the chance of winning the fork game for A is r times bigger as the chance for B with deviations that shrink rapidly as the value of the update increases.

Definition: A Satoshi Blockchain uses useless PoW, is linear-balanced, has exactly one milestone (the genesis Block) and is permissionless since the probability for winning the fork game does not depend on the adversaries identity.

CHAPTER 3

Privacy Enhancing Technologies for PDLs

In this section we provide an overview of current techniques used to establish anonymity on a public distributed ledger. We therefore group a transaction into three parts: Origin, Content and Recipient (see figure 3.1) and cover how privacy relates to them. Also we will use the terms:

- **Unlinkability**: for any two outgoing transactions it is 'impossible' to prove that they were sent to the same person.
- **Untraceability**: for each incoming transaction all possible senders are equiprobable.
- A **transaction graph** holds the information which address sent what amount to whom, ordered in time.

3.1 Anonymity of Transaction Origin

As Bitcoin gained popularity, the claim that: 'it is a secure and private payment system', spread. Time proved both statements to be wrong. Fatal protocol flaws were found as well as the busting of the Internet platform 'Silk Road' revealed that transaction graphs are already being analyzed by investigators in order to trace and track people. Not only does Bitcoin allow the creation and investigation of such a transaction graph, it literally is one big transaction graph which is needed to check the validity of each transaction. The token reward and exchange system (Bitcoins) build on top of the bitcoin blockchain therefore lacks fungibility, this means that two bitcoins may not have the same value since their history is public knowledge. As a consequence, miners and merchants

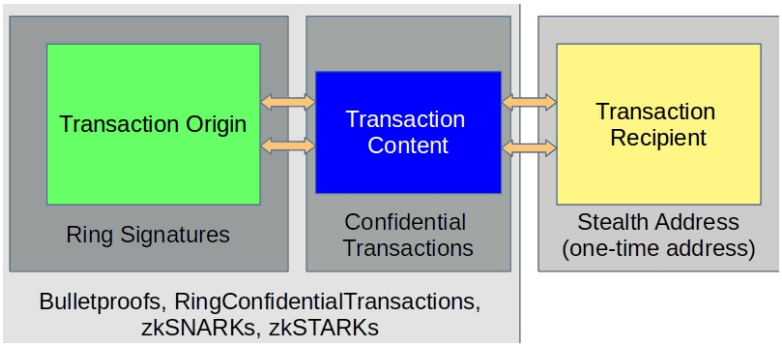


Figure 3.1: Transaction consisting of the three main components Origin, Content and Recipient, along with technologies that aim to enhance privacy on each of these.

can reject transactions that are related to or originate from some address they dislike. This lack of untraceability was noticed by the community around bitcoin very soon and addressed by Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell and Pieter Wuielle[PBF+18], Nicolas van Saberhagens (a pseudonym) and others. They describe a ring signature scheme [Noe15] which has its roots in David Chaums and Eugenes work [CVH91]. It allows a signer to hide among a finite set of public keys $K = \{P_i\}_i$, yet prove that they know the discrete logarithm: I know x st. for some $P_i \in K : G^x = P_i$. The cryptographic assumptions are based on the hardness assumption of the discrete log problem on elliptic curves and do not require any stronger assumptions (besides the randomness and collision resistant hash-function existence assumptions). The consequences and problems of this scheme when implemented in a PDL are:

- The concept and conveniences of a unspent transaction output (UTXO) set for a quick transaction verification cannot be applied any more. The reason for this is the immediate consequence of the fact that each node learns at most that one out of a bunch of outputs has been spent. It is therefore not possible to mark a transaction as being spent but with probability $|K|^{-1}$.
- Instead, a list of so called key images (a list of elliptic curve points), has to be maintained and loaded into some fast accessible memory space. It grows with each transaction, cannot be pruned, and requires a lookup each time a transaction is validated to permit a double-spend. Hashmaps however allow to perform this lookup in quasi constant time.
- The transaction size grows linear with $\mathcal{O}(|K|)$, i.e. the more cosigners among whom a sender chooses to hide, the bigger the transaction and therefore the global validation time, chain storage requirement and transaction fees.
- The signer's choice of the cosigner set is a very delicate matter regarding forward secrecy. The selection algorithm needs to consider the time distribution underlying

the set, since for each output $P_i \in K$ the block-height (and therefore its age) is publicly known.

- For the ecosystem's health and individual privacy it is recommended to re-send coins to oneself from time to time, such that fresh honest outputs are created on chain. It reduces the risk of becoming 'too old to be spent without causing suspicion'.
- To de-anonymize, an attacker can create a huge set of outputs and thereby increase the probability of being selected as a cosigner. Analysis on this topic was done by the Monero Researchers [AM15].
- If a person decides willingly or unwillingly to reveal all or some of their spendings, all transactions that included that output once as cosigner, have their untraceability reduced by the fraction this output accounted for. This point is somehow equivalent to the point above.

Alternatively to ring signatures one can use zkSNARKs as we will describe in more detail in section 4. To name a few properties of this privacy enhancing technique:

- The anonymity set derives from the entire ledger history and grows with each transaction.
- They provide perfect zero knowledge that is: under no circumstance an adversary can extract the sender's identity solely from a transaction.
- They depend on stronger cryptographic assumptions then the above scheme (see chapter 4.4) and require a trusted setup, which is really hard to establish in an environment based on mistrust.
- The computational effort and space they require currently exceeds a common light-weight device's capabilities.

3.2 Anonymity of Transaction Content

Cryptocurrencies like Bitcoin require the sender of a transaction to prove ownership of a UTXO. As a consequence, each recipient of a transaction knows the amount the sender held before, as well as the coin's entire history. Such systems are not fungible and therefore inherently distinct from central bank money, where law obliges merchants to treat each central bank Euro equivalently. Bitcoins with different histories may find different degrees of acceptance among merchants and therefore differ in market value. As one possible solution, confidential transactions where introduced. They are based on non-interactive homomorphic commitment schemes where roughly speaking $C(x)+C(y) = C(x+y)$ holds, where C is some encryption function. Note that this property is inherent to most elliptic curve based cryptographic constructions. This enables verifiers to convince themselves

that $\sum inputs - \sum outputs = C(0)$ without getting to know the actual values being transfered. The difficulty is to prove that each output is within a certain positive range $[0..2^n]$. These so-called range proofs used to drastically blow up transaction size linear in $\mathcal{O}(n)$ [PBF+18], since they were based on the same ring signature scheme we introduced before. This problematic linear growth can be avoided, however. So called 'Bulletproofs' by Bünz manage to significantly reduce space requirements down to $\mathcal{O}(log_2(n))$ as can be seen in figure 3.2, and additionally enable batch validation, which is highly beneficial regarding fast block validation [BBB+18]. Beside confidential transactions Bulletproofs allow general NP statements to be proven without the need for a trusted setup, in contrast to zkSNARKs which are a further alternative we treat explicitly in chapter 4. These zkSNARKs have very small and constant space requirements and are quasi constant in verification time, independent of the statements complexity which is to be proven. Their implementation and setup however, is tedious work and very error prone. From a technical perspective, the major downside of confidential transactions is

- an implementation error or unsound scheme. It potentially could lead to silent inflation, i.e. an attacker creates tokens 'from nothing'. Since the attacker's account is confidential, the network is not able to spot this. If the minted tokens are deeply rooted and distributed in the transaction graph, not even a hardfork can undo the damage. As a consequence, the entire protocol can loose its purpose and even endanger other tokens if they have a tight market coupling. An attacker will probably try not to reveal his 'cash cow' to others, and sell the minted tokens in small amounts over a long period of time. It is an intriguing open question how long it takes 'the market' to react on silent inflation assuming the adversary follows an ideal selling strategy. As a solution, the protocol could require that periodically all users are incentivized or enforced to reveal their account. If the scheme had been exploited and coins were minted, this could be noticed. A weaker countermeasure can be taken by enforcing the blockrewards to be transparent. If the majority of trading platforms collaborates in order to monitor the amount of circulating tokens, an overrun of the expected value (sum over all blockrewards) could trigger an alarm. Another question is whether the damage can be reversed. If the transaction graph is entirely confidential as well, or the attacker's minted tokens widely distributed among merchants, it will be hard.

3.3 Anonymity of Transaction Recipient

Satoshi suggested to generate a new address each time a payment is conducted. However, this has the obvious downside that the two parties either have to exchange an address generation scheme in advance, or they exchange payment data just in time. Both methods come along with security risks as well as inconvenient additional efforts. Furthermore, if a payment requires to merge two or more unspent outputs, all efforts spent for the preservation of privacy immediately vanish. A solution to this problem was introduced by

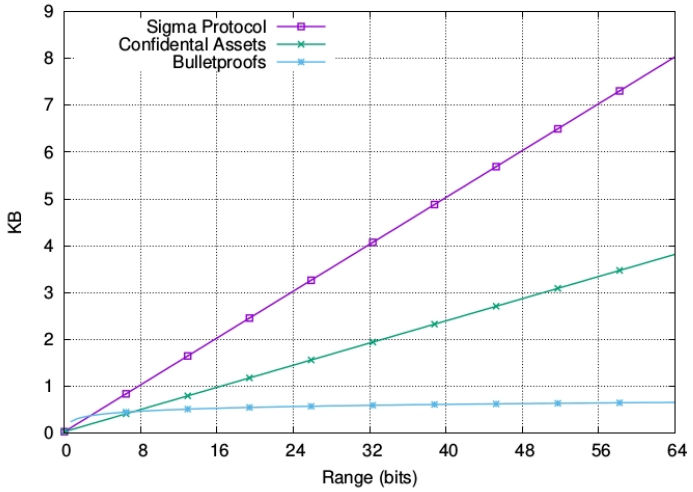


Figure 3.2: Space requirement of various constructions of range proofs with increasing number of bits. The graph was taken from [BBB+18].

an author under the pseudonym of Nicolas Saberhagen [SN13]. She called the commitment scheme 'single address to receive unconditional unlinkable payments' yet it is now rather known as *'stealth addresses'*. A stealth address is a pair of curve points (A, B) where the corresponding secret exponents are (a, b) s.t. $A = G^a$ and $B = G^b$, with G as the curves common generator. The sender takes both A and B, picks a random group element r and creates an output P that is indistinguishable from uniform random due to r. The only way to spend P is by proving knowledge of its discrete logarithm, which requires the knowledge of a and b. Some properties of this scheme are:

- A party holding one of the two secrets is able to remove the randomness r and identify (A, B) as the recipient, however is unable to spend P. This can be used for legal purposes or by NGOs to enable their supporters to view their incomings, as well as it enables light clients.
- every transaction in the network requires two ECC multiplications and one hash operation to be performed before a wallet is able to tell if it is the intended recipient. However, this validation does not directly affect the network's throughput rate since it is performed by the wallet and not necessarily during block/transaction validation.
- the addresses are twice as big as a common Bitcoin address, and each transaction additionally needs to contain the obfuscation point $R = G^r$.
- The scheme can be extended, as Noether an Goodell [SN17] did, to support the creation of subaddresses. They enable the user to create arbitrarily many distinct

public key pairs that are indistinguishable from uniform random, while the wallet's computational effort remains constant.

CHAPTER 4

An Introduction to Pairing-Based Preprocessing zkSNARK

zkSNARKs are a general purpose proof system that allow succinct verification of any NP statement, with proofs of fixed size, independent of the statements complexity. They are a powerful technique to bootstrap the scalability issue all public distributed ledgers are facing today, while providing privacy aside. The idea is that an entire block could potentially consist of transaction statements, the predecessor proof, the difficulty, the nonce, and its zkSNARK only. The proof then guarantees with high probability that the PoW as well as the individual transaction requirements are met and that the predecessor proof is statistically knowledge sound. In this section we explain how non-interactive zero-knowledge proofs of knowledge **zkSNARK** [NY90, BFM88] work, by telling a story where the two protagonists, Bob and Alice, face a problem and the steps towards a solution introduce these kind of proofs. Since there is very little literature on the entire process chain of zkSNARK creation that helps developers to grasp what they are dealing with, and as Christian Reitwießner pointed out: 'most of them become a bit hand-wavy at some point', we here try to bridge a gap between mathematics and programing. Helpful sources to begin with are Christian Reitwießner article[Rei16], the zCash foundations explanation on their web-page[zca19], the blog posts of Vitaliks Buterins, and the lectures of Ben-Sasson, which currently can be watched on YouTube. As Goldwasser et al. [GMR89] first introduced the notion of 'zero knowledge', their work is also worth studying. The intuitively understandable writing on zero knowledge by Jean Jacwu [QQQ+89] is quite nice to start with as well. However, in order to understand how modern zkSNARK constructions work in detail, we suggest to study the work by Parno et al. [PHGR13] and the well written paper on square span programs by Danezis, Fournet, Groth and Kohlweiss [DGM14]. For a more advanced study, we recommend the works from Eli Ben-Sasson et al. such as the work on 'Scalable Zero Knowledge via Cycles of Elliptic Curves'[EBSV17], where they explain the beautiful concept of recursive

zkSNARK composition. The acronym zkSNARK constitutes:

- **z**ero-**k**nowledge: The proof reveals nothing besides the correctness of the statement
- **S**uccinct: The proofs are compact and computationally light to verify.
- **N**on-interactive: Proof generation does not require message exchange between prover and verifier.
- **AR**guments of **K**nowledge: the proof system might not be secure against computationally unbound provers.

There is a multitude of sub-classifications of zkSNARKs. We will investigate the so-called pairing-based and preprocessed zkSNARK construction. Pairing-based indicates a method used to bootstrap succinct verification and preprocessed denotes that the key pair used for proof generation and verification depends on the arithmetic circuit whose knowledge of a satisfiable assignment of values is proved.

The first constructions of SNARK protocols were inspired by the PCP theorem which shows that NP statements have 'short' probabilistically checkable proofs[May16]. We will look at a zkSNARK model with a pre-processing state by Gennaro et al[GGPR13].

4.1 Verifiable Computations

Imagine an English teacher and a student. The teacher hands out a gap text of some form like *"A* $\langle 1.Subject \rangle$ *goes* $\langle 1.Predicate \rangle$*. After* $\langle 1.Predicate \rangle$, $\langle 1.Subject \rangle$ *meets* $\langle 2.Subject \rangle$*..."*, and wants the student to fill out the gaps according to the grammatic and referential requirements e.g. once the student picked a subject for a $\langle 1.Subject \rangle$ gap, the student must always use the same whenever there is gap with $\langle 1.Subject \rangle$ again. Verifiable computations enable the student to create a proof that the blueprint has been filled out correctly which the teacher can verify supposedly in a time quicker then checking entry by entry. For example: the student could fold the paper s.t. all elements of the same type are in alignment. The teacher then holds the folded paper against the light and immediately sees if all of them are equal. The student had the effort of folding the gap text properly. A zkSNARK is quite similar to to this grammatic blueprint example. The gap text is now just a simple computer program. Elements can be composed with arithmetic operations $(+, -, \cdot, /)$ to get a new one. The gap text is now a 'math homework', where the student computes simple linear equations and often reuses results as input for the next equation. For example *"*$\langle a \rangle + \langle b \rangle = \langle c \rangle$*.* $\langle c \rangle \cdot \langle a \rangle = \langle e \rangle$*..."*. Folding the homework to make it succinctly verifiable becomes more abstract now, but the idea somehow remains the same. If the student manages to fold the homework in such a way that the teacher learns nothing besides the fact that the student did everything as required, the homework becomes a 'zk-homework'.

4.1.1 Polynomial Divisibility and Uniqueness

Like every good cryptography tale, we start by introducing Alice and Bob. Alice works in a research center where they gather huge amounts of data for the purpose of mass surveillance and suppression of non-compliant individuals. Bob works in a different company located in Cambridge where they create statistics and perform analysis of big dataset. Every day, Alice sends a new list of one million numbers to Bob. Before Bob can start working with these numbers, he has to check if they are all within a range of 1..10. This work is exhausting, however it must be done carefully since not a single number is allowed to be out of this range. Random sampling is not an option therefore. After years of working as a professional range-checker, one day Bob decides to question the quality of his job and if he can speed up range checking process somehow, so he can finally spare some more time to count till the last prime number. So it happens that Bob gets the idea of using polynomials, as they seem to have some properties perfectly suite able for this task.

Let $P(X) = \sum_{i=0}^{d} p_i X^i$ define a polynomial, where p_i is its i-th's coefficient. The degree $deg(P) = d$ is its highest non-zero coefficients index. Each time a polynomial $P(X)$ has a passing through zero (a root), this root can be extracted and a polynomial $Q(X)$ can be determined using simple polynomial division:

$$\exists x_0 : P(x_0) = 0 \implies \exists Q(X) : P(X) = (X - x_0)Q(X). \tag{4.1}$$

Consequently every root can be extracted to get

$$P(X) = Q(X) \cdot \left(\prod_{\{x_i | P(x_i)=0\}} (X - x_i) \right), \tag{4.2}$$

a unique $Q(X)$. More precisely, for $P(x) \in \mathbb{C}[x]$ it holds that if $deg(p) = d$, it implies that $\exists d$ not necessarily different roots $a_1, a_2, ..., a_d$ such that $P(x) = (x-a_1)(x-a_2)...(x-a_d)$, which is a fundamental theorem of algebra. It follows that $P(x)$ is irreducible if and only if $P(x) = ax + b$ is a linear polynomial. Furthermore: two distinct polynomials have only countably many points of intersections (illustration in figure 4.1). Two polynomials $A(X)$ and $B(X)$ intersect at most in $max(deg(A), deg(B))$ points. Now Bob comes up with the following plan:

- Bob wants Alice to interpolate her enumerated set of $n = 10^6$ data points $\{y_i\}_{i=0}^{n-1}$. From the **Interpolation Theorem from Polynomials** we know that such a polynomial $P(x)$ of degree $deg(P) = n - 1$ s.t. $\forall x_i : P(x_i) = y_i$ exist. It can be computed using the simple technique of Lagrange interpolation, which takes $\mathcal{O}(n^2)$ time, or alternatively using an FFT based polynomial interpolation algorithm which runs in $\mathcal{O}(n \log(n))$ time. So Alice learns the coefficients c_i of this polynomial where $\forall r \in \{0...n-1\} : P(r) = y_r = \sum_{i=0}^{n-1} c_i r^i$. An important observation at this point is that interpolation of floating point data points quickly leads to rounding errors and overflows as n grows. If the points y_i are integers however and all steps of the

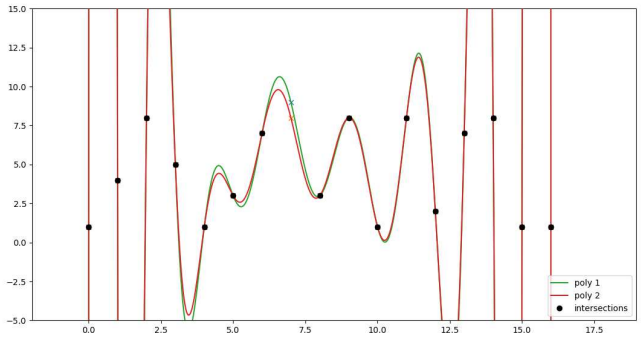


Figure 4.1: Illustrates how two similar polynomials still have only countably many points of intersection. Both polynomials were fitted through the same set of points, except for a slight deviation at $x = 7$.

interpolation are performed over some finite field, then the resulting interpolation polynomial will be correct.

- With equation 4.2 in his mind, Bob defines

$$C(X) = (1 - X)(2 - X)...(10 - X),$$

as **constraint polynomial**. He knows that if Alice's polynomial $P(X)$ is truly always between 1..10 at each point, then $C(P(X)) = 0$ must hold at each of the million points as a consequence! So Bob knows the roots of the polynomial $C(P(X))$ and he can ask Alice to extract a polynomial $Q(X)$ from this $C(P(X))$ such that the relation

$$C(P(X)) = (1 - X)(2 - X)..(10^6 - X)Q(X) = D(X)Q(X),$$

holds. D is oft called **domain polynomial**. It is zero at each x_i of a data-point. C assures that polynomials satisfying its 'constraints' are divisible. In Bob's case: divisible by D.

- The existence of Q and the fact that distinct polynomials of degree m and n have at most $max(m, n)$ intersections, gives Bob to the following idea: he asks Alice to evaluate $C(P)$ and Q at 10^9 different points. This means, he receives two huge lists of evaluated points, for example $\{C(P(i))\}_{i=1}^{10^9}$ and $\{Q(i)\}_{i=1}^{10^9}$. He then picks a random $r \xleftarrow{\$} \{1..10^9\}$, and computes $D(r)$. This is lots of work for him, but he knows that he can reuse $D(r)$ each time Alice sends him a new list, so it is worth the effort he thinks. Bob then checks if

$$C(P(r)) \stackrel{?}{=} D(r)Q(r) \tag{4.3}$$

holds. This is the so called **divisibility requirement**. If it holds, he knows that with 99.9% probability Alice knew a polynomial that has a value within the range from 1..10 at each root of D.

4.1.2 Bootstrapping with Cryptography

Amazed by the simplicity of his technique Bob starts to mistrust his intuition and gives it a second critical look. His idea may work in theory, but practically it is infeasible for several reasons:

- He might be convinced that Alice indeed knew the required polynomial, however he cannot conclude that the received lists were derived from that polynomial.
- What if Alice gets her hands on Bob's secretly chosen r? Since she knows the domain polynomial D she can now send Bob arbitrary data, as long as she crafts the r-th entry $C(P(r))$ and $Q(r)$ s.t. they satisfy $C(P(r)) = D(r)Q(r)$. So far Bob needs to carefully store and protect r from Alice, unless he does not want to recompute a fresh $D(r)$ each time. Since $deg(D) = 10^6 - 1$, computing it is more effort then checking all 10^6 points individually, this is not an option.
- The lists he expects to receive from Alice are huge. Since he wants at least 99.9% certainty, Alice has to evaluate 1000- times as many points on her polynomials as there are actual data points. The probability $\mathbb{P}$ of accepting wrong claims here is $\frac{|\text{Actual Points}|}{|\text{Eval. Points}|} = 10^6/10^9 = 0.001$. This is unacceptable since it does not scale and bandwidth is limited. So if Alice could evaluate her polynomials on a point r to get the values $C(P(r))$ and $Q(r)$, without requiring knowledge what r was, then she would not have to send him the huge lists. Luckily Bob recalls a recent conversation two strangers held next to him in the tram, where they spoke about the **blind evaluation of polynomials**. All it takes is a function H that hides its input but still allows to perform computations with the output. More precisely:
  - for most x's, given $H(x)$ it is hard to find x
  - different inputs lead to different outputs e.g. if $x \neq y \implies H(x) \neq H(y)$
  - given $H(x)$ and $H(y)$, it is possible to generate an arithmetic expression in x and y such as either $H(x \cdot y)$ or $H(x + y)$.

  Such a function can be realized in most cyclic groups $\mathbb{G}$ with generator g where the discrete logarithm is believed to be hard. The function $H : x \mapsto g^x$ then suffices, since given $s = H(x) = g^x$, it is hard to extract x, as well as for any two distinct x_1,x_2 we get $H(x_1) \neq H(x_2)$, as well as it allows to compute the sum $H(a) \cdot H(b) = g^a \cdot g^b = g^{a+b} = H(a + b)$ without the need to know a or b explicitly. This way, bob can choose r at random, and give Alice the set of group elements $\{g^{r^i}\}_{i=0}^{n-1}$ where n is the number of data-points. With this set of group elements,

Alice can evaluate her polynomials at point r, without ever learning what r is, since

$$\begin{aligned} H(Q(r)) &= g^{c_0+c_1\cdot r+c_2\cdot r^2+c_3\cdot r^3+\ldots+c_n\cdot r^{n-1}} \\ &= g^{c_0}g^{c_1\cdot r}g^{c_2\cdot r^2}g^{c_3\cdot r^3}\ldots g^{c_n\cdot r^{n-1}} \\ &= g^{c_0}(g^r)^{c_1}(g^{r^2})^{c_2}(g^{r^3})^{c_3}\ldots(g^{r^{n-1}})^{c_n-1}, \end{aligned}$$

where c_i are the coefficients of the polynomial Q. If she applies the same technique to compute $H(C(P(r)))$, all it takes to convince Bob is the original list of Points and the two blindly evaluated polynomials. He receives $g^{Q(r)}$ and $g^{C(P(r))}$ from Alice, and he computes $D(r)$ by himself. He then checks if

$$g^{C(P(r))} \stackrel{?}{=} \left(g^{Q(r)}\right)^{D(r)} = g^{Q(r)D(r)} \tag{4.4}$$

to achieve the same equality check as in equation 4.3 before, but now 'in the exponent'. The probability for the verification to holds even though Alice did not correctly extract Q from $C(P)$ drops rapidly, since the secret point of evaluation r comes from a group $\mathbb{G}$ of some big order p. The chance that unrelated polynomials coincide at r is now $\mathbb{P} = \frac{|\text{Actual Points}|}{p} \approx 0$, since $|\text{Actual Points}| \ll p$.

- She could cheat with the degrees. Bob needs to make sure that the polynomials she creates are of degree $deg(C(P)) = deg(C)deg(P) = 10^7$ and $deg(Q) = deg(C(P)) - deg(D) = 10^7 - 10^6 + 1$, but not less. One way Bob can enforce this, is via the so called **knowledge of coefficients assumption**:
  - Bob picks $\alpha \xleftarrow{\$} \mathbb{F}_p^*$ and $r \xleftarrow{\$} \mathbb{F}_p$. He creates $\left\{\left(g^{r^i}, g^{\alpha\cdot r^i}\right)\right\}_{i=0}^{n-1}$, and sends the pairs to Alice.
  - As before, Alice blindly evaluates her polynomial $a = H(C(P(r)))$, $b = H(\alpha C(P(r)))$ and $c = H(Q(r))$. Note that she can obtain b without knowing α sice

$$\begin{aligned} b = H(\alpha C(P(r))) &= g^{\alpha(c_0+c_1r+c_2r^2+\ldots+c_dr^d)} \\ &= (g^\alpha)^{c_0}\cdot(g^{\alpha r})^{c_1}\cdot\left(g^{\alpha r^2}\right)^{c_2}\cdot\ldots\cdot\left(g^{\alpha r^d}\right)^{c_d}. \end{aligned}$$

    She sends Bob the tuple (a, b, c).
  - Bob accepts if

$$\begin{aligned} b &\stackrel{?}{=} a^\alpha, \\ a = g^{C(P(r))} &\stackrel{?}{=} c^{D(r)} = \left(g^{Q(r)}\right)^{D(r)}. \end{aligned}$$

This knowledge of coefficient assumption together with the divisibility requirement implies that Alice knows her polynomials coefficients $c_0, .., c_d \in \mathbb{F}_p$ s.t. $a = \sum_{i=0}^{d} g^{c_i r^i}$, except with negligibly probability.

4.1.3 From Designated Verifier to Public Verifier

So far Bob is in charge of creating the secret points r and α. Therefore it is a 'designated verifier' scenario, since only Bob is able to verify Alice's claims. If he shares these points with others, he risks that Alice gets her hands on them too, which would enable her to create fake proofs. If instead a trusted party now computes the group elements $\left(\left\{g^{r^i}\right\}_{i=0}^{d}, \left\{g^{\alpha r^i}\right\}_{i=0}^{d}, g^{D(r)}\right)$ with a randomly chosen r and α, the scenario changes. Bob is now unable to check directly if $b \stackrel{?}{=} a^{\alpha}$ as he did before in equation 4.5, given Alice's tuple (a, b, c), since he does not know $D(r)$ this time. However if there was a function e that satisfies $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ with the property $e(g^x, g^y) = e(g, g)^{xy}$, everybody can check Alice's knowledge of coefficients via:

$$e(b, g) \stackrel{?}{=} e(a, g^{\alpha}). \tag{4.5}$$

The divisibility requirement can be checked similarly with:

$$e(H(C(P(r))), g) = e(g, g)^{C(P(r))} \stackrel{?}{=} e(g, g)^{Q(r)D(r)} = e(H(Q(r)), g^{D(r)}). \tag{4.6}$$

Luckily, with some more restrictions, such a function is feasible and can be found in the literature as **pairing function** or **bilinear map**. They allow to turn the scheme into a publicly verifiable one. They are an essential ingredient for pairing based zkSNARKs, which we treat in grater detail in section 4.4.

4.1.4 Adding Zero Knowledge

If an argument does not leak any information besides the 'truth' of its statement, it is called zero-knowledge. Shafi Goldwasser gives a wonderful informal explanation: Imagine a piece of paper with a red dot on top, and blue dot on the bottom. Alice is color blind. For her, the dots are indistinguishable and so is the rotation of the sheet. Bob wants to convince her that colors exist. Bob gives Alice the paper rotated s.t red is on top. Alice secretly flips a coin. If it is heads, she rotates the paper. If tails, she does not do anything. She hands the paper back to Bob. If Bob truly can see colors, he immediately can tell weather Alice threw heads or tails. If not, he will guess right with probability 1/2. So if they repeat this experiment, Alice can convince herself with exponentially growing confidence that Bob can see colors, she cant. In other words: Alice gains growing confidence on the 'correctness' about a statement (colors exist), without learning the concept of a color e.g. she will not be able to convince others about the statement: 'colors exist', using the same technique as Bob did, because she is still colorblind.

But now lets have a second look on what truly happened here. Assume Bob guessed right consecutively, this means he was able to recreates Alice's secret random distribution. The conclusion Alice draws from Bobs ability to do so is a much more delicate topic however. She could as well just conclude that Bob must have installed a camera behind her back, to film her tossing coins. Or even simpler: what if, there was some detail other then

the colored dot, some splotch or whatever, Bob spotted and used to decide if the paper has been rotated? In fact, Alice can draw a multitude of conclusions. The zk protocol therefore requires a challenge, where only knowledge on the required matter leads to a 100% success rate. All other possibilities must have a success rate smaller then 100%, and therefore fail at some point if enough repetitions are performed. But how can such a challenge ever be set up, since it requires to know the unknown? In fact, it cant. The protocols used in practice are such that knowledge on the required matter leads to a 100% success rate in convincing the verifier. This is called '(perfect) completeness'. All other ways which might have a 100% success rate, known or unknown, are 'blocked' by a computational riddle, which is so difficult to solve that no one should be able to take advantage of them. For that reason, the majority of SNARKs is 'statistical knowledge sound', rather then just 'sound'.

Back to our story where Bob wants to succinctly check if Alice's data points are within a certain range: Turning Bob's SNARK construction into a zero knowledge SNARK (zkSNARK), does not make much sense, since he needs the data points to work with. However, if it is just about convincing Bob, that Alice knows a set of points which are all in the range from 0 to 10, the notion of zk can be applied again. They can achieve this with different techniques. Alice could blind her polynomial by picking $\delta \xleftarrow{\$} \mathbb{Z}_p$ and extracts $Q(X)$ from

$$C(P(X)) + \delta D(X) = D(X)Q(X).$$

Another technique we use in our zkSNARK construction in section 5, is to append a what we call a randomization gate 5.1.1 to the circuit. The seemingly most efficient way however is done by blinding the proof elements directly s.t. they cancel out in the verification process, as it is done in [Gro16], which we treat in section 4.5.

4.2 From Code to Arithmetic Circuits to Polynomials

We quickly recall the scenario: Bob and Alice know a function F which outputs *true* if and only if all inputs are in range from 0 to 10. Alice now wants to convince Bob that F returned *true*, on her data points as input. Bob mistrusts Alice, but for some reason he cannot, or does not want to, check by himself. They agree on a scheme of verifiable computation which should enable Bob to succinctly check the correctness of Alice's claim.

Alice sends Bob now the list of data points and the blindly evaluated polynomials $\{H(Q(r)), H(C(P(r))), H(\alpha C(P(r)))\}$. With these values, Bob can check the divisibility requirement and convince himself of Alice's knowledge of coefficients. Everything seems to have worked out perfectly until Bob notices that Alice keeps sending him the same proof over and over again although the data points vary. At first he felt tricked by Alice but how could he blame her for the flaws in his own protocol. He needs to link the proof with the corresponding data-set. As Bob tries to wrap his head around this issue, he realizes the impossibility of his attempt. The runtime required to evaluate independent data-points cannot undercut the descriptive complexity. As long as the points are not

related or deterministic, Bob will have to continue checking them individually. So what kind of problems could be addressed with Bob's scheme?

Gennaro, Gentry Parno, and Raykova (GGPR) showed how to compactly encode computations as quadratic programs in their work [GGPR13]. This gave rise to the first relatively efficient verifiable computations (VC) as well as zero-knowledge VC schemes. In their work, they also showed how to convert any arithmetic circuit into a comparably sized Quadratic Arithmetic Program (QAP). These techniques we will now investigate, starting with the translation of computations into an arithmetic circuit. This can be done efficiently as already mentioned, however some restrictions to the languages expressibility have to made (no pointers, dynamic looping etc.).The domain specific language we designed in our own implementation[Wol] intentionally shares some similarities with golang. The idea behind this is that at some point, once the language is advanced enough, one can copy past functions written in go and provide a zero knowledge proof for them. Lets take a simple example of a function *foo* we translate into an arithmetic circuit:

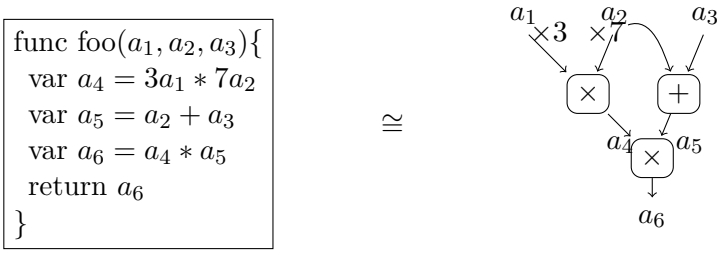


Figure 4.2: **First:** the translation from code to an arithmetic circuit. Note that each multiplication by a constant turns into a scalar signal amplification on the corresponding wire.

So we translated the simple function *foo* into an arithmetic circuit. But what about more sophisticated functions? We certainly want conditional branching and for-loops [1]. Also we want to be able to perform logic operations and comparators $(\&\&, ||, <, >, \geq, \neq ..)$ and bit operations $(\&, |, \ll, , ..)$. Currently our compiler supports static looping and branching but if a condition depends on the users input, the translation into a circuit requires more sophisticated techniques, since circuits are static objects. In such cases, an upper bound of loops has to be defined by the programmer. Lets say, we expect at most 50 loops, and after 10 we are done, the arithmetic circuit then annihilates the remaining 40 loops by multiplying every writes from inside the scope of the loop onto a variable which is outside the scope of the loop with 0. Dynamic branching, $if(condition) - else$ where the condition depends on the prover's input, is done similarly. Both, the if and the *else* statement bodies are expressed as as sub circuits. The clue is again to multiply each result from a body which is used outside of the statement with the result of the condition and then sum them up. For example we declare x and call 'if$(a > 4)$ then $x = 5$ else $x = 6$' what sets x to $x = (a > 4) * 5 + (a > 4) * 6$ after the condition. The details

[1] loops are actually not necessary when the language allows recursive function calls

on how these things can be done will not be part of this thesis and we encourage the interested reader to have a look at the work of Daira Hopwood, Parno et al.[PHGR13] or Virza [Vir14] for further investigations. Important gates to split up a field element x with $k+1$-bits into its bits x_i s.t. $x = \sum_{i=0}^{k} 2^i x_i$ can be found in Parno et al. Each x_i is guaranteed to be either 1 or 0, with the simple constraint $(x_i - 1)x_i = 0$. Once the bit representation can be asserted, one can start to imitate logic gates. For example if $a, b \leftarrow 0|1$ then $a * b$ can be understood as an AND. $1 - (a-1)(b-1)$ behaves like an OR and $1 - ab$ like a NAND.

The next step towards zkSNARKs requires to convert the arithmetic circuit into a so called Rank One Constraint System **(R1CS)**. The R1CS defines a NP-relation

$$R1CS = \left\{ (\Theta, \mathbf{a}) \;\middle|\; \begin{array}{l} \Theta = \{(L,R,O)|L,R,O \in \mathbb{F}^m\} \\ \mathbf{a} = (a_1, ..., a_m) \in \mathbb{F}^m \\ \forall (L,R,O) \in \Theta : \sum_i L_i, a_i \cdot \sum_i R_i, a_i = \sum_i O_i, a_i \end{array} \right\}$$

A program that checks if a given pair $(\Theta, \mathbf{a})$ is a valid instance $\in R1CS$, if implemented naively, requires at most $|\Theta|(3m + m^2)$ multiplications over some finite field. However the vectors in Θ, which were derived from arithmetic circuits that describe an 'average' static-size program, are very spars usually. Such instances always have $m \leq |\Theta|$, where m easily reaches orders of 10^6 if cryptographic and bit operations are performed (see zCash's verification circuit[HBHW16]). This sparsity is exploitable in various ways i.e if the vectors are represented as polynomials, storage requirement shrinks drastically during computation. The witness $\mathbf{a}$ can simply be computed in polynomial time given Θ. Extracting the witness $\mathbf{a}$ directly from the circuit is not usual, since the $R1CS$ representation is handy for some optimizations that require a global view on the circuit in order tu detect redundant parts etc.. In our implementation, the raw circuit structure is a forest derived from the compilation tree, which in turn is the result of parsing the high level language. The vectors can be handled like polynomials in order to avoid storing unnecessarily many 0's. For example $[1, 2, 0, 0, 3]$ can be represented as $1 + 2x^1 + 3x^4$. So thinking in the $R1CS$ representation of a program is useful in order to understand the way how we are going to construct quadratic arithmetic programs from them using polynomials. The circuit derived from our example function *foo* translated into R1CS is illustrated in figure 4.3.

Once we understand how to the high level language program is related to the R1CS form, we make to following observation: imagine stacking the constraint tuples on top of each other and putting a needle through each vector element. The values ordered along the needle (array) are then interpolated to form a polynomial which simply, when evaluated at the former arrays index k, outputs its k'th value. Its degree is therefor a number strictly smaller then the the array's length. So we transposed Θ and interpolated each vector.

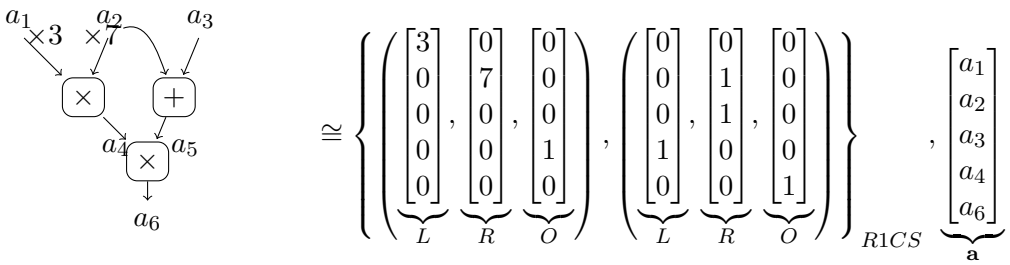
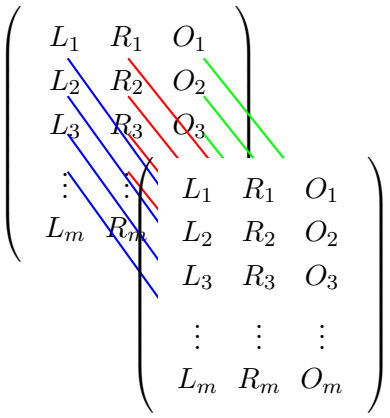


Figure 4.3: **Second:** the translation from arithmetic circuit to a rank 1 constraint system with witness vector a. We can easily convince ourselves that indeed for each constraint the relation is satisfied if we pick $\mathbf{a} = [a_1, a_2, a_3, a_4 = 7a_2 3a_1, a_6 = a_4(a_2 + a_3)]^T$. Note that the addition gate with output a_5 vanishes into the R vector of the second constraint since $\sum a_i R_{2_i} = a_2 + a_3$. In fact, additions and multiplications by constants do not affect the size of the R1CS as we will analyze in more detail in section 4.3.



The polynomial we get by interpolation of the k-th elements of the vectors L we call $L_k(x)$. For R and O respectively. In our example, this produces $3 \cdot 5 = 15$ polynomials of degree 1, which are illustrated in figure 4.4.

For these polynomials we now know by construction that if we linear combine them with $\mathbf{a}$, they still satisfy the R1CS constraints at each constraint/gate index and therefore

$$\sum_{i=1}^{m} a_i L_i(X) \cdot \sum_{i=1}^{m} a_i R_i(X) - \sum_{i=1}^{m} a_i O_i(X) = P(X). \tag{4.7}$$

So we immediately learn two things: $P(X)$ is a polynomial with $deg(P) \leq 2(n-1)$ where $n = |\Theta|$ is the number of constraints (2 in our example) and with roots in $\mathcal{A} \subset \mathbb{N}$ ($\mathcal{A} = \{1, 2\}$ in the example) as we picked them. The extracted polynomial Q from

$$P(X) = Q(X)D(X) = Q(X) \prod_{x_i \in \mathcal{A}} (X - x_i),$$

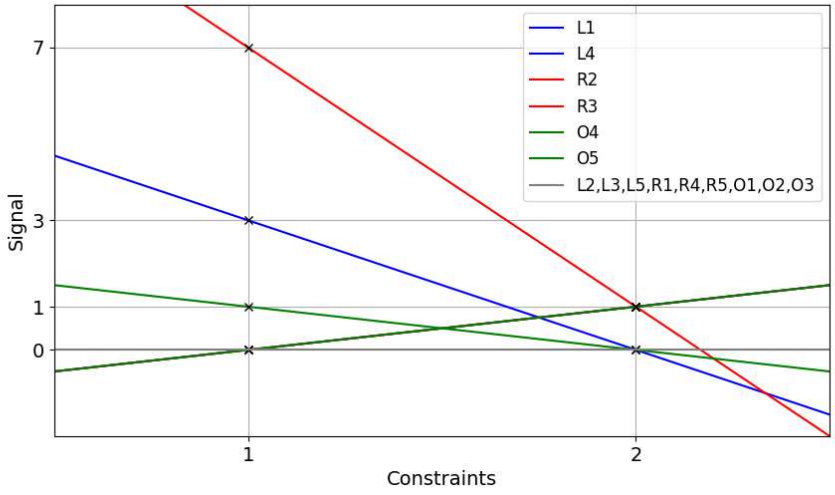


Figure 4.4: **Third:** The translation from the example R1CS (see figure 4.3), to the polynomials needed for the QAP representation.

exists and can be computed exactly as we need it.

Since we found an elegant way to translate a program into polynomials, which, if linear combined, form a polynomial of which we know the roots, lets us clarify how Bob can correlate his expectations with Alice's proof. The trick is to take equation 4.7 and split it into two parts. One part Alice is supposed to compute and the other part Bob s.t.

$$\left(\underbrace{\sum_{i=0}^{l} a_i L_i(X)}_{Bob's Part} + \underbrace{\sum_{i=l+1}^{m} a_i L_i(X)}_{Alice's Part}\right) \cdot \left(\underbrace{\sum_{i=0}^{l} a_i R_i(X)}_{Bob's Part} + \underbrace{\sum_{i=l+1}^{m} a_i R_i(X)}_{Alice's Part}\right) - \underbrace{\sum_{i=0}^{l} a_i O_i(X)}_{Bob's Part} + \underbrace{\sum_{i=l+1}^{m} a_i O_i(X)}_{Alice's Part} = P(X),$$

where $\phi = (a_0, ..., a_l)$ and $w = (a_{l+1}, ..., a_m)$. Throughout this work ϕ is called the statement. As an example we take the most obvious case: Bob has a_1, a_2, a_3 and wants Alice to compute a_6, and provide a proof for him, that she did it using a_1, a_2, a_3 as input for foo. So $\phi = (a_1, a_2, a_3, a_6), w = (a_5)$.

- Bob picks random r and sends $(\left\{g^{r^i}\right\}_{i=0}^{4}, a_1, a_2, a_3)$ to Alice

- Alice computes $foo(a_1, a_2, a_3)$ whereby she obtains $w = (a_4)$ and a_6. She then computes $L_{out} = \sum_{a_i \in w} a_i L_i(X)$, $R_{out} = \sum_{a_i \in w} a_i R_i(X)$, $O_{out} = \sum_{a_i \in w} a_i O_i(X)$, $P(X)$ and $Q(X)$. Then she performs the blind evaluation at the point r of all 5 polynomials

$$\pi = (a_6, A, B, C, D, E) = \left(g^{L_{out}(r)}, g^{R_{out}(r)}, g^{O_{out}(r)}, g^{P(r)}, g^{Q(r)}\right)$$

  She sends π to Bob.

- Bob computes $L_{in} = \sum_{a_i \in \phi} a_i L_i(X)$, $R_{in} = \sum_{a_i \in \phi} a_i R_i(X)$, and $O_{in} = \sum_{a_i \in \phi} a_i O_i(X)$. Bob accepts Alice's proof if:

$$\begin{aligned} e(A \cdot g^{L_{in}(r)}, B \cdot g^{R_{in}(r)}) &\stackrel{?}{=} e(D, g) \cdot e(C \cdot g^{O_{in}(r)}, g) \\ E^{D(r)} &\stackrel{?}{=} D. \end{aligned}$$

  If the proof is valid Bob now learned that with high probability Alice did indeed compute a_6 by using a_1, a_2, a_3 as inputs to the agreed upon function foo[2].

Note that we needed the bilinear map e even though this is a designated verifier setting. But taking a look at equation 4.7 and Alice's proof elements A, B we notice that we can only reconstruct $\sum_{i=0}^{m} a_i L_i(X) \cdot \sum_{i=0}^{m} a_i R_i(X)$ in the exponent of the pairings generator $e(g, g)$. This is a very useful insight as we learn from it why we are currently limited to **Quadratic** Arithmetic Programs and cannot have a circuit language that supports multiplication gates with more then 2 inputs. If we had a trilinear-map however, we could allow gates with three multiplicands. A k-linear map enables k-multiplicands gates.

If we wanted to consider a different kind of proof, for example: Bob only has a_1 and wants Alice to convince him that she picked some a_2,a_3 on her own and with them performed the computation correctly, they can do so by simply agreeing on, which parameters are part of the statement ϕ and which once are part of the witness w. This is an important insight and makes SNARKs incredibly flexible regarding their field of application. In general, if the relation is defined with $w \in \mathbb{F}^m$, there are

$$\sum_{i=0}^{m} \binom{m}{i} = 2^m,$$

possible combinations of witness and statement relations upon which a zkSNARK of this kind can be build. The case where there is no statement, basically is a just proof that Alice did the computation successfully, but besides that Bob learns nothing more. This can be understood as *Proof of Work* likewise.

[2] We neglected the knowledge of coefficients check for simplicity

4.3 R1CS Optimizations

In this section we explain some optimizations we came up with in order to reduce the R1CS size in our implementation[Wol]. As almost all SNARK and SNARG constructions provide their computational space and time complexities in direct dependence of the number of multiplication gates appearing in the underlying arithmetic circuit, we want to challenge this common sense by pointing out that optimization in the R1CS representation can provide considerable improvements as well. In our implementation we provide an algorithm for efficient translation from arithmetic circuit into R1CS, where we consider the following:

- Trivial: reuse a gates output whenever possible. The input factors are hashed and a hashtable lookup reveals if the gate has already been translated into a constraint.
- Exploit the commutative property of a gate. If at some point the computation of $a = x * y$ is needed, and later we write $b = y * x$ somewhere, we omit creating a new constraint from the gate b and take $b = a$. So b is now 'pointing' to a. If the factors x, y are the summation of some values, we order the therms in dictionary manner. For example $x = (7m + 3n + 3k) * (4 + l) \rightarrow x = (l + 4) * (3k + 3n + 7m)$. The ordering technique is irrelevant as long as it is done consistently throughout compiling.
- Extract constant factors: say we have $a = 7 * x * y$, we create the R1C for $a' = x * y$ and memorize the associated 7. If we use a in a later R1C we use $a * 7$. So if we need $b = 8 * x * y$ we set $b = a'$ and memorize the associated factor $7 * 8$ and save one gate. If the circuit ends say with $c = b * a$, we create the constraint $c * (7 * 8)^{-1} * 7^{-1} = a' * a'$. So an example in our domains specific go-like language

```
func main(x) {
  var a = 2*(x*x)
  var b = 3*(a*a)
  return b
}
```

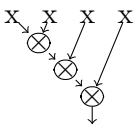
gives the R1CS:

$$\left\{ \left(\underbrace{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}_{L}, \underbrace{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}_{R}, \underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{O} \right), \left(\underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{L}, \underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{R}, \underbrace{\begin{bmatrix}0\\0\\0\\12^{-1}\end{bmatrix}}_{O} \right) \right\}, \underbrace{\begin{bmatrix}1\\x\\a\\b\end{bmatrix}}_{\mathbf{a}}$$

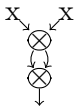
Since a inverse of a small number gives a huge number in the underlying field, we sometimes observe this big numbers in the R1CS, which is surprising if one does not know this optimization, since most of the time a R1C contains 0's and 1's. This

suggests that R1CS standards should support two optional flags next to each entry that indicate if a number is negative and/or inverse. The vectors are usually stored in sparse representation as most entries are 0. Also it is a necessity to add the field order in the R1CS description, since the same program compiled on different fields can lead to different R1CS. For example, the condition $3 < 2$ is true in $\mathbb{F}_3$

- Tree balancing: Say we want $a = ((x * x) * x) * x$. Naively implemented we get:



if we use an AVL tree where an $-, +$ gate is treated as leafes, and only balance the multiplications and divisions, we get $(x * x) * (x * x)$ end therefore can reuse $(x * x)$:



In the extreme case, this turns k gates to $log(k)$ gates. This optimization is a bit tricky and does not always optimize. For example we compute x^4 but need x^3 later again, no gates are saved as we need to create $x^3 = (x * x) * x$ reusing $(x * x)$.

- Compute arguments just-in-time and not in advance. This point is a bit tricky to explain but an example might help. Consider the following function in our language

```
func main(x,y,z) {
  var mul = func(a,b){
   return a*b
  }
  mul(x+y,z)
  return
}
```

were we declare a function *mul* inside the scope of main and then call it. If calling *mul* requires to compute $x + y$ before it is fed into the function, the R1CS therefore would need two elements. First the addition $x + y = a_1$, then $a_1 * z = a_2$. Our language is supports functions as first class types. Therefore $x + y$ is a function and we pass it as such as an argument to *mul*. This way we get $(x + y) * z = a_1$ which is expressible with one R1-constraint instead.

- Optimize addition by default can lead to undesired effects. As we introduced QAPs and R1CS, we mentioned that addition gates do not affect the size of the R1CS as they can be packed into its vectors. But what if we want to convince the verifier that the prover computed the Fibonacci series up to the n-th number for example?

```
func main(){
 return Fibonacci(4)
}
func Fibonacci(a){
 if a==0{
  return 1
 }
 if a==1{
  return 1
 }
 return Fibonacci(a-1)+Fibonacci(a-2)
}
```

Since its a program only consisting of addition, the R1CS would be empty. Therefor we introduced the predefined function *addGateConstraint*$(\cdot,\cdot)$, which needs to be called in case the addition is explicitly wanted to be part of the R1CS. Replacing the last line with :

return addGateConstraint(Fibonacci(a-1),Fibonacci(a-2)),

yields the desired R1CS:

$$\left\{\left(\begin{bmatrix}2\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}\right)\left(\begin{bmatrix}1\\1\\0\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}\right)\left(\begin{bmatrix}0\\1\\1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\right)\left(\begin{bmatrix}0\\0\\1\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}\right)\right\},$$

where one can easily verify that with $a_0 = 1$ the witness becomes $\mathbf{a}=(1,1,2,3,5)^{\mathrm{T}}$, which is precisely the Fibonacci series up to the fifth element. Note that the runtime of the above code is exponential, yet the R1CS is linear in size. This is a consequence of the applied trivial gate-reuse-rule. In our implementation we provide code examples computing Fibonacci series using dynamical programming as well.

4.4 Investigating Bilinear Pairing

Pairing-based cryptography revolves around a particular function with some exciting properties, the so called 'pairing function'. Loosely speaking, they add a new property to cyclic groups, which initially was seen as a new method to break some cryptosystems[FMR99, FR94]. From then it took some time till their constructive use was discovered [Jou02]. Nowadays their applications range from identity-based Encryption, very short signature schemes, non-interactive key exchange, attribute based cryptography up until public key encryption with key-word search. Since they are the essential ingredient for the construction of the SNARK type we investigate as well, we now scratch the surface of their underlying mathematics. The dissertation of Ben Lynn 'on the implementation of pairing-based cryptosystem'[Lyn07] forms the foundation of this chapter and is highly recommendable for everyone who wants to gain a deeper understanding.

We first state the most essential difficulty assumptions in public key cryptosystems. Let $G = \langle g \rangle$ of prime order r, where g is a generator of G and let x, y, z be integers in $[0,r-1]$.

- The **discrete log problem DLP** describes the challenge of computing x given g and g^x.
- The **Computational Diffie-Hellman Problem CDH** says: given g, g^x, g^y, compute g^{xy}.
- The **Decisional Diffie-Hellman Problem DDH**: given g, g^x, g^y, g^z, determine if $xy = z$.

From these three problems, it is immediately immanent that the discrete log problem is the most important one. Solving it, solves the DDH and CDH as well. An elliptic curve E for cryptographic applications is usually defined over a finite field $\mathbb{F}_p$. They form cyclic Abelian groups where the elements are the points $\{(x, y)\}$ that satisfies the short Weierstrass equation $y^2 = x^3 + bx + c$. b and c are curve parameters with the condition $4a^3 + 27b^2 = \Delta$. If $\Delta = 0$, we speak of a singular elliptic curve. If $\Delta \neq 0$ then it is nonsingular, i.e. the roots of the cubic are distinct. The group operation comes from the chord-tangent law. Since the set of points E_s of singular curves over a field K allow efficient isomorphic mappings to $E_s \cong K^*$ or $E_s \cong K^+$, they are useless for cryptographic applications since the discrete log is easy in K^+. This is an interesting insight since cryptography in finite field therefore can be seen as a special case of elliptic curve cryptography. The Montgomery and Edwards equation are reducible to the Weierstrass form and currently not important for the description of the pairing functionality.

The simplest abstract definition of a symmetric bilinear pairing (or bilinear map) e is a function that efficiently maps two elements of G onto G_T, which are cyclic groups of prime order with the generator g of G s.t.:

$$e : G \times G \to G_T,$$

and it holds that:

1. Non-degeneracy $e(g, g) \neq 1$
2. Bilinearity $e(g^a, g^b) = e(g, g)^{ab}$ for each $a, b \in \mathbb{Z}$
3. Admissible: We call a pairing admissible if it is efficiently computable.

Currently there are only two known techniques to produce admissible parings for cryptographic purposes, namely the Weil and the Tate paring. As a remark, proving bi-linearity in these constructions is fairly simple while proving non-degeneracy is the challenging part. Two immediate observations: we can only perform at most one multiplication in the exponent. We then land in the target group which can not be fed back into the pairing. Second: With such pairing the DDH problem immediately becomes solvable, as on only needs to apply the pairing twice and check if $e(g^x, g^y) == e(g, g^z)$. The counterparts in pairing-based cryptography to classical problems are:

- **Bilinear Diffie-Hellman Problem**. g, g^a, g^b, g^c compute $e(g, g)^{abc}$ [BF01].
- **Decisional Bilinear Diffie-Hellman Problem**. Given $g, g^a, g^b, g^c, e(g, g)^w$, determine if $w = abc$[BB04a].
- **q-power knowledge of exponent assumption(q-PKE)** Given $\{g^{r^i}\}_{i=0}^q$ and $\{g^{(\alpha r)^i}\}_{i=0}^q$, find A, B where $A \neq g^{\sum_{i=0}^q a_i s^i}$ for any choice of a_i, s.t. $e(A, g^\alpha) = e(g, B)$ [DGM14].
- **q-Strong Diffie-Hellman Problem (q-SDH).** Given $g, g^s, g^{s^2}, .., g^{s^q}$ with $s \in \mathbb{Z}_r^*$, compute $y \in G_T$ s.t. $y = e(g, g)^{\frac{1}{s+c}}$ for any $c \in \mathbb{Z}_r^*$ [BB04b].

In addition to these assumptions, we are also going to need the following assumption for our verifiable computation construction from Section 5, which is

- **q-power Diffie-hellman assumpition**. Given $g, g^s, g^{s^2}, .., g^{s^q}, , g^{s^{q+2}}, .., g^{s^{2q}}$ with $s \in \mathbb{Z}_r^*$, find $y =, g^{s^{q+1}}$ [PHGR13].

Also as it will turn out, that the target group G_T is a multiplicative cyclic subgroups of prime order of a finite field rather then an group over an elliptic curve and therefore pairings become a double-edged sword, since they allow to transfer a DH problem on an elliptic curve into solving a DH problem on a finite field, for which we know sub-exponential runtime techniques such as index calculus. Given g^x, x can be recovered with

$$Dlog(e(g^x, g), e(g, g)) = Dlog(e(g, g)^x, e(g, g)) = x.$$

Whereas for breaking general elliptic curves the most efficient techniques are Pollard rho and lambda methods, which are comparatively slower then index calculus. As mathematicians noticed this transfer technique the community first went through a shock-wave which luckily turned out to be less fatal as some thought at first glance. It did not break ECC security, but rather added a new criterion which curve implementers now need to consider such that this mapping cannot be abused to endanger their curves security. Curves used for pairing based cryptography try to find an optimal ratio between the element sizes in the target group and computational effort to break DLog in the target group considering the most efficient algorithms. They are also supersingular and can have complex multiplication for which currently no specific attacks are known.

The definition of pairings can be a bit loosened to support a wider range of curves in addition to the supersingular ones we described for symmetric pairings. These pairing types are referred to as asymmetric pairing[BLS01] if the signature is

$$e : G_1 \times G_2 \to G_T,$$

where G_1, G_2, G_T are cyclic groups. Galbraith, Paterson and Smart [GPS08] introduced the following distinction among bilinear groups: Type I if $G_1 = G_2$, Type II there is an efficiently computable non-trivial group isomorphism $\phi : G_2 \to G_1$ and Type III where such a isomorphism as in Type II does not exist in either direction. In practice Type III bilinear groups are the most efficient once. In our implementation [Wol] we use the Type III Optimal Ate pairing over a 256-bit Barreto-Naehrig curve as described in [NNS10] [3]. We again denote g as generator of G_1 and h as generator of G_2. The non-degeneracy, bilinearity and admissibility are required to hold for asymmetric pairings as for symmetric pairings. An intriguing question certainly is, if the target group could be a source group, e.g. if a mapping $e : G \times G \to G$ is feasible. This kind of mapping could also be constructed if there was an efficiently computable isomorphism from $\phi : \mathbb{G}_T \to \mathbb{G}_1$, or both $\phi_1 : \mathbb{G}_T \to \mathbb{G}_2$ and $\phi_1 : \mathbb{G}_2 \to \mathbb{G}_1$. The answer is currently unknown and a question to ongoing research[YYHK17, Bon]. Such a map would immediately break CDH and DDH, but if DLP would still be hard then we could derive fully homomorphic encryption schemes from it.

[3] Deprecated: due to its weakened security (<128 bit security), new systems should not rely on this elliptic curve. The package we use is frozen, and not implemented in constant time.

4.5 Classic Preprocessed Pairing-Based zkSNARK

Now that we explained how to go from code to quadratic arithmetic programs, we can have a look at Jens Groth's protocol[Gro16] which is currently the most efficient type of preprocessed pairing based zkSNARKs. We take the Relation R that defines a field $\mathbb{Z}_p$, a language of statements $\phi = (a_1, ..., a_l) \in \mathbb{Z}_p^l$ and witness data $w = (a_{l+1}, ..., a_m) \in \mathbb{Z}_p^{m-l}$ with the relation

$$\sum_{i=0}^{m} a_i L_i(X) \cdot \sum_{i=0}^{m} a_i R_i(X) - \sum_{i=0}^{m} a_i O_i(X) = P(X) = Q(X)D(X). \tag{4.8}$$

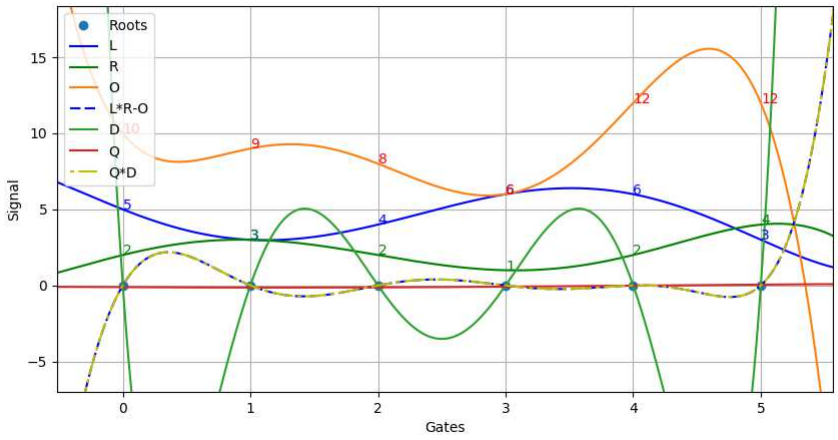


Figure 4.5: Shows an example how the polynomials in equation 4.8 relate to each other, where $L = \sum a_i L_i(X)$,$R = \sum a_i R_i(X)$, and $O = \sum a_i O_i(X)$. Note that in practice the polynomials are elements of $\mathbb{Z}_p[x]$

As in the chapter before, $L_i(X)$ is a polynomial that gives the signal strength of i-th wire in the circuit as 'left-side' input of a multiplication gate X. $R_i(X)$ does the same but only for each gates 'right-side' input and $O_i(X)$ for the output. Note that the left-hand side of the equation is 0 at each gate index r_i as it states: the sum of all left inputs times the sum over all right inputs is equal to the sum over all outputs at each gate. Hence we know that $D(X) = \prod_{i=0}^{n}(X - r_i)$ can be extracted from $P(X)$ and the corresponding polynomial $Q(X)$ exists. The asymmetric bilinear mapping we soon are going to need has the signature $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the property $e(G^x, H^y) = e(G, H)^{xy}$. More details on pairings are stated in chapter 4.4. We pick G as $\mathbb{G}_1$ and H as $\mathbb{G}_2$ generators. Groth gives the following algorithms for NIZKs:

- $(\sigma, \tau) \leftarrow \mathsf{Setup}(R, 1^\lambda)$: Pick $\alpha, \beta, \gamma, \delta, x \xleftarrow{\$} \mathbb{Z}_p^*$.

$$\sigma = \left(\left\{ G^{x^i} \right\}_{i=0}^{n-1}, \left\{ G^{\frac{\beta L_i(x)+\alpha R_i(x)+O_i(x)}{\gamma}} \right\}_{i=0}^{l}, \left\{ G^{\frac{\beta L_i(x)+\alpha R_i(x)+O_i(x)}{\delta}} \right\}_{i=l+1}^{m} \right.$$
$$\left. G^{\alpha}, G^{\beta}, G^{\delta}, H^{\beta}, H^{\gamma}, H^{\delta}, \left\{ H^{x^i} \right\}_{i=0}^{n-1}, \left\{ G^{\frac{x^i D(x)}{\delta}} \right\}_{i=0}^{n-1} \right)$$

Define $\tau = (\alpha, \beta, \gamma, \delta, x)$. This tuple is often referred to as 'toxic waste'. Proofs without knowledge of a witness trace satisfying the relation 4.8 can be easily forged given this τ.

- $\pi = (A, B, C) \leftarrow \mathsf{Prove}(R, \sigma, \phi, w)$: Choose $r, s \xleftarrow{\$} \mathbb{Z}_p$. Compute

$$\begin{aligned} A &= G^{\alpha+r\delta+\sum_{i=0}^{m} a_i L_i(x)} \\ B &= H^{\beta+s\delta+\sum_{i=0}^{m} a_i R_i(x)} \\ C &= G^{\frac{\sum_{i=l+1}^{m} a_i(\beta L_i(x)+\alpha R_i(x)+O_i(x))+Q(x)D(x)}{\delta} + s\left(\alpha+\sum_{i=0}^{m} a_i L_i(x)\right) + r\left(\beta+\sum_{i=0}^{m} a_i R_i(x)\right) + rs\delta} \end{aligned}$$

  We note that $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. Using asymmetric pairings, operations in $\mathbb{G}_2$ pose significantly more computational effort and the elements are twice as big as those in $\mathbb{G}_1$. Therefor a parser that translates code to QAPs is advised to keep the polynomials R_i as sparse as possible, and it explains some of the design choices in Groth's protocol.

- 0,1$\leftarrow \mathsf{Vfy}(R, \sigma_V, \phi, \pi = (A, B, C))$ accept iff

$$e(A, B) \stackrel{?}{=} e(G^{\alpha}, H^{\beta}) \cdot e\left(G^{\frac{\sum_{i=0}^{m} a_i(\beta L_i(x)+\alpha R_i(x)+O_i(x))}{\delta}}, H^{\gamma} \right) \cdot e(C, H^{\delta}) \qquad (4.9)$$

  The verification requires 3 pairing operations, since $e(G^{\alpha}, H^{\beta})$ can be precomputed. Furthermore verification requires l exponentiations e.g. proving time is quasi linear in the statement size. Also it is important to notice that the verifier does not need to store the entire σ, but rather $\sigma_v = \left(\left\{ G^{\frac{\beta L_i(x)+\alpha R_i(x)+O_i(x)}{\gamma}} \right\}_{i=0}^{l}, H^{\gamma}, H^{\delta}, e(G^{\alpha}, H^{\beta}) \right)$

The assignments are computed in the field $\mathbb{Z}_p$ which ideally should be of the same order as the field over which the elliptic curve is defined. Unfortunately this field matching has been proven to be impossible[EBSV17]. The protocol has perfect zero-knowledge as r and s are chosen at random and therefore A, B, C indistinguishable from uniform randomly chosen group elements. It has statistical knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations.

4.6 Proving Sudoku in Zero Knowledge

Assume Bob has the unsolved Sudoku puzzle shown in figure 4.6. He wants to test Alice's skills weather she can solve the puzzle. Alice is willing to solve it, however she does not want to reveal the solution to Bob. The question is now, how can Alice convince Bob that she knows the solution to his Sudoku instance, without giving Bob any hint on the actual solution? We now propose a solution to this problem, entirely based on arithmetic circuits.

j\i	1	2	3	4	5	6	7	8	9
1		2		5		1		9	
2	8			2		3			6
3		3			6			7	
4			1				6		
5	5	4						1	9
6			2				7		
7		9			3			8	
8	2			8		4			7
9		1		9		7		6	

We label the cells starting at the upper left corner with $a_{1,1}$, to $a_{1,9}$ in the first row. $a_{2,1}$, to $a_{2,9}$ in the second row etc. In the given puzzle $a_{1,2}$=2, $a_{1,4} = 5$ etc. A Sudoku $S = \{a_{i,j} | i, j \in \{1, .., 9\}\}$ has an instance $\phi \subseteq S$ and solution $w \subseteq S$ s.t. $\phi \bigcap w = \emptyset$, $w \cup \phi = S$. We first introduce the function $\dot{1}(\cdot):\mathbb{N}_{\geq 1} \to \mathbb{N}$ with $x \mapsto \prod_{j=1}^{x} prime(j)$ where $prime(j)$ is the j-th smallest prime number. For example $4\dot{1}:=\dot{1}(4) = 2 \cdot 3 \cdot 5 \cdot 7 = 210$. A *prime Sudoku* is basically the same then a casual Sudoku, except that instead of the numbers 1 to 9, the first 1 to 9 primes are used.

S is a valid solved instance of a *prime sudoku* if it satisfies:

1. each row must be filled with all of the first 9 primes. We formulate this constraint with
$$\prod_{i=1}^{9} prime(a_{m,i}) \stackrel{!}{=} 9\dot{1} \; \forall m \in \{0, .., 8\}. \tag{4.10}$$
Notice that due to the unique prime factorization the left-hand side can only be equal to $9\dot{1}$ if and only if the selected $a_{i,j}$ form indeed the set $\{prime(i)\}_{i=1}^{9}$. Assignments with values not from the set of the first 9 primes such as eight 1s and one $9\dot{1}$, will be prohibited with constraint 4.

2. each column must be filled with all of the first 9 primes. We enforce this const£raint similarly via
$$\prod_{i=1}^{9} prime(a_{i,m}) \stackrel{!}{=} 9\dot{1} \; \forall m \in \{0, .., 8\}. \tag{4.11}$$

3. each 3x3 box must be filled with all of the first 9 primes. We setup this constraint with

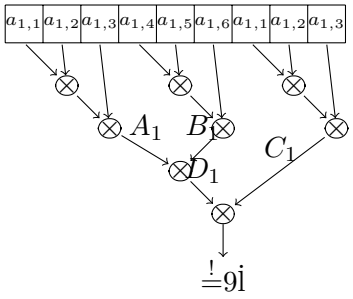
$$\prod_{i=1+3n}^{3+3n} \prod_{j=1+3m}^{3+3m} prime(a_{i,j}) \stackrel{!}{=} 9\dot{1} \; \forall m, n \in \{0, 1, 2\}. \tag{4.12}$$

4. so far the constraints do not prevent the prover from picking inverse elements. For example: a row with 2 and 5 as part of the statement, can be satisfied by simply assigning $9\dot{1} \cdot (5\dot{1} \cdot 2\dot{1})^{-1}$ to one of the unassigned fields. In order to ensure that the prover does not pick any elements besides the first nine prime numbers we add the arithmetic constraint:

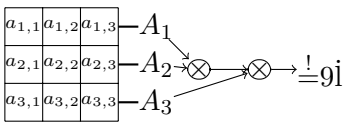
$$\prod_{i=1}^{9} (a_{m,n} - prime(i)) \stackrel{!}{=} 0 \; \forall m, n \in \{0, .., 8\}. \tag{4.13}$$

   This is a polynomial of degree 9. In the code example 4.6 we perform this check on all 81 values and therefore get 729 multiplication constraints to compute the polynomials and additionally 81 equality check constraints.

In the setup phase Bob writes the code as in 4.6, which performs all required checks on a given solved *prime Sudoku* on input. Since gate reduction is essential to reduce the proving time, the outputs of multiplication gates should be reused whenever possible in order to keep the circuit as small. If Bob realizes the first row constraint with the arithmetic circuit such as:



Bob can reuse the the gates A_i, B_i, C_i from the row constraint to construct a box constraint by simply multiplying $A_i \cdot A_{i+1} \cdot A_{i+2}$ for $i \in \{1, 4, 7\}$, which only requires 2 gates per box constraint now instead of 8. The upper left box constraint then becomes a circuit of the form:



From this we learn that considering the order in which multiplications are performed, can have a significant impact on the overall amount of gates. We generalize and exploit this fact in various ways in our implementation [Wol]. Now suppose Bob has the arithmetic circuit and reuses gates whenever possible. He performs the translation into the R1CS as described in chapter 4.3. The requirement that the product of a row, column or box has the constant output $9\dot{\text{i}}$, can be done in different ways. Either we add $9\dot{\text{i}}$ to the statement or we embed it in the R1CS with:

$$\left\{ ..., \left(\underbrace{\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \end{bmatrix}}_{L}, \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix}}_{R}, \underbrace{\begin{bmatrix} 9\dot{\text{i}} \\ \vdots \\ 0 \\ \vdots \\ 0 \\ \vdots \end{bmatrix}}_{O} \right) \right\}_{R1CS}, \underbrace{\begin{bmatrix} 1 \\ \vdots \\ C_i \\ \vdots \\ D_i \\ \vdots \end{bmatrix}}_{\mathbf{a}}$$

Hard coding it into the R1CS is under most circumstances the better choice since it decreases the computational effort for both, prover and verifier. If however we wanted the flexibility of changing what the product of a cell, row and box should be after the setup was performed, embedding it in the witness $\mathbf{a}$ would make sense. Once Bob created the R1CS and translated it into the QAP relation, they can apply Groth's protocol[Gro16] as explained in section 4.5.

```
func main(x[9][9]){   #the input is a solved 9x9 prime sudoku field
  var ProductOfPrimes = 2*3*5*7*11*13*17*19*23
  var i = 0
  for (i<9;i=i+1){# we check if all inputs are among the first 1 to 9 prime
    numbers
    var j = 0
    for (j<9;j=j+1){
      #equal ist the inbuilt function to create an equality assertion
    constraint
      equal(RangeCheckPolynomial(x[i][j]),0)
    }
  }
  i = 0
  for (i<9;i=i+1){      #we check all columns
    var Product = 1
    var j = 0
    for (j<9;j=j+1){
      Product = Product * x[i][j]
    }
    equal(Product,ProductOfPrimes)
  }
  i = 0
  for (i<9;i=i+1){  #we check all rows
    var Product = 1
    var j = 0
    for (j<9;j=j+1){
      Product = Product * x[j][i]
    }
    equal(Product,ProductOfPrimes)
  }
  i = 0
  for (i<9;i=i+3){  #we check if each 3x3 box is satisfied
    var j = 0
    for (j<9;j=j+3){
      var Product = 1
      var k = 0
      for (k<3;k=k+1){
        var l = 0
        for (l<3;l=l+1){
          Product = Product * x[i+k][j+l]
        }
      }
      equal(Product,ProductOfPrimes)
    }
  }
  return
}
func RangeCheckPolynomial(x){
  return (x-2)*(x-3)*(x-5)*(x-7)*(x-11)*(x-13)*(x-17)*(x-19)*(x-23)
}
```

Figure 4.6: *Prime Sudoku* satisfiability check program written in our domain specific language in order to create a zkSNARK over it. The code compiles into a R1CS with 966 constraints and a witness length of 980 field elements.With optimal ate pairing over a 256 bit Baretto-Naehring curve, CRS creation on a intel i7-7700HQ @ 2.80Ghz×8 took 10 min, whereby computing the basis polynomials for the Lagrange interpolation took up 90%. Proof generation took 1 min and verification 50 ms. The range checks are performed on all 81 fields. This increases the CRS and proof time, however makes the scheme fully operational for arbitrary sudoku instances. In case the knowledge proof should be about one particular instance of a sudoku, the range check of the statements in the code can be omitted.

CHAPTER 5

zkSNARKs from Extended Algebraic Programs

In this section we provide a new type of NIZK argument which allows to prove knowledge of a secret key, given the corresponding public key, with only one circuit gate instead of >864 as in current state-of-the-art implementations (disregarding the additional constraints that are required in order to assure that the input is valid). It is an extension of Groth's [Gro16] QAP-based zkSNARK construction where we, roughly speaking, add the operation for 'scalar point multiplication on an elliptic curve' to the arithmetic operations '+' and '×'. Therefore, the notion of of an arithmetic circuit, R1CS and QAP had to be extended consequently to something we name R1CS* and extended algebraic program (**EAP**) instead of the commonly used QAPs.

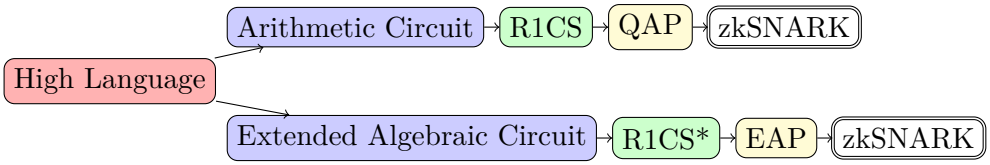


Figure 5.1: Shows the necessary steps to make a program written in a high level language (could be a subset of common c), provable with a zkSNARK. The upper path is the standard procedure. The path below our contribution.

The verifier work increases from 3 to 4 pairing function calls, 1 exponentiation in $\mathbb{G}_{\mathbb{T}}$, 1 in $\mathbb{G}_1$ and 2 in $\mathbb{G}_2$, as compared to Groth's work. Before this extension, proving knowledge of the discrete logarithm of an elliptic curve point (e.g. I prove I know x s.t. $pk = g^x$ without revealing x) had to be done by expressing the point multiplication as arithmetic circuit. This required at least 864 gates for each point multiplication within a circuit, and required sophisticated techniques. For n point multiplications it took $N{\cdot}864$

arithmetic gates, which increases the common reference string (CRS) by at least $N \cdot 3 \cdot 864$ elements. Here it takes N gates and therefore adds $3N$ elements to the CRS. The prove time improves drastically as it is not necessary any more to compute the coefficients of the polynomials that were needed for proving knowledge of a valid arithmetic circuit assignment that expressed a scalar point multiplication. Consequently the prover also does not need to perform the corresponding blind evaluations, which require a point multiplication for each of the $N \cdot 3 \cdot 864$ constraints. The polynomial division effort decrease likewise as the degree is reduced by $\approx N{\cdot}864$. Before this scheme, it was almost infeasible to pack multiple point multiplications into one SNARK. Performing a shielded transaction in zCash using our extension can be performed in a few milliseconds instead of seconds. To make the proof elements indistinguishable from uniform random and therefore zero-knowledge, only one additional gate (see section 5.1.1) is required.

Our NIZK arguments for EAPs considers a circuit consisting of addition, multiplication and elliptic curve point multiplication gates. The computations are performed over a finite field $\mathbb{F}$. Figure 5.2 shows how a EAP enforced R1CS* constraint for a simple statement looks like. In the following lines we write $\langle \mathbf{A}, \mathbf{B} \rangle_{[N_i,N_j]} := \sum_{i=N_i}^{N_j} A_i B_i$, and $\langle \mathbf{A}, \mathbf{B} \rangle_{[N_j]} := \langle \mathbf{A}, \mathbf{B} \rangle_{[0,N_j]}$ for shortness. Also we use multiplicative notation for the elliptic-curve group operation.

An efficient-prover publicly verifiable non interactive argument is a quadruple of probabilistic polynomial-time algorithms

- $(\sigma_P, \sigma_V, \tau) \leftarrow \mathsf{Setup}(R, 1^\lambda)$: where R is a polynomial-time decidable binary relation with elements $(\phi, w) \in R$ where ϕ is a statement and w the witness. λ is the security parameter.
  Setup returns two common reference strings: σ_P for the prover and σ_V for the verifier. The non-interactive argument is called publicly verifiable since σ_V can be deduced from σ_P. Otherwise it is called a designated-verifier argument, which are of minor importance for distributed ledgers and therefore not treated in this work. τ we call a simulation trapdoor. It allows the creation of 'fake'-proofs. Usually it expected to be deleted right after the setup has been performed.
- $\pi \leftarrow \mathsf{Prove}(R, \sigma_P, \phi, w)$: Takes the relation R such as 'I know w s.t. $w^2 + w = \phi$', where w is called the witness, and ϕ the statement. It outputs an argument π on the knowledge of w.
- $0{,}1 \leftarrow \mathsf{Vfy}(R, \sigma_V, \phi, \pi)$: Takes a relation, statement and an argument as input and outputs '0' (reject) if the argument is not convincing or '1' (accept) otherwise.
- $\pi \leftarrow \mathsf{Sim}(R, \tau, \phi)$ creates arguments for a statement ϕ without a witness, but using the trapdoor instead.

The description of a EAP is

$$EAP := (\mathbb{F}, e(\cdot,\cdot), \mathcal{G}_M^{\mathcal{A}}(\cdot), l, \{L_i(X), R_i(X), E_i(X), O_i(X)\}_{i=0}^m, D(X)),$$

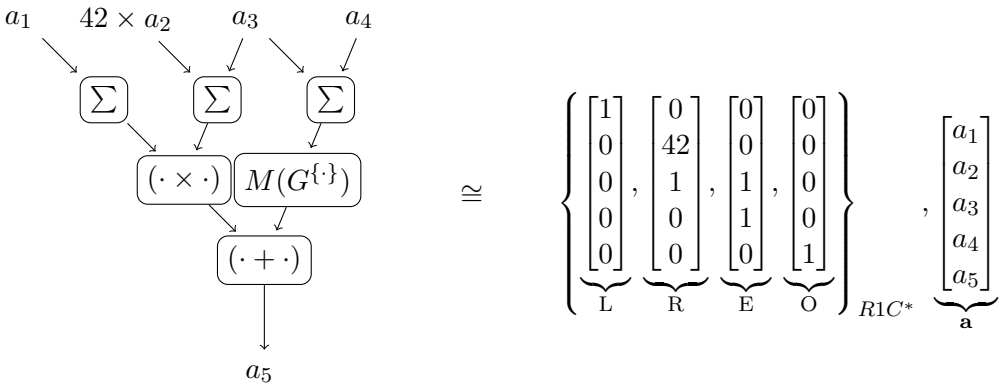


Figure 5.2: Shows how the example statement $a_1 \times (42 \times a_2 + a_3) + M\left(G^{a_3+a_4}\right) = a_5$ is realizable with one EAP gate and hence can be expressed as a single extended Rank 1 Constraint (R1C*). The prover will convince the verifier that it has knowledge of a witness $\mathbf{a}$ s.t. it satisfies: $\mathcal{G}_M^{\mathcal{A}}\left(\langle E, \mathbf{a}\rangle\right) + \langle L, \mathbf{a}\rangle \cdot \langle R, \mathbf{a}\rangle = \langle O, \mathbf{a}\rangle$.

where

- $\mathbb{F}$ is a finite field.
- e is a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as described in section 4.4. G as $\mathbb{G}_1$ and H as $\mathbb{G}_2$ generators.
- n is the number of multiplication gates in a circuit
- m is the number of wires of the circuit
- $\mathcal{A} \subset \mathbb{N}_{\geq 0}$ defines the set of gate indexes. We enumerate the gates from 0 to $n-1$ e.g. $\mathcal{A} = \{0, .., n-1\}$
- $L_i(X), R_i(X), E_i(X), O_i(X)$ are polynomials of degree $n-1$ that express the scale factor for the i-th wire at gate X. For example: $L_3(42) = 1$ states that the 3rd wire is the 'left' multiplication input of gate with index 42. $R_7(42) = 2$ implies that the 7th wire is 'right' multiplication input of gate with index 42 and that this wires assigned signal will be scaled by the linear factor of 2. $E_7(42) = 7$ implies that the 7th wire is used as an input to the gate with index 42 and whatever will be assigned to this wire, it will be amplified by a factor of 7. We write $\mathbf{L}$ for the vector of polynomials $(L_0(X), L_1(X), .., L_m(X))$ (same for **R,E,O**).
- Each EAP-gate has a uniquely assigned index $r_i \in \mathcal{A}$, which compose the domain polynomial $D(X) = \prod_{r_g \in \mathcal{A}}(X - r_g)$. The degree $deg(D(X)) = n = |\mathcal{A}|$ consequently.
- A crucial role plays the projection function $M(\cdot) : \mathbb{G}_1 \mapsto \mathbb{F}$. Its best possible design currently exceed our understanding. Its most trivial realization would be taking x

coordinate of the curve point. However: This would reduce its bit security by one bit since for each curve point (X,Y) (except the roots) exists (X,-Y).

- $\mathcal{G}_M^{\mathcal{A}}(P(X)) \to \widetilde{P}(X)$, where $\mathcal{A}$ is the set of gate indexes and $|\mathcal{A}| = k+1$ takes as input a polynomial of degree k and returns a polynomial of degree k s.t. $\forall x \in \mathcal{A}$: $\widetilde{P}(x) = M\left(G^{P(x)}\right)$.

A EAP is defined by the binary relation

$$R = \left\{ (\phi\ ,\mathrm{w}) \;\middle|\; \begin{array}{l} \phi = (a_1, ..., a_l) \in \mathbb{F}^l \\ w = (a_{l+1}, ..., a_m) \in \mathbb{F}^{m-1} \\ \mathcal{G}_M^{\mathcal{A}}\left(\langle \mathbf{a}, \mathbf{E}\rangle\right) + \langle \mathbf{a}, \mathbf{L}\rangle \cdot \langle \mathbf{a}, \mathbf{R}\rangle \equiv \langle \mathbf{a}, \mathbf{O}\rangle \bmod D(X) \end{array} \right\} \tag{5.1}$$

for some degree $n-2$ quotient polynomial $Q(X)$. This gives us the NIZK argument:

- $(\sigma, \tau) \leftarrow \mathsf{Setup}(R, 1^\lambda)$: Pick $\alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{Z}_p^*, x \xleftarrow{\$} \mathcal{A}$. Define $\tau = (\alpha, \beta, \gamma, \delta, x)$. The common reference string is:

$$\sigma = \left(\begin{array}{c} \left\{ G^{\frac{\alpha R_i(x) + \beta(L_i(x) + E_i(x)) + O_i(x)}{\gamma}} \right\}_{i=0}^{l}, \left\{ G^{\frac{\alpha R_i(x) + \beta(L_i(x) + E_i(x)) + O_i(x)}{\delta}} \right\}_{i=l+1}^{m}, \\ G^{\alpha}, G^{\beta}, G^{\delta}, H^{\beta}, H^{\gamma}, H^{\delta}, \left\{ G^{\frac{x^i D(x)}{\delta}} \right\}_{i=0}^{n-1}, \left\{ G^{x^i} \right\}_{i=0}^{n-1}, \left\{ H^{x^i} \right\}_{i=0}^{n-1} \end{array} \right)$$

- $\pi = (A, B, C, F) \leftarrow \mathsf{Prove}\left(EAP, \sigma, \phi, w\right)$: Compute

$$A = G^{\alpha + \langle \mathbf{a}, \mathbf{L}(x)\rangle_{[m]} + \langle \mathbf{a}, \mathbf{E}(x)\rangle_{[m]}} \tag{5.2}$$

$$B = H^{\langle \mathbf{a}, \mathbf{R}(x)\rangle_{[m]}} \tag{5.3}$$

$$C = G^{\frac{\langle \mathbf{a}, \beta\mathbf{L}(x) + \alpha\mathbf{R}(x) + \beta\mathbf{E}(x) + \mathbf{O}(x)\rangle_{[l+1,m]} + Q(x)D(x)}{\delta}} \tag{5.4}$$

$$F = G^{\langle \mathbf{a}, \mathbf{E}(x)\rangle_{[m]}} \tag{5.5}$$

- $0/1 \leftarrow \mathsf{Vfy}\left(EAP, \sigma, \phi, \pi\right)$ accept iff:

$$e\left(A, B \cdot H^{\beta}\right) \cdot e\left(G, H\right)^{M(F)} \stackrel{?}{=}$$
$$e\left(G^{\alpha}, H^{\beta}\right) \cdot e\left(G^{\frac{\langle \mathbf{a}, \alpha\mathbf{R}(x) + \beta(\mathbf{L}(x) + \mathbf{E}(x)) + \mathbf{O}(x)\rangle_{[l]}}{\gamma}}, H^{\gamma}\right) \cdot e\left(C, H^{\delta}\right) \cdot e\left(F, B\right)$$

Regarding efficiency we observe that the one-time setup σ runs in time linear to the circuits size $\mathcal{O}(|C|)$. The prover must perform $\mathcal{O}(|C|)$ cryptographic work and $\mathcal{O}(|C| log^2(|C|))$ to compute $Q(x)$. The polynomial representation $L_i(X), R_i(X)$ etc. is not needed for proving and the prover rather works with the vectors $l_i = (L_i(1), L_i(2), .., L_i(n)), r_i = (R_i(1), R_i(2), .., R_i(n))$ where most elements are 0. This sparsity of the evaluation vectors

is then exploited. The $\mathcal{O}(|C|log^2(|C|))$ runtime is achieved by using FFT techniques and binary tree based polynomial interpolation algorithms as it is done in the Pinocchio-Protocol[PHGR13] instead of naive Lagrange interpolation and polynomial division which would take $\mathcal{O}(n^2)$.

Note that $e\left(G,H\right)$ and $e\left(G^{\alpha},H^{\beta}\right)$ can be precomputed. Furthermore verification requires l exponentiations e.g. proofing time is quasi linear in the statement size. Constructing the pairing-friendly elliptic curves such that $\mathbb{G}_1$ representations of group elements are smaller, C is assigned to the first group for efficiency. For the same reason the verifier includes the statement ϕ into the verification process over the group $\mathbb{G}_1$.

We proof perfect completeness by direct verification:

$$
\begin{aligned}
e\left(A,B\cdot H^{\beta}\right)\cdot e\left(G,H\right)^{M(F)} &= \\
e\left(A,B\cdot H^{\beta}\right)\cdot e\left(G,H\right)^{\mathcal{G}_M^{\mathcal{A}}(\langle\mathbf{a},\mathbf{E}\rangle)} &= \\
e\left(G,H\right)^{\left(\alpha+\langle\mathbf{a},\mathbf{L}\rangle_{[m]}+\langle\mathbf{a},\mathbf{E}\rangle_{[m]}\right)\left(\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+\beta\right)+M(F)} &= \\
e\left(G,H\right)^{\alpha\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+\langle\mathbf{a},\mathbf{L}\rangle_{[m]}\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+M(F)+\langle\mathbf{a},\mathbf{E}\rangle_{[m]}\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+\alpha\beta+\beta\langle\mathbf{a},\mathbf{E}+\mathbf{L}\rangle_{[m]}} &= \\
e\left(G,H\right)^{\alpha\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+\langle\mathbf{a},\mathbf{O}\rangle_{[m]}+QD+\langle\mathbf{a},\mathbf{E}\rangle_{[m]}\langle\mathbf{a},\mathbf{R}\rangle_{[m]}+\alpha\beta+\beta\langle\mathbf{a},\mathbf{E}+\mathbf{L}\rangle_{[m]}} &= \\
e\left(G^{\alpha},H^{\beta}\right)\cdot e\left(G,H\right)^{\langle\mathbf{a},\alpha\mathbf{R}+\beta\mathbf{E}+\beta\mathbf{L}+\mathbf{O}\rangle_{[m]}+QD+\langle\mathbf{a},\mathbf{E}\rangle_{[m]}\langle\mathbf{a},\mathbf{R}\rangle_{[m]}} &= \\
e\left(G^{\alpha},H^{\beta}\right)\cdot e\left(G,H\right)^{\langle\mathbf{a},\alpha\mathbf{R}+\beta\mathbf{E}+\beta\mathbf{L}+\mathbf{O}\rangle_{[m]}+QD}\cdot e\left(F,B\right) &= \\
e\left(G^{\alpha},H^{\beta}\right)\cdot e\left(G^{\frac{\langle\mathbf{a},\alpha\mathbf{R}+\beta\mathbf{E}+\beta\mathbf{L}+\mathbf{O}\rangle_{[l]}}{\gamma}},H^{\gamma}\right)\cdot e\left(C,H^{\delta}\right)\cdot e\left(F,B\right) & \quad \square
\end{aligned}
$$

Note that the first transformation $M(F(x))=M\left(G^{\langle\mathbf{a},\mathbf{E}(x)\rangle_{[m]}}\right)=\mathcal{G}_M^{\mathcal{A}}\left(\langle\mathbf{a},\mathbf{E}\rangle\right)(x)$ can only be applied because $x\in\mathcal{A}$. For that reason this scheme is not statistical knowledge sound. If the E_i polynomials are all 0 however, e.g. we don not use the point multiplication in in the entire circuit, then we can safely pick $x\xleftarrow{\$}\mathbb{Z}_p^*$ from the entire group and end up with a scheme similar to the original[Gro16] which is statistically knowledge sound again.

We try to argue why it might be sound (disregarding the issue with the small space from which x can be chosen). Lets call the exponents of the proof elements a,b,c,f where $A=G^a,B=H^b,C=G^c,F=G^f$. We also define $\zeta=\frac{\langle\mathbf{a},\alpha\mathbf{R}(x)+\beta\mathbf{E}(x)+\beta\mathbf{L}(x)+\mathbf{O}(x)\rangle_{[l]}}{\gamma}$ as the input that will end up in the exponent a verifier will produce given the statement $\phi=(a_1,..,a_l)$. The equation for which an adversary effectively has to find a satisfying assignment is

$$a(b+\beta)+M(F)=\alpha\cdot\beta+\zeta\cdot\gamma+c\cdot\delta+f\cdot b. \tag{5.6}$$

We now make the following observation:

- a,b,c,f are under full control of the adversary i.e. they can be forged at will.

- ζ is depended on $\phi = (a_1, .., a_l)$ and the identity $a_0 = 1$. An PPT adversary cannot forge In at will. However ζ can be set to 0 if $a_1 = a_2 = .. = a_l = 0 \wedge L_0 = R_0 = E_0 = O_0 = 0 \implies \zeta = 0$. If the identity a_0 is used in the circuit e.g. at least one of the wire polynomials L_0, R_0, E_0, O_0 is unequal to 0 then $\zeta \neq 0$. Notice that the unlikely possibility exists, where there is $k \in \{1, .., l\}$ s.t. $L_k = L_0, R_k = R_0, E_k = E_0, O_0 = O_k$. Then setting $a_k = |\mathbb{F}|$ pushes $\zeta = 0$ again.
- $M(F)$ is defined as $M(G^f)$. DLP on $\mathbb{G}_1$ is assumed to be hard, therefore a PPT adversary cannot forge $M(F)$ at will.
- $\alpha, \beta, \gamma, \delta$ are unknown to the attacker, however G^α etc. are known and part of the CRS. Therefore the adversary can set $a, c, f \in \{\alpha, \beta, \delta\}$ and $b \in \{\beta, \gamma, \delta\}$, without knowing $\{\alpha, \beta, \gamma, \delta\}$ explicitly.
- If there was an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, the adversary can set a, c, f to γ in addition to above named choices.

We now try to reduce equation 5.6 to the seemingly easiest satisfiable form by setting $A = G^\alpha$,$b = c = 0$. Then it remains for an attacker to find an assignment for:

$$M(F) = \zeta \cdot \gamma. \tag{5.7}$$

In case M is the trivial function that takes a curve point and returns its X-coordinate and $G^0 = (0, 1)$, an adversary can create a satisfying assignment by setting $\phi = (0, .., 0)$. Consequently a verifier needs to check the proof elements or the statement, and reject if among them is one of the trivial assignments. If the trivial cases are rejected, finding a valid assignment therefore always ends up facing the DLP, which is assumed to be hard.

We want to point out that Groth's protocol [Gro16] upon we build this work, has the same trivial satisfying assignment. Using the same tricks and argumentation leads to:

$$0 = \zeta \cdot \gamma, \tag{5.8}$$

which is satisfiable if $\phi = (0, .., 0), A = G^\alpha, B = H^\beta, C = G^0$. In both schemes the trivial assignment is not an immanent thread however since $a_0 = 1$ and it is very unlikely that verifier does not perform any sanity checks on the statement and that the identity element is never used in any real world application circuit.

5.1 Special EAP Gates

5.1.1 Perfect Zero-Knowledge Gate

As the prover did not include any randomness so far, the proof elements in 5.2 are not zero-knowledge. Proving the same statement twice leads to equivalent proof elements and consequently an adversary can learn more then just the correctness of the statement. To make the proof elements indistinguishable from uniform random, we introduce a

randomization gate. This gate has no connection to the rest of the circuit and is suggested to be the last gate regarding its index. This choice is arbitrary however. The witness vector has to be extended by two more elements we call a_m and a_{m-1} to satisfy this gate. Its structure and constraint representation is shown in figure 5.3. The prover picks $r \xleftarrow{\$} \mathbb{Z}_p^*$ and set $a_m = r * r + M(G^r)$ and $a_{m-1} = r$. The proof elements are now uniformly random iff the secret point of evaluation is in $[m+1, p]$, since for any family of polynomials $\{R_i(x)\}_i$, $\langle \mathbf{a}, \mathbf{R}(x)\rangle_{[m]} = \langle \mathbf{a}, \mathbf{R}(x)\rangle_{[m-2]} + a_{m-1}R_{m-1}(x) + a_m R_m(x)$ is a polynomial of degree at most $n-1$ that intersect with itself when a different a_{m-1} is chosen, at most at 2 points within $[m+1, p]$. Note that this contradicts our requirement of choosing the secret point of evaluation from $[0, m]$. Another flaw in our protocol. A different approach to randomize proof elements therefor would be required.

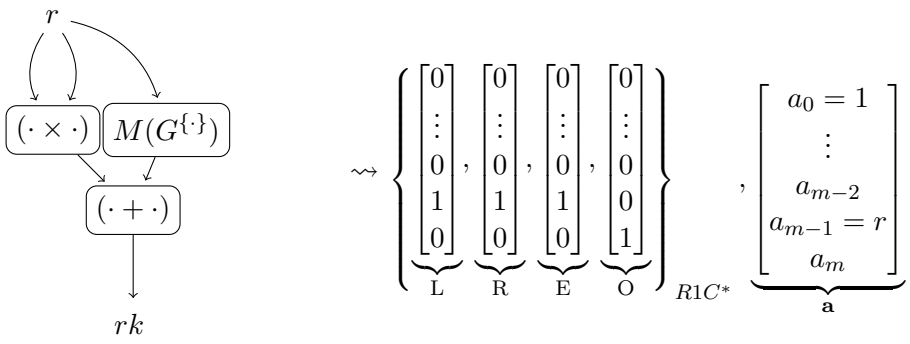
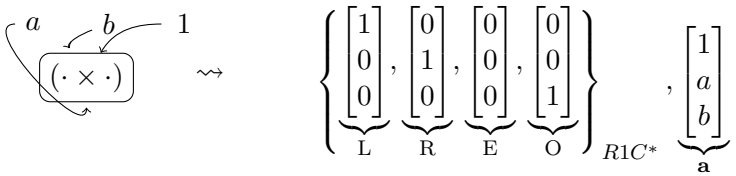


Figure 5.3: Shows the randomization gate and constraint. It enforces $a_m \equiv a_{m-1} \cdot a_{m-1} + M(G^{a_{m-1}}) \bmod p$. Since the r^2 might reveal some information, the randomization circuit could use three different random inputs for the price of increasing $|\mathbf{a}|$ by 4 instead of 2 elements.

5.1.2 Equality Assertion Gate

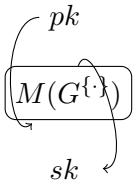
Checking weather two values are equal (or more precisely: Checking weather two values are in the same residue class) is essential for every R1CS as well as R1CS* based proof system. It is needed to verify signatures, knowledge of some DLP relation, proving that some derived Merkle-root hash is equal to some input hash etc. In general the equality gate enables recursive proving e.g. on can provide a proof that contains the verification of some other proof which in turn could be a proof of some other proof and so on. In therms of R1CS(*), a gate that ensures that $a = b$ is simple to realize via:



In our language the inbuilt function: $equal(\cdot,\cdot)$ introduces such an equality assertion constraint. Note that an equality assertion constraint does not increase the witness size.

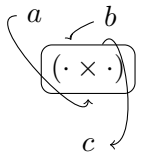
5.1.3 Inverse Gate

In case a public key is passed into the circuit as an argument and we want to proof knowledge of the secret key, we immediately observe that one gate to perform this inversion is sufficient since gates are only a guarantee that the input-output relation is satisfied and therefore they can be used in either direction.



$$\rightsquigarrow \quad \left\{ \underbrace{\begin{bmatrix}0\\0\end{bmatrix}}_{\mathrm{L}}, \underbrace{\begin{bmatrix}0\\0\end{bmatrix}}_{\mathrm{R}}, \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{\mathrm{E}}, \underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{\mathrm{O}} \right\}_{R1C^*}, \underbrace{\begin{bmatrix}pk\\sk\end{bmatrix}}_{\mathbf{a}}.$$

Division can be understood likewise. Consider the example where we need to compute $c = a/b$ given a, b. The corresponding circuit and R1C* is



$$\rightsquigarrow \quad \left\{ \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{\mathrm{L}}, \underbrace{\begin{bmatrix}0\\0\\1\end{bmatrix}}_{\mathrm{R}}, \underbrace{\begin{bmatrix}0\\0\\0\end{bmatrix}}_{\mathrm{E}}, \underbrace{\begin{bmatrix}1\\0\\0\end{bmatrix}}_{\mathrm{O}} \right\}_{R1C^*}, \underbrace{\begin{bmatrix}a\\b\\c\end{bmatrix}}_{\mathbf{a}}.$$

5.1.4 Combined Gate

Consider a example where we want to apply multiplication and curve group multiplication at once in one gate. It can be done but it is **very important** to notice possible side effects! Lets say we have a, b, c and want to compute $d = a * b$ and $f = M(G^c)$:

$$\left\{ \underbrace{\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}}_{\mathrm{L}}, \underbrace{\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}}_{\mathrm{R}}, \underbrace{\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}}_{\mathrm{E}}, \underbrace{\begin{bmatrix}0\\0\\0\\1\\1\end{bmatrix}}_{\mathrm{O}} \right\}_{R1C^*}, \underbrace{\begin{bmatrix}a\\b\\c\\d\\f\end{bmatrix}}_{\mathbf{a}},$$

would be **insufficient** since all we could derive from this assignment is that $d + f = a * b + M(G^c)$. If we continue to use d and f we loose their determinism, however they remain entangled. If another constraint assigns a value to one of them, the other one will be fixed too. This phenomena we call 'un-spooky action in a circuit'.

5.1.5 Simple Elliptic Curve Point Multiplication Circuit

Lets consider the case, where only one point multiplication defines the circuit e.g. the prover wants to convince the verifier that it knows sk s.t $pk = M(G^{sk})$ without revealing sk. M maps a curve point onto its x coordinate for example. The circuit, its corresponding constraint and the assignment vector become

$$\begin{array}{c} sk \\ \downarrow \\ \boxed{M(G^{\{\cdot\}})} \\ \downarrow \\ pk \end{array} \quad \rightsquigarrow \quad \left\{ \underbrace{\begin{bmatrix}0\\0\end{bmatrix}}_{\mathrm{L}}, \underbrace{\begin{bmatrix}0\\0\end{bmatrix}}_{\mathrm{R}}, \underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{\mathrm{E}}, \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{\mathrm{O}} \right\}_{R1C^*}, \underbrace{\begin{bmatrix}sk\\pk\end{bmatrix}}_{\mathbf{a}},$$

therefore the polynomials $L_1(X) = L_2(X) = R_1(X) = R_2(X) = E_1(X) = O_2(X) = 0$, $E_2(X) = O_1(X) = 1$ are simple constant polynomials. The domain polynomial times the extracted polynomial therefore also $Q(X)D(X)$=0. The statement $\phi = (pk)$, the witness $w = (sk)$.

The proof elements become:

$$\begin{aligned} A &= G^{\alpha+sk} \\ B &= H^0 \\ C &= G^{\frac{\beta sk}{\delta}} \\ F &= G^{sk} \end{aligned}$$

The verifier checks if

$$e\left(A, B \cdot H^{\beta}\right) \cdot e\left(G, H\right)^{M(F)} = e\left(G^{\alpha}, H^{\beta}\right) \cdot e\left(G^{\frac{pk}{\gamma}}, H^{\gamma}\right) \cdot e\left(C, H^{\delta}\right) \cdot e\left(F, B\right)$$

what is indeed true since

$$\begin{aligned} e\left(A, B \cdot H^{\beta}\right) \cdot e\left(G, H\right)^{M(F)} &= \\ e\left(G, H\right)^{(\alpha+sk)\beta} \cdot e\left(G, H\right)^{pk} &= \\ e\left(G, H\right)^{\alpha\beta} \cdot e\left(G, H\right)^{\frac{pk}{\gamma}\cdot\gamma} \cdot e\left(G, H\right)^{\frac{\beta sk}{\delta}\cdot\delta} \cdot e\left(G, H\right)^{sk\cdot 0} &= \\ e\left(G^{\alpha}, H^{\beta}\right) \cdot e\left(G^{\frac{pk}{\gamma}}, H^{\gamma}\right) \cdot e\left(C, H^{\delta}\right) \cdot e\left(F, B\right) \quad \square \end{aligned}$$

From this we derive a simple cryptographic protocol: Proving knowledge of sk, s.t. $pk = G^{sk}$: Verifier: pick random $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$ send it to the prover. Prover: set the proof $\pi = (A) = (G^{sk+\alpha})$ Verifier take $\pi = (A), \alpha, pk$ and accept iff:

$$e(pk, H) \cdot e(G, H)^{\alpha} \stackrel{?}{=} e(A, H). \tag{5.9}$$

CHAPTER 6

Conclusion

The mindful reader certainly noticed the wide spectrum of topics contained within this thesis. We started by giving our own definition of a Satoshi blockchain where we put emphasis on the networks inability to distinguish attackers from honest users in order to maintain public accessibility and where we disallow advantages in mining strategies other then the expense of a resource. In the chapter on privacy enhancing technologies we explain and motivate the need for strong cryptographic tools and treat sender, recipient and transaction content anonymity in separate sections. This chapter introduces the zkSNARKs which then become the main point of our investigations. In oder to understand how these general purpose proof systems operate we provide a theoretic description aimed for undergraduates in computer-science and a full implementation of the entire process chain required for the creation of a zkSNARK. We programmed our own domain specific context free language and discovered optimizations to reduce the R1CS's description length. On top of all this we extend the commonly used relation of 'quadratic arithmetic programs' to something we named 'extended algebraic program' with the intention to outsource proofs of knowledge of a discrete logarithm on an elliptic curve from the circuit to the prover side only. Our construction is publicly verifiable. A trusted third-party generates a proving key and a verification key. Afterwards anyone can use the proving key to generate non-interactive proofs for adaptively-chosen NP statements, and the proofs can be verified by anyone using the verification key. The extension was successfully implemented however turned out to have very limited statistic knowledge soundness that is proportional to number of multiplication gates in the underlying circuit, which therefore is not suitable for cryptographic applications. We came up with a pairing based proof system for the purpose of proving knowledge of a discrete logarithm, as well as we were able to formulate and generate a Sudoku zkSNARK with our own zkSNARK creation toolchain. The satisfiability check program is purely based on number-theoretic considerations and therefor it is simple, beautiful and fast.

Bibliography

[AGM18] Agrawal, Shashank ; Ganesh, Chaya ; Mohassel, Payman: *Non-Interactive Zero-Knowledge Proofs for Composite Statements*. Cryptology ePrint Archive, Report 2018/557, 2018. – https://eprint.iacr.org/2018/557, visited 2020-06-28

[AM15] Adam Mackenyie, Monero Core T. Surae Noether N. Surae Noether: Improving Obfuscation in the CryptoNote Protocol. (2015). – https://ww.getmonero.org/resources/research-lab/, visited 2020-06-28

[B+02] Back, Adam u. a.: Hashcash-a denial of service counter-measure. (2002). – ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf, visited 2020-06-28

[BB04a] Boneh, Dan ; Boyen, Xavier: Efficient selective-ID secure identity-based encryption without random oracles. In: *International conference on the theory and applications of cryptographic techniques* Springer, 2004, S. 223–238

[BB04b] Boneh, Dan ; Boyen, Xavier: Short signatures without random oracles. In: *International conference on the theory and applications of cryptographic techniques* Springer, 2004, S. 56–73

[BBB+18] Bünz, Benedikt ; Bootle, Jonathan ; Boneh, Dan ; Poelstra, Andrew ; Wuille, Pieter ; Maxwell, Greg: Bulletproofs: Short proofs for confidential transactions and more. In: *2018 IEEE Symposium on Security and Privacy (SP)* IEEE, 2018, S. 315–334

[BF01] Boneh, Dan ; Franklin, Matt: Identity-based encryption from the Weil pairing. In: *Annual international cryptology conference* Springer, 2001, S. 213–229

[BFM88] Blum, Manuel ; Feldman, Paul ; Micali, Silvio: Non-interactive zero-knowledge and its applications. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 1988

[BLS01] Boneh, Dan ; Lynn, Ben ; Shacham, Hovav: Short signatures from the Weil pairing. In: *International Conference on the Theory and Application of Cryptology and Information Security* Springer, 2001, S. 514–532

[Bon] Boneh, Dan: boneYT. . – https://www.youtube.com/watch?v=F4x2kQTKYFY, visited 2020-06-28

[can] *Reuters' reference on The Financial Times article.* – http://web.archive.org/web/20140309004338/http://uk.reuters.com/article/2009/01/03/idUKPTIP32510920090103, visited 2020-06-28

[CVH91] Chaum, David ; Van Heyst, Eugène: Group signatures. In: *Workshop on the Theory and Application of of Cryptographic Techniques* Springer, 1991, S. 257–265

[DGM14] Danezis George, Groth J. Fournet Cédric C. Fournet Cédric ; Markulf, Kohlweiss: Square span programs with applications to succinct NIZK arguments. In: *International Conference on the Theory and Application of Cryptology and Information Security* Springer, 2014, S. 532–550

[DN92] Dwork, Cynthia ; Naor, Moni: Pricing via processing or combatting junk mail. In: *Annual International Cryptology Conference* Springer, 1992, S. 139–147

[Dud] Dudley, Rick: Parsimonious Answers to Difficult Questions. . – https://www.youtube.com/watch?v=GOkSg0BuSdw&feature=youtu.be, visited 2020-06-28

[EBSV17] Eli Ben-Sasson, Eran T. Alessandro Chiesa C. Alessandro Chiesa ; Virza, Madars: Scalable zero knowledge via cycles of elliptic curves. In: *Algorithmica* 79 (2017), Nr. 4, S. 1102–1160

[FMR99] Frey, Gerhard ; Muller, Michael ; Ruck, H-G: The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. In: *IEEE Transactions on Information Theory* 45 (1999), Nr. 5, S. 1717–1719

[FR94] Frey, Gerhard ; Rück, Hans-Georg: A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. In: *Mathematics of computation* 62 (1994), Nr. 206, S. 865–874

[GGPR13] Gennaro, Rosario ; Gentry, Craig ; Parno, Bryan ; Raykova, Mariana: Quadratic span programs and succinct NIZKs without PCPs. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques* Springer, 2013, S. 626–645

[GKL14] Garay, Juan ; Kiayias, Aggelos ; Leonardos, Nikos: *The Bitcoin Backbone Protocol: Analysis and Applications*. Cryptology ePrint Archive, Report 2014/765, 2014. – https://eprint.iacr.org/2014/765, visited 2020-06-28

[GMR89] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The knowledge complexity of interactive proof systems. In: *SIAM Journal on computing* 18 (1989), Nr. 1, S. 186–208

[GPS08] Galbraith, Steven D. ; Paterson, Kenneth G. ; Smart, Nigel P.: Pairings for cryptographers. In: *Discrete Applied Mathematics* 156 (2008), Nr. 16, S. 3113–3121

[Gro16] Groth, Jens: On the size of pairing-based non-interactive arguments. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques* Springer, 2016, S. 305–326

[HBHW16] Hopwood, Daira ; Bowe, Sean ; Hornby, Taylor ; Wilcox, Nathan: Zcash protocol specification. In: *GitHub: San Francisco, CA, USA* (2016)

[Her16] Herrmann, Ulrike: *Kein Kapitalismus ist auch keine Lösung: Die Krise der heutigen Ökonomie oder Was wir von Smith, Marx und Keynes lernen können*. Westend Verlag, 2016

[Jou02] Joux, Antoine: The Weil and Tate pairings as building blocks for public key cryptosystems. In: *International Algorithmic Number Theory Symposium* Springer, 2002, S. 20–32

[kov] Kovri:A free, decentralized, anonymity technology based on I2P's open specifications. . – https://gitlab.com/kovri-project/kovri, visited 2020-06-28

[LLP+19] Li, Jingming ; Li, Nianping ; Peng, Jinqing ; Cui, Haijiao ; Wu, Zhibin: Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies. In: *Energy* 168 (2019), S. 160–168

[Lor] Loraine: Selling Shovels In a Gold Rush. . – https://self-made.io/selling-shovels-gold-rush-10-companies-got-rich-big-economic-trends/2357/, visited 2020-06-28

[LSP82] Lamport, Leslie ; Shostak, Robert ; Pease, Marshall: The Byzantine generals problem. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4 (1982), Nr. 3, S. 382–401

[Lyn07] Lynn, Ben: *On the implementation of pairing-based cryptosystems*, Stanford University Stanford, California, Diss., 2007

[Max] Maxwell, Greg: A trip to the moon requires a rocket with multiple stages. . – https://www.reddit.com/r/Bitcoin/comments/438hx0/a_trip_to_the_moon_requires_a_rocket_with/, visited 2020-06-28

[May16] Mayer, Hartwig: zk-SNARK explained: Basic Principles. (2016). – https://blog.coinfabrik.com/wp-content/uploads/2017/03/zkSNARK-explained_basic_principles.pdf, visited 2020-06-28

[N+08] Nakamoto, Satoshi u. a.: Bitcoin: A peer-to-peer electronic cash system. (2008)

[NNS10] Naehrig, Michael ; Niederhagen, Ruben ; Schwabe, Peter: New software speed records for cryptographic pairings. In: *International Conference on Cryptology and Information Security in Latin America* Springer, 2010, S. 109–123

[Noe15] Noether, Shen: Ring SIgnature Confidential Transactions for Monero. In: *IACR Cryptology ePrint Archive* 2015 (2015), S. 1098

[NY90] Naor, Moni ; Yung, Moti: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990, S. 427–437

[PBF+18] Poelstra, Andrew ; Back, Adam ; Friedenbach, Mark ; Maxwell, Gregory ; Wuille, Pieter: Confidential assets. In: *International Conference on Financial Cryptography and Data Security* Springer, 2018, S. 43–63

[PHGR13] Parno, Bryan ; Howell, Jon ; Gentry, Craig ; Raykova, Mariana: Pinocchio: Nearly practical verifiable computation. In: *2013 IEEE Symposium on Security and Privacy* IEEE, 2013, S. 238–252

[Pip90] Pippenger, Nicholas: Developments in “the synthesis of reliable organisms from unreliable components”. In: *The legacy of John von Neumann* (1990), S. 311–324

[QQQ+89] Quisquater, Jean-Jacques ; Quisquater, Myriam ; Quisquater, Muriel ; Quisquater, Michaël ; Guillou, Louis ; Guillou, Marie A. ; Guillou, Gaïd ; Guillou, Anna ; Guillou, Gwenolé ; Guillou, Soazig: How to explain zero-knowledge protocols to your children. In: *Conference on the Theory and Application of Cryptology* Springer, 1989, S. 628–631

[Rei16] Reitwiessner, Christian: zkSNARKs in a nutshell. In: *Ethereum Blog* 6 (2016)

[RSW96] Rivest, Ronald L. ; Shamir, Adi ; Wagner, David A.: Time-lock puzzles and timed-release crypto. (1996)

[Sch19] Schneier, Bruce: There's No Good Reason to Trust Blockchain Technology. In: *Wired* (2019). – https://www.wired.com/story/theres-no-good-reason-to-trust-blockchain-technology/, visited 2020-06-28

[SKG19] Stoll, Christian ; Klaassen, Lena ; Gallersdörfer, Ulrich: The carbon footprint of bitcoin. In: *Joule* 3 (2019), Nr. 7, S. 1647–1661

[SN13] Saberhagen Nicolas van: CryptoNote v 2.0. (2013). – https://ww.getmonero.org/resources/research-lab/, visited 2020-06-28

[SN17] Sarang Noether, Brandon G.: An efficient implementation of Monero subaddresses. (2017). – https://ww.getmonero.org/resources/research-lab/, visited 2020-06-28

[SZ13] Sompolinsky, Yonatan ; Zohar, Aviv: Accelerating bitcoin's transaction processing. In: *Fast money grows on trees, not chains* (2013)

[Vir14] Virza, Madars: *SNARKs for C: verifying program executions succinctly and in zero knowledge*, Massachusetts Institute of Technology, Diss., 2014

[Wol] Wolf, Mathias: go-AlgebraicProgram-SNARK. . – https://github.com/mottla/go-AlgebraicProgram-SNARK, visited 2020-06-28

[YYHK17] Yamakawa, Takashi ; Yamada, Shota ; Hanaoka, Goichiro ; Kunihiro, Noboru: Self-bilinear map on unknown order groups from indistinguishability obfuscation and its applications. In: *Algorithmica* 79 (2017), Nr. 4, S. 1286–1317

[Zam] Zamfir, Vlad: Blockchain Governance. . – https://www.youtube.com/watch?v=PKyk5DnmW50, visited 2020-06-28

[zca19] *zk-SNARK explained: Basic Principles.* 2019. – https://z.cash/technology/zksnarks/, visited 2020-06-28