UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Shahla Atapoor

# On Privacy Preserving Blockchains and zk-SNARKs

Master's Thesis (30 ECTS)

Supervisor: Helger Lipmaa, PhD
Co-Supervisor: Janno Siim, MSc
Co-Supervisor: Karim Baghery, MSc

Tartu 2019

# On Privacy Preserving Blockchains and zk-SNARKs

**Abstract:**
During last few years, along with blockchain technology, cryptocurrencies have found huge attention from both commercial and scientific perspectives. Cryptocurrencies are digital coins which use cryptographic tools to allow secure peer-to-peer monetary transactions. Bitcoin is the most well-known cryptocurrency that allows direct payments between pseudonyms without any third party. If a user's pseudonym is linked to her identity, all her transactions will be traceable, which will violate her privacy. To address this, various privacy-preserving cryptocurrencies have been proposed that use different cryptographic tools to achieve anonymous transactions. Zerocash is one of the most popular ones that uses zero-knowledge proofs to hide the source, destination and value of each transaction.

This thesis consists of two main parts. In the first part, after a short overview of some cryptocurrencies (precisely Bitcoin, Monero and Zerocoin), we will explain the construction of Zerocash cryptocurrency and discuss the intuition behind the construction. More precisely, we will introduce the deployed primitives and will discuss the role of each primitive in the construction of the coin. In particular, we explain zero-knowledge Succinct Non-Interactive Arguments of Knowledge (a.k.a. zk-SNARKs) that play the main role in achieving strong privacy in Zerocash. Due to the importance of zk-SNARKs in privacy-preserving applications, in the second part of the thesis, we will present a new variation of Groth's 2016 zk-SNARK that currently is the most efficient pairing-based scheme. The main difference between the proposed variation and the original one is that unlike the original version, new variation guarantees non-malleability of generated proofs. Our analysis shows that the proposed changes have minimal effects on the efficiency of the original scheme and particularly it outperforms Groth and Maller's 2017 zk-SNARK that also guarantees non-malleability of proofs.

**Keywords:**
Privacy preserving coin, Zerocash, Zero-knowledge proof, zk-SNARK

**CERCS:** P175 Computer Science

## Privaatsed plokiahelad ja zk-SNARKid

**Lühikokkuvõte:** Viimastel aastatel on krüptoraha ja plokiahela tehnoloogia leidnud suurt tähelepanu nii kaubanduslikust kui ka teaduslikust vaatenurgast. Krüptoraha kujutab endast digitaalseid münte, mis kasutades krüptograafilisi vahendeid võimaldab turvalisi tehinguid võrdvõrkudes. Bitcoin on kõige tuntum krüptoraha, mis võimaldab otsetehinguid kasutajate pseudonüümide vahel ilma, et oleks vaja kolmandaid osapooli. Paraku kui kasutaja pseudonüüm on seotud tema identiteediga, on kõik tema tehingud jälgitavad ning kaob privaatsus. Selle lahendamiseks on välja pakutud erinevaid privaatsust säilitavaid krüptorahasi, mis kasutavad anonüümsete tehingute saavutamiseks krüptograafilisi tööriistu. Zerocash on üks populaarseimatest privaatsetest krüptorahadest, mis kasutab iga tehingu allika, sihtkoha ja väärtuse varjamiseks nullteadmustõestust.

Antud töö koosneb kahest peamisest osast. Esimeses osas kirjeldame, pärast lühikest ülevaadet mõnest privaatsest krüptorahast (Bitcoin, Monero ja Zerocoin), Zerocashi konstruktsiooni ja anname intuitsiivse seletuse selle tööpõhimõttele. Me tutvustame kasutuselevõetud primitiive ja arutleme iga primitiivi rolli üle mündi konstruktsioonis. Erilist tähelepanu pöörame kompaktsetele nullteadmustõestusetele (zk-SNARKidele), millel on peamine roll Zerocashis. Kuna nullteadmustõestus on niivõrd olulisel kohal Zerocashis (ja teistes privaatsetes rakendustes) siis töö teises osas pakume välja uue variatsiooni Grothi 2016. aasta zk-SNARKile, mis on seni kõige tõhusam. Erinevalt Grothi konstruktsioonist, meie variatsioonis ei ole võimalik tõestusi modifitseerida. Muudatused mõjutavad nullteadmustõestuse tõhusust vaid minimaalselt ning meie konstruktsioon on kiirem kui Grothi ja Malleri 2017. nullteadmustõestus, mis samuti välistab muudetavuse.

**Võtmesõnad:**
Privaatne krüptoraha, Zerocash, Nullteadmustõestus, zk-SNARK

**CERCS:** P175 Informaatika

# Dedication

Dedicated to the memory of my Mam,

**Maliheh Hadbi**

who always believed in my ability to be
successful in the academic arena.

# Publications.

- Shahla Atapoor and Karim Baghery. Simulation Extractability in Groth's zk-SNARK; Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2019 International Workshops, DPM 2019 and CBT 2019, September 26-27, 2019, Luxembourg [AB19].

# Acknowledgments.

I would like to express my deep and sincere gratitude to my research supervisor Prof. Helger Lipmaa and co-supervisors Karim Baghery and Janno Siim for giving me the opportunity to do research and providing invaluable guidance throughout this research.

In addition to supervisors, my thanks go to Dr. Kateryna Pavlyk, Dr. Liina Tammekänd, all my kind friends and my classmates that have contributed immensely to my personal and professional time at Tartu university and without their support, I would not be able to continue studying.

I am extremely grateful to my family specially my sister for their love, caring and sacrifices for educating and preparing me for my future.

Finally, I would like to thank my Mom for her rare and unassuming love. I am following my Mom's dream of study and I hope she is happy in the sky.

The author was supported by the Estonian Research Council grant (PRG49) and the IT academy Specialization Stipend of academic year 2017/2018.

# Contents


**1 Introduction** **9**
- 1.1 Goals and Contribution . . . 11
- 1.2 Outline . . . 12

**2 Preliminaries** **13**
- 2.1 Commitment Scheme . . . 13
  - 2.1.1 Pedersen Commitment Scheme. . . . 14
- 2.2 Key-Private Public-Key Encryption . . . 15
  - 2.2.1 IND-CCA . . . 15
  - 2.2.2 IK-CCA . . . 16
- 2.3 Digital Signatures . . . 16
  - 2.3.1 One-Time Secure Digital Signatures . . . 16
  - 2.3.2 Linkable Ring Signatures . . . 17
- 2.4 One-Way Functions . . . 18
  - 2.4.1 Universal One-Way Hash Functions . . . 18
  - 2.4.2 Pseudo-Random Functions . . . 19
- 2.5 Nakamoto Consensus and Proof-of-Work . . . 19
- 2.6 Decentralized Anonymous Payment (DAP) Schemes . . . 20
  - 2.6.1 Data Structures . . . 20
  - 2.6.2 Definition of Algorithms . . . 22
  - 2.6.3 Requirements in a DAP Scheme . . . 23
- 2.7 Non-Intractive Zero-Knowledge (NIZK) Proofs . . . 24
  - 2.7.1 zk-SNARKs . . . 24
  - 2.7.2 Groth's zk-SNARK . . . 27

**3 Privacy in Digital Coins** **29**
- 3.1 Bitcoin . . . 29
  - 3.1.1 Overview . . . 29
  - 3.1.2 Construction . . . 30
  - 3.1.3 Security and Privacy Concerns . . . 31
- 3.2 Monero . . . 31
  - 3.2.1 MLSAG Signature . . . 32
  - 3.2.2 Ring Confidential Transaction . . . 32
  - 3.2.3 Privacy of Ring CT. . . . 32
- 3.3 Zerocoin . . . 33
  - 3.3.1 Construction . . . 34
  - 3.3.2 Security . . . 34
- 3.4 Zerocash . . . 35
  - 3.4.1 Construction . . . 35
  - 3.4.2 Instantiation . . . 40
  - 3.4.3 Security . . . 41
  - 3.4.4 Efficiency . . . 41

**4 An Efficient Simulation Extractable zk-SNARK** **43**
4.1 A Variation of Groth's zk-SNARK . . . . . 45
4.1.1 New Construction . . . . . 45
4.1.2 Security Proofs . . . . . 46
4.2 Instantiation and Efficiency Evaluation . . . . . 51

**5 Conclusion** **53**

**References** **58**

# 1 Introduction

Distributed ledger and blockchain is a technology which stores data on a distributed blocks are only in blockchain. In general distributed ledgers might not contain blocks that are linked together by can it ever be anything else than hash function?. Due to distributed nature of the network, there is no central node and central storage, while there is a consensus protocol to extend the ledger concurrently continuously. As a ledger is kept by all nodes of the network, so once a piece of data is entered to the ledger, it is difficult to remove or modify the data. Among various application that this technology offers, the most known one is cryptocurrencies such as Bitcoin [Nak08].

A cryptocurrency is a digital asset which allows one payer to perform direct payments (transactions) without interference of a third party, unlike in classic electronic payment systems. Cryptocurrencies use various cryptographic operations and primitives, but the main technology behind them is the blockchain. Namely, a chain of blocks (a.k.a. ledger) is shared between a large number of nodes that is write-only and records list of transactions between addresses (e.g. account numbers) of individual users. Nodes of the network run a consensus protocol to get agreement on the transactions that should be recorded in the ledger. As a result there is no central point to make a decision about a transaction to be recorded, instead if majority of nodes (more precisely computation power of nodes) confirm a valid transaction, then it will be recorded in the ledger and the ledger will be extended. The most famous and widely used cryptocurrency is Bitcoin (at time of writing the thesis) that was proposed by an unknown person or a group of people; known as Satoshi Nakamoto [Nak08].

In Bitcoin, each user has an address and a secret key to control the address. Strictly speaking, a user can get a direct transaction on her public address, but in order to be able to spend the received coin, she must know the secret key of that address. These public addresses of users are known as *pseudonyms*. The system avoids double spending a coin with broadcasting all transactions to all nodes and recording them on a public distributed ledger. As the ledger of Bitcoin is distributed and public, and transactions are done between pseudonyms, then once a user's pseudonym is linked to her identity, all her transactions will be traceable which violates her privacy. Indeed, there are various studies [RH11, RS13, FKP15] that looking at the transactions' graph, by some empirical analysis, can extract lots of information about the goal of transaction, amount of money and the sender of the transaction, which means that system leaks some personal information. In practice, such attacks can have different negative effects. As an example, if you are a merchants, once your competitor knows what your address is, they can track your cash flow, track your costumers and a lot of valuable business information may be leaked.

To address this concern, various privacy preserving cryptocurrencies have been proposed [MGGR13, BCG$^+$14a, Noe15] that use different cryptographic techniques to provide strong security for end-users. For instance by hiding the source, destination and the value of a transaction. A possible solution for hiding the sender of a transaction is mixing the transactions by a trusted party, but this would require trust in a third party. In this scenario, the Bitcoin users need to send their coins to a third party, to use laundry for mixing the coins, and wash them up, and get them back. (They get different coins but with the same value). This approach still has some problems including that the time which is needed to have the coins completely mixed must be large enough and the possibility of stealing and tracing the coin by the mixer. On the other hand, the legitimate users require following: (1) they want to have secure transactions, (2) they want to it without having much effort, and (3) sometimes they may even not be aware of

weakness of privacy.

Zerocoin that is presented by Mires et al. [MGGR13] tackled the mentioned problems with an interesting cryptographic technique. In their system, the users who wish to have their transaction history hidden can use the help of Mixes (laundries). Zerocoin Created an anonymous protocol that users do not need to trust to the third party, since there is a single party (third party) that uses cryptographic proofs to ensure that the mixing was done correctly. Zerocoin does have some constrains in performance and functionality. Regarding performance, one problem is that, those proofs that are required to ensure the correct operation of the mix are about 45 KB per transaction (for 128-bit security) and they take about half a second to verify. Moreover these mixes can only work on the values of the same denomination, which means that one cannot split values and just can spend multiple coins of the same denominations. It is discussed that this can leak information about the transactions [BCG$^+$14a]. In Zerocoin each user still has to take his coins, load them to laundry service and wash them clean and get them back, which still requires partial trust to a third party.

Monero [Noe15] is another elegant cryptocurrency that deals with privacy concern in Bitcoin. Monero uses ring signatures which allows a spender remain hidden in a group of potential signers. Technically speaking, a ring signature mixes the spender's input with a group of individuals, which makes it computationally hard to make a connection or link between each subsequent transaction [Noe15]. To guarantee privacy of recipient of a transaction, Monero generates *stealth address* for each transaction that make the transaction anonymous to anyone except the spender and recipient. The value of each transaction is hidden by committing to the values.

A recent proposal Quisquis [FMMO18] aims to solve the same privacy concern in Bitcoin by combining various cryptographic tools including updatable public keys, shuffle arguments and zero-knowledge proofs. Intuitively, in their coin, a spender gives a zero-knowledge shuffle proof for updating commitments and public key of two coins (coins of sender and receiver) among a set of coins.

Ben-Sasson et al. [BCG$^+$14a] presented an improved version of Zerocoin [MGGR13] and Pinocchio Coin [DFKP13] that also allows completely anonymous transactions. Zerocash uses an efficient type of zero-knowledge proofs [GMR89] that are known as zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARK) [PHGR13].

Among various type of Non-Interactive Zero-Knowledge (NIZK) arguments, zk-SNARKs [Gro10, Lip12, PHGR13, BCTV13, Gro16, GM17] are the most efficient and practically interesting zero-knowledge proofs that have appeared in various applications including cloud computing [PHGR13], privacy-preserving cryptocurrencies [BCG$^+$14a], privacy-preserving smart contracts [KMS$^+$16, JKS16, Bag19], distributed storage [BG18a] and faster ledger verification [MS18]. Majority of known zk-SNARKs are designed to guarantee *completeness*, *zero-knowledge* and (non-black-box) *knowledge soundness* [PHGR13, BCTV13, Gro16]. Completeness guarantees that an honest prover always convinces an honest verifier. Non-black-box *knowledge soundness* ensures that a malicious prover cannot convince the honest verifier, unless she knows the witness. Equivalently, if an adversarial prover can convince a verifier, she must *know* the witness. Zero-knowledge property assures that the proof generated by prover does not leak any information about the witness other than the validity of the statement. Formal definitions will be given later in Sec. 2.7.1. Currently the most efficient zk-SNARK is the one that is proposed by Groth in Eurocrypt 2016 [Gro16]. Groth's zk-SNARK is constructed for Quadratic Arithmetic Programs [1] (QAPs) and works over bilinear groups. While using zk-SNARKs in prac-

[1] As an NP language where for an input $x$ and witness $w$, $(x, w) \in \mathbf{R}$ can be verified by using a parallel quadratic

tical systems, their *knowledge soundness* property is not enough to guarantee non-malleability of proofs. In other words, knowledge-sound proofs are vulnerable to the man-in-the-middle attacks where adversary modifies and reuses an existing proof. Due to this fact, zk-SNARKs that only guarantee knowledge-soundness cannot be deployed in many of practical applications straightforwardly [BCG$^+$14a, KMS$^+$16, JKS16, Bag19]. For instance, privacy-preserving crypocurrencies such as Zerocash that uses zk-SNARKs [BCTV13, Gro16] as a subroutine, takes extra steps to prevent malleability attacks in the SNARK proofs for *pour* transactions [BCG$^+$14a]. Similarly, privacy-preserving smart contract systems [KMS$^+$16, JKS16] show that knowledge-soundness of zk-SNARKs is not enough for their systems. *Simulation-knowledge soundness* which also is known as *simulation extractability*, is an amplified version of knowledge-soundness that is proposed to achieve extractable and non-malleable proofs. A Simulation-Extractable (SE) zk-SNARK guarantees the proof is succinct, zero-knowledge and *simulation-extractable*, meaning that an adversarial prover is unable to come out with a new proof unless he knows a witness, even if he has already seen arbitrary number of simulated proofs.

As mentioned before, Groth's zk-SNARK [Gro16] is the most efficient zk-SNARK in the CRS model which requires a trusted third party to generate a common reference string shared between prover and verifier, but it does not guarantee non-malleability of proofs. In Crypto 2017, Groth and Maller [GM17] proposed the first SE zk-SNARK that is constructed for Square Arithmetic Programs [2] (SAPs) and guarantees non-malleability of proofs. In practice, Groth and Maller's SAP-based SE zk-SNARK[Gro16] is significantly more inefficient than Groth's QAP-based zk-SNARK [KZM$^+$15]. Later in Sec. 4 we will compare empirical performance of both zk-SNARKs. To recap, we construct an efficient version of Groth's zk-SNARK which will guarantee non-malleability of proofs and will act better than Groth and Maller's SE zk-SNARK [GM17] in all efficiently metrics, which can be used in practically interesting constructions.

## 1.1 Goals and Contribution

Roughly speaking, this thesis chases two main goals. The First part focuses on the privacy-preserving cryptocurrencies and the second part considers achieving simulation extractability in the stat-of-the-art zk-SNARK.

During last few years various practical systems [KMS$^+$16, JKS16, CMP$^+$19] have started to use Zerocash protocol to enable privacy-preserving payments. Initial goal of the thesis is to present a high level explanation of Zerocash cryptocurrency while describing the main ideas behind its structure. We additionally highlight main differences between Zerocash and some other cryptocurrencies. To this aim, after a short overview of some cryptocurrencies including Bitcoin and Monero, we will go trough the construction of Zerocash step-by-step and describe the main ideas and intuition behind the structure of each part (or algorithm) of the coin. Precisely speaking, we will introduce the primitives deployed in the construction of coin and will discuss the role of each one. In particular, we will explain zk-SNARKs that is the main primitive in construction of Zerocash to achieve strong anonymity. We will introduce the state-of-the-art zk-SNARK that recently Zerocash has started to use.

In the second part of the thesis, due to importance of zk-SNARKs in privacy preserving applications, we propose a variation of Groth's zk-SNARK [Gro16] that can achieve simulation-

check.

[2] As an NP language where for an input $x$ and witness $w$, $(x, w) \in \mathbf{R}$ can be verified by using a parallel squaring check.

extractability and consequently can guarantee non-malleability of proofs. To this aim, we show that one can define a new language $L'$, using an OR-based construction, based on original language $L$ in Groth's zk-SNARK that is appended with a pseudo-random function $f_s(\cdot)$ and a commitment of the secret randomness $s$. The secret randomness $s$ will act as a key for a pseudorandom function. Based on new language $L'$ we construct a variation of original scheme that can achieive simulation extractability. The intuition for a pseudo-random function $f_s(\cdot)$ is that without the knowledge of the key $s$, $f_s(\cdot)$ behaves like a true random function. However, given $s$, one can compute $f_s(\cdot)$ easily. In new language $\mathbf{L}'$, for a statement-witness pair to be valid, either a witness for $\mathbf{R_L}$ is provided (by honest prover) or an opening to commitment of $s$ together with the value of $f_s(\mathsf{pk}_{\mathsf{Sign}})$ is provided (by simulator), where $\mathsf{pk}_{\mathsf{Sign}}$ is the public key of a signature scheme. One may note that in order for a statement to pass the verification without a valid witness, the prover must generate $f_s(\mathsf{pk}_{\mathsf{Sign}})$ without the knowledge of $s$ which requires to break the pseudo-random function $f_s$.

Analysis and implementations show that in practical cases overload has minimal effects on the efficiency of the original scheme.

**Publications.** Our results in Sec. 4 is accepted to be published in proceeding of 3rd International Workshop on Cryptocurrencies and Blockchain Technology - CBT'19 [3] [AB19]. In the published paper, the author of this thesis read related literature, participated in several discussions and wrote initial version of security proofs for the proposed construction. Moreover, the author had active contribution in writing and revising the paper.

## 1.2 Outline

The structure of thesis is as follows: Section 2 presents an overview on some background concepts that are used in the later sections. We consider privacy in cryptocurrencies and introduce some known anonymous cryptocurrencies in Section 3. In the same section, we will explain Zerocash and the underlying algorithms with more details and explanations. As the second contribution of the thesis, in Section 4, we present a variation of Groth's zk-SNARK that can achieve simulation-extractability efficiently. Finally, we conclude the thesis in Section 5.

[3] http://deic.uab.cat/conferences/cbt/cbt2019/

# 2 Preliminaries

This section presents an overview of cryptographic primitives involved in the rest of thesis. We review the cryptographic primitives such as commitment schemes, key-private public key encryption, one-time secure digital signatures, one-way functions, Non-Interactive Zero-Knowledge (NIZK) arguments and particularly zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) that are an efficient type of NIZK arguments.

**Notations.** Let $n \in \mathbb{N}$ is the security parameter, say $n = 128$, and $1^n$ shows its unitary representation. We use $\Pr[x : y]$ to show the probability that $y$ happens for experiment $x$. A function $f : F \to \mathbb{R}$ with $F = \mathbb{N}, \mathbb{R}$ is *negligible* (shown with $\mathsf{negl}(n)$) iff for all $c > 0$ such that $f(x) \leq x^{-c}$ for sufficiently large $x$. Let PPT denote probabilistic polynomial-time, and NUPPT denote non-uniform PPT. All adversaries will be stateful. For an algorithm $\mathcal{A}$, let $\mathrm{im}(\mathcal{A})$ be the image of $\mathcal{A}$, i.e. the set of valid outputs of $\mathcal{A}$, let $\mathsf{RND}(\mathcal{A})$ denote the random tape of $\mathcal{A}$, and let $r \leftarrow \mathsf{RND}(\mathcal{A})$ denote sampling of a randomizer $r$ of sufficient length for $\mathcal{A}$'s needs. By $y \leftarrow \mathcal{A}(x; r)$ we denote the fact that $\mathcal{A}$, given an input $x$ and a randomizer $r$, outputs $y$. For algorithms $\mathcal{A}$ and $\mathsf{Ext}_{\mathcal{A}}$, we write $(y||y') \leftarrow (\mathcal{A}||\mathsf{Ext}_{\mathcal{A}})(x; r)$ as a shorthand for $y \leftarrow \mathcal{A}(x; r), y' \leftarrow \mathsf{Ext}_{\mathcal{A}}(x; r)$. We denote by $\mathsf{negl}(n)$ an arbitrary negligible function. For distributions $A$ and $B$, $A \approx_c B$ means that they are computationally indistinguishable.

## 2.1 Commitment Scheme

Commitment schemes are one of the key tools in cryptography that are deployed in various applications and larger cryptographic systems. A commitment scheme usually has two phases: committing phase and opening phase. At the committing phase, a committer commits to a secret value. After a while, in the opening phase, committer opens the commitment and reveals the committed value (and some secret information) that allows a verifier to check whether the same value has been committed.

The nature of a commitment scheme requires two essential properties that are known as *hiding* and *binding*. First requirement is that, the receiver of a commitment should not be able to learn any information about the committed value before opening it by the committer itself; this property is known as hiding. The second requirement from a commitment scheme is that the committer can not change the committed value after committing. In other words, the committer should not be able to commit a message in first phase and then open it to a different message; this property is known as binding. Later we will see that, commitment schemes play an essential role in construction of the DAP scheme behind the Zerocash to provide strong privacy. A commitment scheme formally can be defined as follows.

**Definition 1** (Commitment Schemes)**.** *A commitment scheme is a tuple of efficient algorithms* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{OpenVerify})$ *such that,*

- ***Key Generation Phase,*** $\mathsf{ck} \leftarrow \mathsf{KGen}(1^n)$*:Generates a commitment public key* ck*. It specifies a message space* $M_{\mathsf{ck}}$*, a randomness space* $R_{\mathsf{ck}}$*, and a commitment space* $C_{\mathsf{ck}}$*. Given* $1^n$*, the algorithm computes a commitment key* ck*.*

- ***Committing Phase,*** $(\mathsf{cm}, \mathsf{op}) \leftarrow \mathsf{Com}(\mathsf{ck}, m)$*: Outputs a commitment* cm *and an opening information* op*. This algorithm specifies a function* $\mathsf{Com} : M_{\mathsf{ck}} \times R_{\mathsf{ck}} \to C_{\mathsf{ck}}$*. Given a*

*message $m \in M_{\mathsf{ck}}$, the sender picks a randomness $\rho \in R_{\mathsf{ck}}$ and computes the commitment* $(\mathsf{cm}, \mathsf{op}) = \mathsf{Com}(\mathsf{ck}, m)$.

- ***Opening Phase,*** $0/1 \leftarrow \mathsf{OpenVerify}(\mathsf{ck}, \mathsf{cm}, m, \mathsf{op})$*: Outputs* $1$ *if the value* $m \in M_{\mathsf{ck}}$ *is the committed message in the commitment* $\mathsf{cm}$ *and* $0$ *if* $(m, \mathsf{op}, \mathsf{cm})$ *does not correspond to a valid opening.*

The commitment scheme $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{OpenVerify})$ is secure if it satisfies the following properties:

**Correctness.** Let $\mathsf{ck} \leftarrow \mathsf{KGen}(1^n)$. Any honestly generated commitment of $m \in M_{\mathsf{ck}}$ as $(\mathsf{cm}, \mathsf{op}) \leftarrow \mathsf{Com}(\mathsf{ck}, m)$, will successfully pass the verification by $\mathsf{OpenVerify}(\mathsf{ck}, \mathsf{cm}, m, \mathsf{op})$.

**Hiding.** For any adversary $\mathcal{A}$, it is difficult to generate two messages $m_0, m_1 \in M_{\mathsf{ck}}$ such that $\mathcal{A}$ can distinguish between their corresponding commitments $\mathsf{cm}_0$ and $\mathsf{cm}_1$ where $(\mathsf{cm}_0, \mathsf{op}_0) \leftarrow \mathsf{Com}(\mathsf{ck}, m_0)$ and $(\mathsf{cm}_1, \mathsf{op}_1) \leftarrow \mathsf{Com}(\mathsf{ck}, m_1)$. In the other term, the probability of winning the game depicted below for the adversary is $\mathsf{negl}(n)$. Meaning that the advantage of adversary $\mathcal{A}$ in the following game is negligible:

$$\Pr\begin{bmatrix}\mathsf{ck} \leftarrow_{\$} \mathsf{KGen}(1^n), (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ck}), b \leftarrow_{\$} \{0,1\}, \\ (\mathsf{cm}_b, \mathsf{op}_b) \leftarrow \mathsf{Com}(\mathsf{ck}, m_b), b' \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{cm}_b) : b = b'\end{bmatrix} = \frac{1}{2} + \mathsf{negl}(n)\,.$$

where $b \leftarrow_{\$} \{0,1\}$ denotes uniformly sampling a bit.

**Binding.** It is computationally hard, for any adversary $\mathcal{A}$, to come up with a collision $(\mathsf{cm}, m_0, \mathsf{op}_0, m_1, \mathsf{op}_1)$, such that $\mathsf{op}_0$ and $\mathsf{op}_1$ are valid opening values for two different pre-images $m_0 \neq m_1$ for $\mathsf{cm}$. Formally speaking, the success probability of polynomial time $\mathcal{A}$ is negligible in the following game,

$$\Pr\begin{bmatrix}\mathsf{ck} \leftarrow_{\$} \mathsf{KGen}(1^n), (\mathsf{cm}, (m_0, \mathsf{op}_0), (m_1, \mathsf{op}_1)) \leftarrow \mathcal{A}(\mathsf{ck}) : (m_0 \neq m_1) \wedge \\ (\mathsf{OpenVerify}(\mathsf{ck}, \mathsf{cm}, m_0, \mathsf{op}_0) = 1) \wedge (\mathsf{OpenVerify}(\mathsf{ck}, \mathsf{cm}, m_1, \mathsf{op}_1) = 1)\end{bmatrix} = \mathsf{negl}(n)\,.$$

In the later sections, we will see that Zerocash instantiates commitment schemes with some collisions resistant hash functions.

### 2.1.1 Pedersen Commitment Scheme.

One of the most known and widely used commitment schemes is Pedersen commitment scheme [Ped91], that guarantees computationally binding and perfectly hiding. The binding property of the scheme relies on the discrete logarithm problem. The definition of this commitment scheme is summarized below.

**Definition 2** (Pedersen Commitment)**.** *Pedersen commitment scheme is a tuple of efficient algorithms as* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{OpenVerify})$ *such that,*

- ***Key Generation Phase,*** $\mathsf{ck} \leftarrow \mathsf{KGen}(1^n)$*: Generates the description of a group* $\mathbb{G}$ *of prime order* $p$ *together with two generators* $(g, h) \leftarrow \mathbb{G}^2$*. It specifies a message space* $M_{\mathsf{ck}}$*, a randomness space* $R_{\mathsf{ck}}$*, and a commitment space* $C_{\mathsf{ck}}$*. It returns* $\mathsf{ck} := (\mathbb{G}, p, g, h)$*;*

- ***Committing Phase***, $(\mathsf{cm}, \mathsf{op}) \leftarrow \mathsf{Com}(\mathsf{ck}, m)$*: Let* $\mathbb{Z}_p$ *be a multiplicative group of integers modulo* $p$*, where p is a prime number. Given* $\mathsf{ck}$ *and the message* $m \in \mathbb{Z}_p$*, it samples a randomness* $r \leftarrow_{\$} \mathbb{Z}_p$ *and computes and returns the commitment* $(\mathsf{cm}, \mathsf{op}) := (g^m h^r, r)$.

- ***Opening Phase***, $0/1 \leftarrow \mathsf{OpenVerify}(\mathsf{ck}, \mathsf{cm}, m, \mathsf{op})$*: Outputs* $1$ *if the value* $m \in M_{\mathsf{ck}}$ *is the committed message in the commitment* $\mathsf{cm}$ *by checking* $g^m h^r = c$*; otherwise returns* $0$.

## 2.2 Key-Private Public-Key Encryption

Public-key encryption is one of the key concepts in cryptography that allows two parties to communicate securely on a public channel without sharing a secret key. A standard public-key encryption scheme is a set of three algorithms $\Pi_{\mathsf{Enc}} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ that are defined as follows,

- **Key Generation Phase**, $(\mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}) \leftarrow \mathsf{KGen}_{\mathsf{Enc}}(1^n, \mathsf{pp}_{\mathsf{Enc}})$: Given public parameters $\mathsf{pp}_{\mathsf{Enc}}$, $\mathsf{KGen}_{\mathsf{Enc}}$ generates a public key and a secret key for a user.

- **Encryption Phase**, $c \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{Enc}}, m)$: Given a public key $\mathsf{pk}_{\mathsf{Enc}}$ and a message $m$, algorithm $\mathsf{Enc}$ encrypts $m$ and outputs a ciphertext $c$.

- **Decryption Phase**, $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{Enc}}, c)$: Given a secret key $\mathsf{sk}_{\mathsf{Enc}}$ and a ciphertext $c$, algorithm $\mathsf{Dec}$ decrypts ciphertext $c$ an outputs the message $m$ (or $\perp$ if decryption fails).

Zerocash uses a particular type of public key encryption scheme that is called *key-private public-key encryption* (proposed by Bellare et al. [BBDP01]) and provides slightly more security than the a standard public key encryption scheme. A *key-private public-key encryption* scheme $\Pi_{\mathsf{Enc}}$ is required to satisfy two security requirements. First, ciphertext indistinguishability under chosen-ciphertext attack (IND-CCA security), and second key indistinguishability under chosen-ciphertext attack (IK-CCA security). However the first requirement is quite known about encryption schemes, but the second one is defined particularly for a *key-private public-key encryption* scheme and intuitively an IK-CCA notion means that it is not possible to link the encrypted message either to the public key which used for encryption, or other encrypted messages using this public key. Precise definitions of these two properties are explained in the rest of the subsection.

### 2.2.1 IND-CCA

IND-CCA stands for Indistinguishability Under Chosen Ciphertext Attack and the formal definition is as follows,

**Definition 3** (IND-CCA security)**.** *A public key encryption scheme* $\Pi_{\mathsf{Enc}} = (\mathsf{KGen}_{\mathsf{Enc}}, \mathsf{Enc}, \mathsf{Dec})$*, consisting of a key-generation algorithm* $\mathsf{KGen}$*, an encryption algorithm* $\mathsf{Enc}$*, and a decryption algorithm* $\mathsf{Dec}$*, is IND-CCA (indistinguishable under chosen ciphertext attacks) if for any polynomial time algorithm* $\mathcal{A}$*, we have that*

$$\Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{Enc}}(1^n), (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk}, \cdot)}(\mathsf{pk}), b \leftarrow_{\$} \{0,1\}, \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}(c) : b = b'\end{bmatrix} = \frac{1}{2} + \mathsf{negl}(n)\,.$$

*where* $c$ *is a some arbitrary information that* $\mathcal{A}$ *wants to send.*

### 2.2.2 IK-CCA

IK-CCA stands for Key Indistinguishable Under Chosen Ciphertext Attack and the formal definition is as follows,

**Definition 4** (IK-CCA Security)**.** *Let* $\Pi_{\mathsf{Enc}} = (\mathsf{KGen}_{\mathsf{Enc}}, \mathsf{Enc}, \mathsf{Dec})$ *be an encryption scheme. Let* $\mathcal{A}$ *be an adversary which has access to the oracles* $\mathsf{Dec}(\mathsf{sk}_0, \cdot)$ *and* $\mathsf{Dec}(\mathsf{sk}_1, \cdot)$*. By considering this, an encryption scheme* $\Pi_{\mathsf{Enc}}$ *is said to be IK-CCA secure if the success probability of any polynomial time adversary* $\mathcal{A}$ *is negligible in the following game,*

$$\Pr\begin{bmatrix}(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}_{\mathsf{Enc}}(1^n), (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KGen}(1^n), \\ (x, s) \leftarrow \mathcal{A}^{\mathsf{Dec}_{\mathsf{sk}_0}(.), \mathsf{Dec}_{\mathsf{sk}_1}(.)}(\mathsf{pk}_0, \mathsf{pk}_1), b \leftarrow_{\$} \{0,1\}, \\ y \leftarrow \mathsf{Enc}_{\mathsf{pk}_b}(x), b' \leftarrow \mathcal{A}(y, s) : b = b'\end{bmatrix} = \frac{1}{2} + \mathsf{negl}(n)\,.$$

*where* $s$ *is a some arbitrary information that* $\mathcal{A}$ *wants to send.*

## 2.3 Digital Signatures

Digital signature is a cryptography tool for providing data integrity and non-malleability. A digital signature is used to authenticate the real sender of a message on a public channel. In a public-key based digital signature scheme, there is a pair of public and secret key, where the secret key is used to sign a message by a sender, and receiver can use the public key to verify the validity of a given message-signature pair.

### 2.3.1 One-Time Secure Digital Signatures

The standard security requirement for a signature scheme is unforgeability which is defined as "strong unforgeability" property of signatures meaning that, there is no bounded adversary who can forge a valid signature without knowing the secret key. Formal definition and strong unforgeability of digital signatures and property of this schemes are described bellow.

**Definition 5** (Digital Signature)**.** *A digital signature scheme* $\Pi_\Sigma$ *is a tuple of algorithms as* $\Pi_\Sigma = (\mathsf{KGen}_{\mathsf{Sign}}, \mathsf{Sign}, \mathsf{Vf})$ *working as follows:*

- ***Key Generation Phase****,* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^n)$*: given the security parameter* $1^n$ *as input to the key generation, it will produce a pair of keys* $(\mathsf{sk}, \mathsf{vk})$ *where* $\mathsf{sk}$ *is the secret key for signing the message and* $\mathsf{vk}$ *is the public key for verifying a message-signature pair.*
- ***Signing Phase****,* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$*: given a secret key* $\mathsf{sk}$ *and a message* $m$ *to the the signing algorithm* $\mathsf{Sign}$ *as input, the output will be a signature* $\sigma$*.*
- ***Verification Phase****,* $0/1 \leftarrow \mathsf{Vf}(\mathsf{vk}, m, \sigma)$*: given a triple* $\mathsf{vk}, m, \sigma$ *to the verification algorithm* $\mathsf{Vf}$*, it will output* $1$*, checking with verification key* $\mathsf{vk}$*, if* $\sigma$ *is the valid signature of the massage* $m$*, otherwise it will output* $0$*.*

**UF-CMA Security.** As mentioned above an essential security requirement from a signature scheme $\Pi_\Sigma$ is unforgeability against chosen-message attacks (UF-CMA security) which its formal definition can be expressed as follows.

**Definition 6** (UF-CMA security)**.** *A digital signature scheme $\Pi_\Sigma$ is UF-CMA-secure if the probability of winning for any forger $F$ running in time $\mathsf{poly}(n)$ and making $Q = \mathsf{poly}(n)$ signing queries, in the following game is negligible:*

$$\mathsf{Adv}^{\mathrm{uf-cma}}_{\Pi_\Sigma,F}(n) = \Pr\left[\begin{matrix}(\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^n), (m^*,\sigma^*) \leftarrow F^{\mathsf{Sign}(\mathsf{vk},\cdot)}(\mathsf{pk}) : \\ \mathsf{Vf}(\mathsf{vk},m^*,\sigma^*) = 1 \wedge (m^* \textit{ is "new"})\end{matrix}\right] = \mathsf{negl}(n)\,.$$

*As a strengthened version, strong unforgeability against chosen-message attacks (SUF-CMA) is a bit stronger than just UF-CMA. In SUF-CMA signatures there is a checking for freshness of both message and signature. In the other words, the verification algorithm checks whether the pair $(m^*,\sigma^*)$ is "new", i.e. if $(m^*,\sigma^*)$ is different from all the pairs $(m_i,\sigma_i)$ obtained by $F$ from the signing oracle.*

Zerocash uses a one-time strong unforgeability against chosen-message attacks (SUF-1CMA security) signature scheme to prevent malleability attacks on transactions. SUF-1CMA security is a particular case of SUF-CMA which the scheme only is secure for one time using.

### 2.3.2 Linkable Ring Signatures

Linkable ring signatures are one type of digital signatures that allows anonymous signing and is proposed by Rivest et al. [**?**] in 2001. Later we will observe that linkable ring signatures are the main primitives in Monero cryptocurrency to achieve unlinkability in transactions. In a short comparison, common cryptographic digital signatures allow to link transactions to their respective senders and receivers. But a protocol based on linkable ring signatures allows users to achieve unlinkability. A linkable ring signature can be formally defined as the following.

**Definition 7** (Linkable Ring Signatures)**.** *A linkable ring signature consists of four algorithms ($\mathsf{KGen}_{\mathsf{Sign}}, \mathsf{Sign}, \mathsf{Vf}, \mathsf{Link}$) defined as,*

- ***Key Generation Phase**, $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{KGen}_{\mathsf{Sign}}(1^n)$: given the security parameter $1^n$ as input to the key generation, it will produce a pair of keys $(\mathsf{sk},\mathsf{vk})$ where $\mathsf{sk}$ is the secret key for signing the message and $\mathsf{vk}$ is the public key for verifying a signature.*
- ***Signing Phase**, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, T, m)$: given a secret key $\mathsf{sk}_i$ where $i \in N$, tag $T = (issue, \mathsf{vk}_N)$ and message $m \in \{0,1\}^*$ to the signing algorithm $\mathsf{Sign}$ as input, it outputs a signature $\sigma$.*
- ***Verification Phase**, $0/1 \leftarrow \mathsf{Vf}(\mathsf{vk}, T, m, \sigma)$: given tag $T = (issue, \mathsf{vk}_N)$, message $m \in \{0,1\}^*$ and a signature $\sigma$ to the verification algorithm $\mathsf{Vf}$, it will output $1$, checking with verification key $\mathsf{vk}$, if $\sigma$ is the valid signature of the massage $m$, otherwise it will output $0$.*
- ***Link Phase**, "indep", "linked", or $\mathsf{vk} \leftarrow \mathsf{Link}(\mathsf{vk}, T, \{(m,\sigma),(m',\sigma')\})$: given tag $T = (issue, \mathsf{vk}_N)$, two message/signature pairs, $\{(m,\sigma),(m',\sigma')\}$ to the link algorithm $\mathsf{Link}$, it will output one of the following strings: "indep" if they $\sigma$ and $\sigma'$ are from different parties of the group; "linked" if $m = m'$; or $\mathsf{vk}$ otherwise, where $\mathsf{vk} \in \mathsf{vk}_N$.*

Generally, a linkable ring signature satisfies the following security properties [FS07].

- **Public Traceability:** Anyone who creates two signatures for different messages w.r.t. the same tag can be traced, where the trace can be done only with pairs of message/signature pairs and the tag.
- **Tag-Linkability:** Every two signatures generated by the same signer w.r.t. the same tag are linked, that is, the total number of signatures w.r.t. the same tag cannot exceed the total number of ring members in the tag, if any two signatures are not linked.
- **Anonymity:** As long as a signer does not sign on two different messages w.r.t. the same tag, the identity of the signer is indistinguishable from any of the possible ring members. Additionally, any two signatures generated with respect to two distinct tags are always unlinkable. Namely, it is infeasible for anyone to determine whether they are generated by the same signer.
- **Exculpability** A honest ring member cannot be accused of signing twice w.r.t. the same tag — Namely, an adversary cannot produce a traceable ring signature such that, along with one generated by the target, it can designate the target member in the presence of the publicly traceable mechanism. This should be infeasible even after the attacker has corrupted all ring members but the target.

## 2.4 One-Way Functions

One-way functions (OWFs) are another prominent cryptographic primitives that is used in construction of Zerocash. A OWF is a function that can be computed efficiently, but, as it can be guessed from the word one-way, finding any pre-image of a random image can not be done efficiently (finding pre-image means inverting). The formal definition of one-way functions is as follows,

**Definition 8** (One-Way Function)**.** *A function $f$: $\{0,1\}^* \to \{0,1\}^*$ is a one-way function if,*

- ***Efficient**: $f$ can be evaluated in polynomial time*
- ***One-Way**: For every polynomial time adversary $\mathcal{A}$, the following probability is negligible in security parameter $n$:*

$$\Pr[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x)) : x \leftarrow_{\$} \{0,1\}^n] = \mathsf{negl}(n)\,.$$

We will later see that Zerocash instantiates OWFs with hash functions. So in the rest, we recall definition of Universal One-Way Hash Functions that are closely related to OWFs.

### 2.4.1 Universal One-Way Hash Functions

Universal one-way hash functions are a flavour of cryptographic one-way functions that have appeared in various applications.

**Definition 9** (Universal One-Way Hash Family)**.** *A collection of function families $\mathcal{H} = \{\mathsf{H}_n\}$ where each $\mathsf{H}_n$ is a function family $\mathsf{H}_n = \{h : \{0,1\}^{l(n)} \to \{0,1\}^{q(n)}\}$ is an universal one-way hash family if:*

- ***Efficient**: The functions $q(n)$ and $\ell(n)$ are polynomially-bounded; furthermore, given $n$ and $x \in \{0,1\}^{q(n)}$ the value $h(x)$ can be computed in polynomial time.*
- ***Compressing**: For all $n$ we have that $q(n) > \ell(n)$.*
- ***Universal one-way**: For all polynomial time adversary $\mathcal{A}$, the following probability is negligible in the security parameter $n$:*

$$\Pr\left[\begin{array}{l} x \leftarrow \mathcal{A}(1^n), h \leftarrow_{\$} \mathsf{H}_n, x' \leftarrow \mathcal{A}(1^n, h, x) : \\ x, x' \in \{0,1\}^{q(n)} \wedge\ x \neq x' \wedge\ h(x) = h(x') \end{array}\right] = \mathsf{negl}(n)\,.$$

### 2.4.2 Pseudo-Random Functions

A pseudo-random function family, abbreviated PRF, is a collection of efficiently-computable functions which emulate a random oracle in the following way: no efficient algorithm can distinguish (with significant advantage) between a function chosen randomly from the PRF family and a random oracle (a function whose outputs are fixed completely at random). It can be formally defined as below.

**Definition 10.** *A function $f : K \times M \to N$ (where $f, K, M$, and $N$ may depend on the security parameter $n$) is a $(\epsilon, q, \tau)$-pseudo-random function (PRF) if for all $\tau$-time algorithms $A$ that make at most $q$ queries we have that,*

$$\Pr\left[b^* = 1 : k \leftarrow_{\$} K, b^* \leftarrow \mathcal{A}^{f(k,\cdot)}()\ \ \right] \approx_c \Pr\left[b^* = 1 : \pi \leftarrow_{\$} Fun_{M\to N}, b^* \leftarrow \mathcal{A}^{\pi}()\ \ \right]$$

*where $Fun_{M\to N}$ denotes the set of all functions from $M$ to $N$.*

## 2.5 Nakamoto Consensus and Proof-of-Work

Consensus protocols are essential to check the authenticity of distributed ledgers and making agreement among nodes of a a distributed network. Strictly speaking, in Bitcoin, in order to extend the ledger and add a new block, nodes of the network need to participate in a consensus protocol that is known as *Nakamoto consensus protocol*. Nakamoto Consensus protocol, that is proposed by Satoshi Nakomoto, allude to the Proof-of-Work (PoW) consensus model in conjunction with a set of rules in Bitcoin network, that govern the procedure of consensus and guarantees its security. Bitcoin is the first practical open and distributed peer-to-peer network that uses a distributed network of anonymous nodes that can leave or join at will.

Briefly speaking, Nakamoto consensus protocol can be expressed in four blocks as follows:

- **Proof-of-Work (PoW)**: In comparison with other parts, PoW is the most important part that drives consensus in Bitcoin. In a PoW work system, a particular *work* (or puzzle) is given to a user and then is asked to give a *proof* that the work is done (provide a solution). Similarly, in Bitcoin a miner uses a specific node to compete in a PoW competition to mine new blocks and earn the block reward, that is issued for each successfully mined and validated block. Contributing in the competition of mining is based on node's computational power, as nodes need to solve a puzzle which needs huge computations. Precisely speaking, nodes are given a task that "For given *random challenge* $c$ find a $nonce$ such that first $x$ bits of $\mathsf{Hash}(c \,||\, nonce)$ are zero bits." So, the more power a node assigns

to the competition, the more likely she will be selected to mine the next block. One may note that the process of PoW is stochastic, so it basically is sort of a lottery with random chance of wining, so it is not determined who will win the next round. The longest chain is considered the valid chain as it came from the pool with largest computational power.

We note that given $c$ and $nonce$, verifying whether $\mathsf{Hash}(c \,||\, nonce) = \mathsf{target\ value}$ can be done efficiently, as it is the case in verification of PoW puzzle's solution. The validation ensures that proposed blocks have the requisite computational work performed to be accepted and as long as honest parties controls majority of the network's computational power the main ledger will remain secure.

- **Block Selection**: In the PoW, the miners in the network all compete to solve the given puzzle and the first one who managed to solve it wins the round of the lottery. In other words, the result of PoW is that when puzzle is solved by a particular miner, the miner proposes next block to the network, and then the network verifies the validity of the proposed block before adding to the ledger. If all the transactions within the proposed block are valid, and the block is added to the longest chain, the proposed block will be added to the ledger.

- **Scarcity**: Traditionally precious metals were the main form of value storage and commerce. Two main reason that they were used, and still keep their value (e.g. gold), are that they are scarce and also require so much effort (similar to PoW in Bitcoin) to mine and then use them.

  In Bitcoin, scarcity is based on the fact that the total number of Bitcoins that will be mined will be restricted to 21 million. More importantly, new Bitcoin are inserted into the system by geometric rewinding, meaning that the reward for mining a block reward is halved every 210,000 blocks which by considering the fact that currently each block of Bitcoin is constructed in 10 minutes, and every 4 years the block reward will be divided by 2.

- **Incentive Structure**: Having a incentive structure in a digital coin allows to long-term vested owners and participants in the network of coin to further secure and validate the network of coin. Block rewarding is the strategy that Bitcoin uses to incentivise miners to validate and support the network. By this strategy, in the case of drop in value of Bitcoin or other related concerns, they also will be affected.

## 2.6 Decentralized Anonymous Payment (DAP) Schemes

In order to construct Zerocash, initially Ben Sasson et al. defined the notion of *Decentralized Anonymous Payment scheme* (DAP scheme) that is an extended version of the notion of *decentralized e-cash* proposed in [MGGR13]. In this section, we review their DAP scheme and later show that Zerocash is a construction for this notion. Initially, we review data structures deployed in the scheme and then introduce its algorithms and properties.

### 2.6.1 Data Structures

Later we will observe that their proposed DAP scheme (denoted by $\mathsf{DAP}$) uses the following data structures.

**Basecoin ledger.** The DAP scheme is constructed to act on top of a ledger-based digital coin, e.g. Bitcoin, which is called *basecoin*. At time $T$, all nodes have access to an append-only ledger $L_T$. Extra from provided transactions in basecoin (precisely Bitcoin), new DAP scheme allows two new type of transactions called $\mathsf{Tran}_{\mathsf{Mint}}$ and $\mathsf{Tran}_{\mathsf{Pour}}$ that will be introduced later.

**Public parameters.** The DAP scheme has a list of public parameters $\mathsf{pp}$ that are generated once by a trusted third party and shared with all users (nodes). Later we will see that these are some public parameters for the deployed primitives (e.g. signature scheme, encryption scheme and zero-knowledge proof system).

**Addresses.** In the DAP scheme, each user generates two address keys as public address key $\mathsf{addr}_{\mathsf{pk}}$ and secret address key $\mathsf{addr}_{\mathsf{sk}}$. To do a direct payment one needs $\mathsf{addr}_{\mathsf{pk}}$ of the receiver and to receive a payment, the $\mathsf{addr}_{\mathsf{sk}}$ corresponding to the $\mathsf{addr}_{\mathsf{pk}}$ is required. Each node requires to generate at least one key pair.

**Coins.** In the DAP scheme, each coin is shown with $c$ that contains following four piece of information. 1) Coin commitment $\mathsf{cm}$: In the minting process of a coin, a string will be produced which is called coin commitment, denoted with $\mathsf{cm}$, and will appear on the ledger. 2) Coin value $v$: Each coin has a positive value which is an integer less equal than $\mathsf{V}_{\max}$ (i.e. $0 \leq \mathsf{V} \leq \mathsf{V}_{\max}$, where $\mathsf{V}_{\max}$ is maximum possible coin value allowed in the scheme). 3) Coin serial number $\mathsf{sn}$: To prevent double spending of coins, each coin has a unique serial number that will be determined in (minting) coin $c$ and will be revealed when the coin is spent. 4) Coin address $\mathsf{addr}_{\mathsf{pk}}$: Each coin has public address which belongs to the owner.

**New transactions.** As briefly mentioned above, extra from Bitcoin, new DAP scheme provides two more type of transactions that are called $\mathsf{Tran}_{\mathsf{Mint}}$ and $\mathsf{Tran}_{\mathsf{Pour}}$. 1) Mint transaction, $\mathsf{Tran}_{\mathsf{Mint}}$: A mint transaction is required in the case of minitng new coin and is defined as a tuple $(\mathsf{cm}, v, *)$, where $\mathsf{cm}$ is a coin commitment, $v$ is the value of a coin, and $*$ is some extra information related to the transaction. Each transaction $\mathsf{Tran}_{\mathsf{Mint}}$ shows that a coin $c$ with the coin commitment $\mathsf{cm}$ and the value $v$ has been minted. 2) Pour transaction, $\mathsf{Tran}_{\mathsf{Pour}}$: A pour transaction is required in anonymous payments and is defined a tuple $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info}, *)$, where $\mathsf{rt}$ is a root of the Merkle tree containing all coin commitments, $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$ are serial numbers of two coins that are going to be spent, $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$ are commitments of two new coins that are going to be created, $\mathsf{V}_{pub}$ is a coin value which is public, $\mathsf{info}$ is a string to determine the destination of $\mathsf{V}_{pub}$, and $*$ is extra information used in implementation. Each transaction $\mathsf{Tran}_{\mathsf{Pour}}$ takes two old coins $c_1^{\mathsf{old}}$ and $c_2^{\mathsf{old}}$ with the serial numbers $\mathsf{sn}_1^{\mathsf{old}}$, $\mathsf{sn}_2^{\mathsf{old}}$ respectively and mints (produces) two new coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ with respective coin commitments $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$. Note that $\mathsf{V}_{pub}$ is considered to pay transaction fee and could be zero as well. The $\mathsf{info}$ string determines who is the recipient of $\mathsf{V}_{pub}$, and mentioned above $\mathsf{rt}$ is the root of the Merkle tree over all coin commitments when the transaction was produced.

**Commitments of minted coins ($\mathsf{CMList}_T$) and serial numbers of spent coins ($\mathsf{SNList}_T$).** In the DAP scheme, at each time $T$, there are two universal lists which are shared with all nodes and updated frequently. First one is $\mathsf{CMList}_T$ that contains the list of all coin commitments which are created after each *mint* or *pour* transactions and appeared on the ledger. Second one is $\mathsf{SNList}_T$ that contains serial numbers of all the spent coins by time $T$ (that appear on the ledger).

**Merkle tree over coin commitments.** Recall that a Merkle tree [Mer] is a tree in which every leaf node is labelled with the hash of a data block and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. A Merkle tree allows efficient

and secure verification of the contents of large data structures. A graphical representation of a sample Merkle tree is shown in Fig.1. In the DAP scheme, in each pour transaction, to spend a coin, a spender needs to prove that she knows the secret information about the target coin. In order to make that procedure efficient, at any time $T$, $\mathsf{Tree}_T$ shows a Merkle tree over all coin commitments in $\mathsf{CMList}_T$ and $\mathsf{rt}_T$ shows its root. The function $\mathsf{Path}_T(\mathsf{cm})$ shows the authentication path from a coin commitment $\mathsf{cm}$ recorded in $\mathsf{CMList}_T$ to the root $\mathsf{rt}_T$.

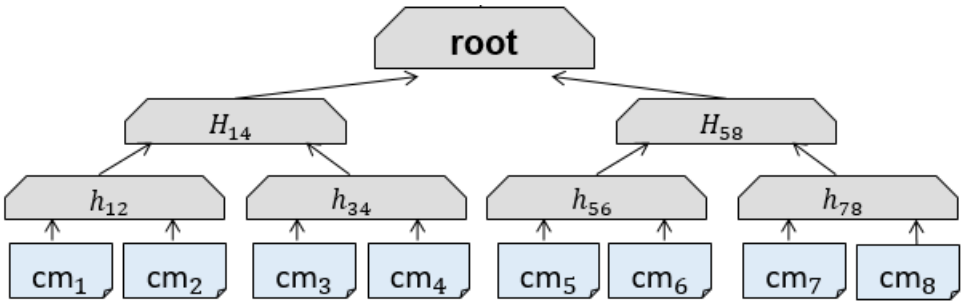


Figure 1. A Merkle tree over commitments $\mathsf{cm}_i$ for $i = 1, 2, \ldots, 8$. Each inner node of the tree is the hash value of the concatenation of its two children.

### 2.6.2 Definition of Algorithms

A DAP scheme $\Pi_{\mathsf{DAP}}$ is a tuple of polynomial time algorithms $\Pi_{\mathsf{DAP}} = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ which are defined as below,

- **Setup Algorithm**, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^n)$: Given a security parameter $n$ generates public parameter $\mathsf{pp}$ needed in other algorithms. This algorithm is executed only once by a trusted third party or by some distributed authorities.

- **Create Address**, $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}) \leftarrow \mathsf{CreateAddress}(\mathsf{pp})$: Given the public parameters $\mathsf{pp}$, it generates a key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$, where $\mathsf{addr}_{\mathsf{pk}}$ is public address and $\mathsf{addr}_{\mathsf{sk}}$ is the corresponding secret address. In the DAP scheme, the sender sends coins to the $\mathsf{addr}_{\mathsf{pk}}$ of receiver and receiver used $\mathsf{addr}_{\mathsf{sk}}$ to redeem coins sent to $\mathsf{addr}_{\mathsf{pk}}$.

- **Minting Coin**, $(c, \mathsf{Tran}_{\mathsf{Mint}}) \leftarrow \mathsf{Mint}(\mathsf{pp}, v, \mathsf{addr}_{\mathsf{pk}})$: This algorithm is used to create a new coin with a special value. Given the public parameters $\mathsf{pp}$, coin value $v \in [0, \ldots, \mathsf{V}_{\max}]$, and receiver's $\mathsf{addr}_{\mathsf{pk}}$ it produces a coin $c$ and a *mint transaction* $\mathsf{Tran}_{\mathsf{Mint}}$ to record the coin $c$ in the ledger. The mint transaction $\mathsf{Tran}_{\mathsf{Mint}}$ consists of $(\mathsf{cm}, v, *)$, where $\mathsf{cm}$ is the coin commitment, $v$ is the value of coin and $*$ some extra information required to register on the ledger (e.g. randomnesses used in generating $\mathsf{cm}$).

- **Pour (Spending) Algorithm**, $(c_1^{\mathsf{new}}, c_2^{\mathsf{new}}, \mathsf{Tran}_{\mathsf{Pour}}) \leftarrow \mathsf{Pour}(\mathsf{pp}, \mathsf{rt}, c_1^{\mathsf{old}}, c_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathsf{Path}_1, \mathsf{Path}_2, \mathsf{V}_1^{\mathsf{new}}, \mathsf{V}_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info})$: This algorithm is used to spend two old coins and produce two new coins along with a pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$ to record the procedure on the ledger. To do so, given public parameters $\mathsf{pp}$, the Merkle root $\mathsf{rt}$, old coins $c_1^{\mathsf{old}}$ and $c_2^{\mathsf{old}}$, old address secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}$ and $\mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$, path $\mathsf{Path}_1$ from commitment $\mathsf{cm}(c_1^{\mathsf{old}})$ to the root $\mathsf{rt}$ and path $\mathsf{Path}_2$ from commitment $\mathsf{cm}(c_2^{\mathsf{old}})$ to the root $\mathsf{rt}$, new values $\mathsf{V}_1^{\mathsf{new}}$ and $\mathsf{V}_2^{\mathsf{new}}$, new addresses public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$, public

value $\mathsf{V}_{pub}$, and transaction string info, it produces two new coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ for the public address keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{sk},2}^{\mathsf{new}}$, respectively. It also outputs some public value which can be spent as the transaction fee.

This algorithm allow to merge coins, subdivide coins into smaller denominations and also do public payments. To check the validity of spent coins, the algorithm takes the root $\mathsf{rt}$ of the Merkle tree over all the coin commitments and verifies the authentication paths $\mathsf{Path}_1$ and $\mathsf{Path}_2$ which should go to two spent coins. The algorithm also takes values $v_1^{\mathsf{new}}$ and $v_2^{\mathsf{new}}$ as values of two new anonymous coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ with the corresponding address public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$. The input $\mathsf{V}_{pub}$ shows the amount which is publicly spent, for instance for transaction fees. The summation of the new coins' value plus the $\mathsf{V}_{pub}$ must be exactly equal to the the addition of the old coins' value. The input $\mathsf{info}$ is an arbitrary string related to destination of $\mathsf{V}_{pub}$.

The output of algorithm consists of two new coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ and a pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$. The transaction $\mathsf{Tran}_{\mathsf{Pour}}$ contains the root $\mathsf{rt}$, two old coins' serial numbers $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$, two commitments' of new coins $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$, the public value $\mathsf{V}_{pub}$, a arbitrary string $\mathsf{info}$ and $*$, where $*$ is some related information (e.g. randomnesses).

- **Verifying Transactions**, $\{0,1\} \leftarrow \mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{Tran}, L)$: This algorithm is used to check if a transaction is valid. Given the public parameters $\mathsf{pp}$, a mint or pour transaction $\mathsf{Tran}$ and the current ledger $L$, it returns either 1 (if the transaction is valid) or 0 (if the transaction is invalid). In practice, this algorithm is executed by the nodes of the distributed network to verify the transactions.

- **Receiving Transactions**, $\{\text{a set of received coins}\} \leftarrow \mathsf{Receive}(\mathsf{pp}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}, \mathsf{L})$: This algorithm is used to receive the coins sent to a particular public address. More precisely, given public parameters $\mathsf{pp}$, an address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ and the current ledger $\mathsf{L}$, it scans the ledger and returns the set of coins sent to the (target) public address $\mathsf{addr}_{\mathsf{pk}}$. The algorithms returns the corresponding coins that their serial number have not published on the list of serial numbers $\mathsf{SNList}$.

### 2.6.3 Requirements in a DAP Scheme

Based on the definition of Ben Sasson et al. [BCG+14a], a DAP scheme should guarantee *completeness*, *ledger indistinguishability*, *transaction non-malleability*, and *balance* that we briefly introduce in the rest (in later sections we will consider these notions more formally):

**Completeness of a** DAP **scheme.** Intuitively the completeness of a DAP scheme requires that unspent coins can be spent. Specifically, result of running algorithm Pour is a pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$ which VerifyTransaction accepts it, and the target recipients can receive the desired coins by running the algorithm Receive; additionally, $\mathsf{Tran}_{\mathsf{Pour}}$ correctly records the intended $\mathsf{V}_{pub}$ and transaction string $\mathsf{info}$ on the ledger.

**Ledger indistinguishability in a DAP scheme.** This requirement satisfies the need for having completely zero knowledge ledger which means that the ledger doeas not leak any information unless the public parameters including values of minted coins, public values, information strings, total number of transactions, etc. Technically speaking, the advantage of adversary $\mathcal{A}$ to distinguish between two ledgers $L_0$ and $L_1$ which are constructed by $\mathcal{A}$ is negligible.

More precisely, adversary several times queried to two DAP scheme oracles and constructed two ledgers $L_0$ and $L_1$. The queries should satisfy the publicly consistent requirement which means that they should have the same type, public information and information about addresses controlled by $\mathcal{A}$.

**Transaction non-malleability.** This requirement prevents malleability attacks such as retargeting the basecoin which are public on the pour transactions which before recording into the ledger. More technically, the satisfaction of this requirement means that the advantage of the adversary $\mathcal{A}$ to change the valid pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$ is negligible.

**Balance.** The advantage of the adversary $\mathcal{A}$ to have more coins than he minted or received is negligible.

## 2.7 Non-Intractive Zero-Knowledge (NIZK) Proofs

Zero Knowledge (ZK) proofs is a cryptographic tools that has recently appeared in various applications for different reasons including users privacy or efficient verification. A ZK proof allows a prover to convince a verifier that some statement is correct without revealing any information except the truth of the statement. For example, a prover can prove that she lives in a European country without revealing her personal information or name of her home country. ZK proofs can be either *interactive* or *non-interactive*. Usually non-interactive proofs are more convenient and popular, as they can be generated without verifier's participation and verifier or verifiers can check the proof whenever they are available. In other words, in the real world, any interactivity between prover and verifier needs some communication over a network, which raises some latency issues.

It has been shown that Non-Interactive Zero-Knowledge (NIZK) arguments can be constructed in the Random Oracle (RO) model or Common Reference String (CRS) model. In the RO model, we assume that both prover and verifier have access to a RO, which does not exists and in practice and it is instantiated instead with a collision-resistant hash function.

On the other hand, in the CRS model there is a trusted third party who generates a CRS containing: proving key $\mathsf{pk}$ for prover and verification key $\mathsf{vk}$ for verifier. Prover uses $\mathsf{pk}$ and generates a proof $\pi$ which can be verified by any verifier (in publicly verifiable proofs).

Among various techniques for NIZK proofs, zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) are the most practical and popular one. These types of zero-knowledge proofs are the main primitives in the structure of Zerocash, so in the rest of section, we introduce this type of NIZK arguments with an example that currently is used in Zerocash protocol.

### 2.7.1 zk-SNARKs

As already mentioned zk-SNARKs are the most popular and practical type of NIZK proofs. Indeed, this comes form the fact that they generate succinct proofs which allows very fast verification. Due to high efficiency, beside Zerocash, recently zk-SNARK are found in various applications that aims to guarantee strong privacy (e.g. smart contracts). As in the rest of thesis, we will talk about zk-SNARKs and using them in Zerocash, so in this we recall the definition of zk-SNARKs and then briefly summarize the state-of-the-art zk-SNARK which was proposed by Groth [Gro16] and recently deployed in Zerocash.

**Definition of zk-SNARKs** Initially, we recall the definition of zk-SNARKs for arithmetic circuit satisfiability. For a field $\mathbb{F}$, an $\mathbb{F}$-arithmetic circuit[4] takes some elements in $\mathbb{F}$ as input, and output some elements in $\mathbb{F}$. A circuit is associated with the function that the circuit computes. In Zerocash, they consider the circuits that have an input $\mathsf{x} \in \mathbb{F}^n$ and an auxiliary input or a witness $\mathsf{w} \in \mathbb{F}^h$. It is assumed that the circuits only have bilinear gates. A gate with inputs $y_1, \cdots, y_m \in \mathbb{F}$ is bilinear if the output is $\langle \vec{\mathsf{w}}, (1, y_1, \cdots, y_m) \rangle \; h \cdot \langle \vec{b}, (1, y_1, \cdots, y_m) \rangle$ for some $\vec{\mathsf{w}}, \vec{b} \in \mathbb{F}^{m+1}$, where $\langle \cdot, \cdot \rangle$ shows the inner product. These support addition, multiplication, negation, and constant gates. By considering this, for the boolean case, the arithmetic circuit satisfiability is defined as below,

**Definition 11** (zk-SNARK)**.** *The arithmetic circuit satisfiability problem of an $\mathbb{F}$-arithmetic circuit $\mathsf{C} : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ is captured by the relation $\mathbf{R}_{\mathsf{C}} = \{(\mathsf{x}, \mathsf{w}) \in \mathbb{F}^n \times \mathbb{F}^h : \mathsf{C}(\mathsf{x}, \mathsf{w}) = 0^l\}$; its language is $\mathbf{L}_{\mathsf{C}} = \{\mathsf{x} \in \mathbb{F}^n : \exists \mathsf{w} \in \mathbb{F}^h \; s.t. \; \mathsf{C}(\mathsf{x}, \mathsf{w}) = 0^l\}$. By considering above relation and language, given a field $\mathbb{F}$, a zk-SNARK for $\mathbb{F}$-arithmetic circuit satisfiability is a triple of polynomial-time algorithms $(\mathsf{KGen}, \mathsf{P}, \mathsf{Vf})$, that are defined as below:*

- ***Public Parameter Generation***, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^n, \mathsf{C})$*: Given a security parameter $n$ and an arithmetic circuit $\mathsf{C}$, the CRS generator $\mathsf{KGen}$ samples a CRS which contains proving and verification keys $\mathsf{crs} = (\mathsf{pk}, \mathsf{vk})$ and publishes them. Now these keys, can be used to give/verify a proof for a membership in $\mathbf{L}_{\mathsf{C}}$.*
- ***Proof Generation***, $\pi \leftarrow \mathsf{P}(\mathsf{pk}, \mathsf{x}, \mathsf{w})$*: Given a proving key $\mathsf{pk}$ and any $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}_{\mathsf{C}}$, the algorithm $\mathsf{P}$ generates a succinct NIZK proof $\pi$ for the statement $\mathsf{x} \in \mathbf{L}_{\mathsf{C}}$.*
- ***Proof Verification***, $b \leftarrow \mathsf{Vf}(\mathsf{vk}, \mathsf{x}, \pi)$*: Given a verification key $\mathsf{vk}$, an input $\mathsf{x}$, and a proof $\pi$, the algorithm $\mathsf{Vf}$ returns $b = 1$ if verification passed ($\mathsf{x} \in \mathbf{L}_{\mathsf{C}}$), else returns $0$.*

Generally, a zk-SNARK should guarantee the following properties.

- **Completeness**: For any security parameter $n$, any $\mathbb{F}$-arithmetic circuit $\mathsf{C}$, and any $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}_{\mathsf{C}}$, the honest prover can convince honest verifier. Namely,

$$\Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^n, \mathsf{C}), (\mathsf{x}, \mathsf{w}) \leftarrow \mathsf{P}(\mathsf{pk}, \mathsf{vk}), \\ \pi \leftarrow \mathsf{P}(\mathsf{pk}, \mathsf{x}, \mathsf{w}) : \mathsf{Vf}(\mathsf{vk}, \mathsf{x}, \pi) = 1\end{bmatrix} = 1 - \mathsf{negl}(n)\,.$$

- **Knowledge Soundness**: If verifier $\mathsf{Vf}$ accepts a proof output by a computationally bounded prover $\mathsf{P}$, then the prover *knows* a witness for the given instance. More formally, for every $\mathsf{poly}(n)$-time adversarial $\mathsf{P}$, there exists a $\mathsf{poly}(n)$-time extractor $\mathsf{Ext}$ such that,

$$\Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^n, \mathsf{C}), (\mathsf{x}, \mathsf{w}) \leftarrow \mathsf{P}(\mathsf{pk}, \mathsf{vk}), (\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}; r), \\ \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{pk}, \mathsf{vk}; r) : (\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathsf{C}} \wedge \mathsf{Vf}(\mathsf{vk}, \mathsf{x}, \pi) = 1\end{bmatrix} = \mathsf{negl}(n)\,.$$

- **Perfect Zero-Knowledge**: An honestly-generated proof is perfect zero knowledge, meaning that the proof does not reveal any information about witness $\mathsf{w}$. More formally, there

[4] An $\mathbb{F}$-arithmetic circuit is a circuit that performs arithmetic functions like Addition, Subtraction and Multiplication on elements from field $\mathbb{F}$.

is a polynomial-time simulator $\mathsf{Sim}$, such that for all distinguishers $\mathsf{D}$, the following two probabilities are equal:

$$\Pr\begin{bmatrix}(\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KGen}(1^n,\mathsf{C}), (\mathsf{x},\mathsf{w}) \leftarrow \mathsf{D}(\mathsf{pk},\mathsf{vk}),\\ \pi \leftarrow \mathsf{P}(\mathsf{pk},\mathsf{x},\mathsf{w}) : (\mathsf{x},\mathsf{w}) \in \mathbf{R}_{\mathsf{C}} \wedge \mathsf{D}(\pi) = 1\end{bmatrix}$$

(the probability that $\mathsf{D}(\pi) = 1$ on an honest proof) and

$$\Pr\begin{bmatrix}(\mathsf{pk},\mathsf{vk},\mathsf{tr}) \leftarrow \mathsf{Sim}(1^n,\mathsf{C}), (\mathsf{x},\mathsf{w}) \leftarrow \mathsf{D}(\mathsf{pk},\mathsf{vk}),\\ \pi \leftarrow \mathsf{Sim}(\mathsf{x},\mathsf{tr}) : (\mathsf{x},\mathsf{w}) \in \mathbf{R}_{\mathsf{C}} \wedge \mathsf{D}(\pi) = 1\end{bmatrix}$$

(the probability that $\mathsf{D}(\pi) = 1$ on a simulated proof).

- **Succinctness**: An honestly-generated proof $\pi$ has $\mathsf{O}_n(1)$ bits and $\mathsf{Vf}(\mathsf{vk},\mathsf{x},\pi)$ runs in time $\mathsf{O}_n(|\,\mathsf{x}\,|)$, where $\mathsf{O}_n$ hides a fixed polynomial factor in $n$.

Usually a common NIZK only satisfies, completeness, zero-knowledge and soundness; soundness is a weaker notion than knowledge soundness. Below we define non-black-box simulation knowledge soundness (a.k.a. simulation extractability) that is an amplified version of knowledge soundness that guarantees non-malleability of proofs. Intuitively, simulation extractability guarantees that knowledge soundness holds even if adversary has seen arbitrary number of simulated proofs. This notion can be defined formally as bellow.

**Definition 12** ((Non-Black-Box) Simulation Extractability [GM17] or Simulation-knowledge soundness)**.** *A non-interactive argument $\Psi$ is* (non-black-box) simulation-extractable for $\mathcal{R}$*, if for any NUPPT $\mathcal{A}$, there exists a NUPPT extractor $\mathsf{Ext}_{\mathcal{A}}$ s.t. for all $n$,*

$$\Pr\begin{bmatrix}\mathsf{crs} \leftarrow \mathsf{KGen}(\mathbf{R},\xi), r \leftarrow_r \mathsf{RND}(\mathcal{A}), ((\mathsf{x},\pi)||\mathsf{w}) \leftarrow (\mathcal{A}^{\mathsf{O}(.)}||\mathsf{Ext}_{\mathcal{A}})(\mathbf{R},\xi,\mathsf{crs};r) :\\ (\mathsf{x},\pi) \notin Q \wedge (\mathsf{x},\mathsf{w}) \notin \mathbf{R} \wedge \mathsf{V}(\mathbf{R},\xi,\mathsf{crs}_{\mathsf{V}},\mathsf{x},\pi) = 1\end{bmatrix} \approx_n 0\ ,$$

where, $Q$ is the set of $(\mathsf{x},\pi)$-pairs generated by the adversary's queries to $\mathsf{O}(.)$ and $\mathsf{O}(.)$ generates simulated proofs for arbitrary statements submitted by the adversary. Note that *(non-black-box) simulation extractability* implies *knowledge-soundness*, as the earlier additionally allows the adversary to send query to the proof simulation oracle. Note that in Def. 12 the extractor is non-black-box and to be able to extract the witness, it should have access to the source code and the random coins of $\mathcal{A}$. Later in sec. 4, we present a variation of the stat-of-the-art zk-SNARK that can achieve this notion.

While using a zk-SNARK in Zerocash, *knowledge soundness* and *zero-knowledge* play essential role to achieve privacy and checking the validity of transactions. Later we will discuss with more details that, Zerocash considers a circuit $\mathsf{C}$ for privacy-preserving transactions. The circuit mostly verifies knowledge of SHA256 pre-image as a binding commitment. Knowledge soundness implies that if a prover can give a valid proof for that circuit with a particular statement, so she must *know* the witness (pre-image of SHA256) for the statement. The word *know* is formalized by extracting the witness from the prover.

About the role of zero knowledge in Zerocash, it guarantees that the proof (attached to each transaction) does not reveal any secret information about the spender, transferred values and receiver, beyond the fact that $\mathsf{x} \in \mathbf{L}_{\mathsf{C}}$.

**Remark.** While proving security of Zerocash, authors deal with an extended version of knowledge soundness for the case that a prover outputs a vector pairs of inputs and proofs, $(\vec{\mathsf{x}}, \vec{\pi})$. Consequently, they consider an extractor which extracts a vector $\vec{\mathsf{w}}$. Strictly speaking, if $(\mathsf{KGen}, \mathsf{P}, \mathsf{Vf})$ is a zk-SNARK, then for any adversary $\mathcal{A}$, there exists a extractor $\mathsf{Ext}$ such that,

$$\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KGen}(1^n, \mathsf{C}), (\vec{\mathsf{x}}, \vec{\pi}) \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{vk}), \vec{\mathsf{w}} \leftarrow \mathsf{Ext}(\mathsf{pk},\mathsf{vk}) : \\ \exists\ i\ s.t.\ \mathsf{Vf}(\mathsf{vk}, \mathsf{x}_i, \pi_i) = 1 \wedge (\mathsf{x}_i, \mathsf{w}_i) \notin \mathbf{R}_{\mathsf{C}}\end{array}\right] \leq \mathsf{negl}(n)\,.$$

From the beginning till April 2018, Zerocash was using a variation of Pinocchio zk-SNARK [PHGR13] which is proposed by Ben Sasson et al.'s in [BCTV13]. Due to better efficiency, from April 2018 they switched to use Groth's zk-SNARK [Gro16] that is the state-of-the-art by the date of writing the thesis. We review Groth's zk-SNARK in next subsection.

### 2.7.2 Groth's zk-SNARK

As recently Zerocash started to use Groth's zk-SNARK, so this section reviews the construction of the scheme which is proposed by Groth in [Gro16]. As Groth zk-SNARK is constructed for Quadratic Arithmetic Programs (QAPs), so before going through the structure of the scheme we recall the definition of QAPs.

**Quadratic Arithmetic Program (QAP).** Let $\mathbb{Z}_p$ is a multiplicative group of integers modulo $p$, where p is a prime number. The multiplicative group $\mathbb{Z}_p^*$ uses only integers between 1 and $p-1$. A QAP instance $\mathcal{Q}_p$ is specified by the so defined $(\mathbb{Z}_p, m_0, \{u_j, v_j, w_j\}_{j=0}^m)$. This instance defines the following relation, where we assume that $A_0 = 1$:

$$\mathbf{R}_{\mathcal{Q}_p} = \left\{\begin{array}{l}(\mathsf{x},\mathsf{w})\colon \mathsf{x} = (A_1, \ldots, A_{m_0})^\top \wedge \mathsf{w} = (A_{m_0+1}, \ldots, A_m)^\top \wedge \\ (\sum_{j=0}^{m} A_j u_j(X))(\sum_{j=0}^{m} A_j v_j(\mathsf{x})) \equiv \sum_{j=0}^{m} A_j w_j(\mathsf{x}) \pmod{\ell(\mathsf{x})}\end{array}\right\}$$

Alternatively, $(\mathsf{x},\mathsf{w}) \in \mathbf{R}$ if there exists a (degree $\leq n-2$) polynomial $h(\mathsf{x})$, s.t.

$$(\sum_{j=0}^{m} A_j u_j(\mathsf{x}))(\sum_{j=0}^{m} A_j v_j(\mathsf{x})) - \sum_{j=0}^{m} A_j w_j(\mathsf{x}) = h(\mathsf{x})\ell(\mathsf{x})\ .$$

Clearly, $\mathbf{R}_{\mathcal{Q}} = \mathbf{R}_{\mathcal{Q}_p}$, given that $\mathcal{Q}_p$ is constructed from $\mathcal{Q}$ as above. We note that QAP language has an efficient reduction from the *circuit satisfiability* language. As a result, an efficient zk-SNARK constructed for a QAP, can be used as an efficient zk-SNARK for circuit satisfiability. This is the key point that has led to use zk-SNARKs in verifying various computations, including verifying the computations that are done in each Pour transaction of Zerocash.

Groth's zk-SNARK is a pairing based scheme. In this thesis, in reviewing of the scheme, we use additive notation together with the bracket notation, i.e., in group $\mathbb{G}_2$, $[a]_2 = a\,[1]_2$, where $[1]_2$ is a fixed generator of $\mathbb{G}_2$. A *bilinear group generator* $\mathsf{BGgen}(1^n)$ returns $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where $p$ (a large prime) is the order of cyclic abelian groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Finally, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficient non-degenerate bilinear pairing, s.t. $\hat{e}([a]_1, [b]_2) = [ab]_T$. Denote $[a]_1 \bullet [b]_2 = \hat{e}([a]_1, [b]_2)$.

On input a QAP instance $\mathcal{Q}_p = (\mathbb{Z}_p, m_0, \{u_j, \mathsf{V}_j, w_j\}_{j=0}^m)$, its algorithms act as follows:

$\mathsf{KGen}(\mathbf{R}_{\mathcal{Q}_p}, \mathsf{C})$**:** Generate $\mathsf{tr} = (\chi, \alpha, \beta, \gamma, \delta) \leftarrow_{\$} (\mathbb{Z}_p^*)^5$ and compute $(\mathsf{pk}, \mathsf{vk})$ such that,

$$\mathsf{pk} \leftarrow \begin{pmatrix} [\alpha, \beta, \delta, (\frac{u_j(\chi)\beta + \mathsf{V}_j(\chi)\alpha + w_j(\chi)}{\delta})_{j=m_0+1}^m]_1, \\ [(\chi^i \ell(\chi)/\delta)_{i=0}^{n-2}, (u_j(\chi), \mathsf{V}_j(\chi))_{j=0}^m]_1, [\beta, \delta, (\mathsf{V}_j(\chi))_{j=0}^m]_2 \end{pmatrix} ,$$
$$\mathsf{vk} \leftarrow \left( [(\frac{u_j(\chi)\beta + \mathsf{V}_j(\chi)\alpha + w_j(\chi)}{\gamma})_{j=0}^{m_0}]_1, [\gamma, \delta]_2, [\alpha\beta]_T \right) .$$

Return $(\mathsf{pk}, \mathsf{vk})$.

$\mathsf{P}(\mathbf{R}_{\mathcal{Q}_p}, \mathsf{pk}, \mathsf{x} = (A_1, \cdots, A_{m_0}), \mathsf{w} = (A_{m_0+1}, \cdots, A_m))$**:** Act as follows,

1. Let $a^\dagger(\chi) \leftarrow \sum_{j=0}^m A_j u_j(\chi)$, $b^\dagger(\chi) \leftarrow \sum_{j=0}^m A_j \mathsf{V}_j(\chi)$,
2. Let $c^\dagger(\chi) \leftarrow \sum_{j=0}^m A_j w_j(\chi)$,
3. Set $h(\chi) = \sum_{i=0}^{n-2} \mathsf{h}_i \chi^i \leftarrow (a^\dagger(\chi) b^\dagger(\chi) - c^\dagger(\chi))/\ell(\chi)$,
4. Set $[h(\chi)\ell(\chi)/\delta]_1 \leftarrow \sum_{i=0}^{n-2} \mathsf{h}_i [\chi^i \ell(\chi)/\delta]_1$,
5. Set $r_a \leftarrow_r \mathbb{Z}_p$; Set $\mathbf{a} \leftarrow \sum_{j=0}^m A_j [u_j(\chi)]_1 + [\alpha]_1 + r_a[\delta]_1$,
6. Set $r_b \leftarrow_r \mathbb{Z}_p$; Set $\mathbf{b} \leftarrow \sum_{j=0}^m A_j [\mathsf{V}_j(\chi)]_2 + [\beta]_2 + r_b[\delta]_2$,
7. Set $\mathbf{c} \leftarrow r_b \mathbf{a} + r_a(\sum_{j=0}^m A_j [\mathsf{V}_j(\chi)]_1 + [\beta]_1) +$
   $\sum_{j=m_0+1}^m A_j [(u_j(\chi)\beta + \mathsf{V}_j(\chi)\alpha + w_j(\chi))/\delta]_1 + [h(\chi)\ell(\chi)/\delta]_1$,
8. Return $\pi \leftarrow (\mathbf{a}, \mathbf{b}, \mathbf{c})$.

$\mathsf{Vf}(\mathbf{R}_{\mathcal{Q}_p}, \mathsf{vk}, \mathsf{x} = (A_1, \cdots, A_{m_0}), \pi = (\mathbf{a}, \mathbf{b}, \mathbf{c}))$**:** Check that

$$\mathbf{a} \bullet \mathbf{b} = \mathbf{c} \bullet [\delta]_2 + (\sum_{j=0}^{m_0} A_j [\frac{u_j(\chi)\beta + \mathsf{V}_j(\chi)\alpha + w_j(\chi)}{\gamma}]_1) \bullet [\gamma]_2 + [\alpha\beta]_T .$$

$\mathsf{Sim}(\mathbf{R}_{\mathcal{Q}_p}, \mathsf{pk}, \mathsf{x} = (A_1, \cdots, A_{m_0}), \mathsf{tr})$**:** Pick $a', b' \leftarrow \mathbb{Z}_p^*$, and compute $\pi = (\mathbf{a}, \mathbf{b}, \mathbf{c})$ such that

$$\mathbf{a} \leftarrow [a']_1 , \quad \mathbf{b} \leftarrow [b']_2 , \quad \mathbf{c} \leftarrow \left[ \frac{a'b' - \alpha\beta - \sum_{j=0}^{m_0} A_j (u_j(\chi)\beta + \mathsf{V}_j(\chi)\alpha + w_j(\chi))}{\delta} \right]_1$$

Figure 2. The structure of Groth's zk-SNARK [Gro16]

By considering the above preliminaries, Fig. 2 summarizes the construction of scheme which gets a QAP instance and executes the algorithms $(\mathsf{KGen}, \mathsf{P}, \mathsf{Vf})$, where $\mathsf{KGen}$ generates the public parameters (a.k.a. CRS), $\mathsf{P}$ generates a proof for given statement, and $\mathsf{Vf}$ verifies if the proof is valid. The figure also describes the simulation procedure (algorithm $\mathsf{Sim}$) of the scheme which is used in proof of zero-knowledge property.

# 3 Privacy in Digital Coins

The aim of this section is to first give a short overview on Bitcoin and its privacy concerns, and then introduce some of privacy-preserving cryptocurrencies that are proposed to improve privacy in Bitcoin.

We begin with describing construction of Bitcoin and some privacy flaws in the coin. Then, we continue with introducing Monero [Noe15], Zerocoin [MGGR13] and Zerocash [BCG+14a] that are some known privacy-preserving cryptocurrencies that are proposed to improve privacy of in Bitcoin. Similar to original coin, all the mentioned coins are constructed based on PoW consensus protocol and run on top of Bitcoin.

## 3.1 Bitcoin

Bitcoin was the first distributed peer-to-peer electronic cash system that can be used for online direct payments. It uses various cryptographic primitives to enable secure and direct payments. Roughly speaking, transactions are made by digitally signing the transferred value and the public key of the target receiver, and a public distributed ledger records all transactions. This public ledger is the key point in eliminating the need for a trusted third party (e.g. central banks). The founder is unknown and Satoshi Nakamoto [Nak08] is a pseudonym.

### 3.1.1 Overview

As briefly mentioned before, Bitcoin operates in a distributed network with a large number of nodes. To transfer coins, nodes of the network require to perform public transactions between pseudonyms. A Bitcoin transaction sends coins by signing a hash of the public key of the target receiver and the transferred value. The hash of the target receiver's public key is known as her address. In Bitcoin, this hash is displayed in a Base58 format, e.g. `3D2oetdNuZUqQHPJmcMDDHYoqkyNVsFk9r`. In practice, each user has several addresses (public keys) of digital signature scheme that are linked to electronic online wallets. An individual node of the network, can perform the transactions between one address that has enough money to the target address or addresses (public keys). To show that a particular user is the legitimate owner of the source address, and to validate the transaction, the particular user should sign the transaction with her secret key, such that other nodes in the network can verify the signature with her public key. There is a particular type of nodes that are called miners, which verify all transactions and construct new blocks to append to the ledger. Once a miner received a transaction and successfully verified it, they add the transaction to the upcoming block that they (miner or group of miners) will make and add to the ledger (blockchain). Once a miner is elected to make the next block (by a PoW lottery), she verifies the transaction and putd it to a new block and announces to the Bitcoin network to be confirmed by other nodes and users. Other nodes and miners, receive new blocks and append to the their local ledgers if their verification successfully passed. For each transaction, the payer needs to pay a transaction fee which corresponds to the cost of a transaction that should be paid to the miners. Additionally, to incentive the miners, the network adaptively gives rewards to the miners that construct the new block. Currently mining has a reward, so miners get reward and transaction fee, but later since the number of coins would be constant, so the reward for miners will be provided by transaction fees. Finally, the ledger is a chain of blocks where each block is connected to previous block with a cryptographic hash

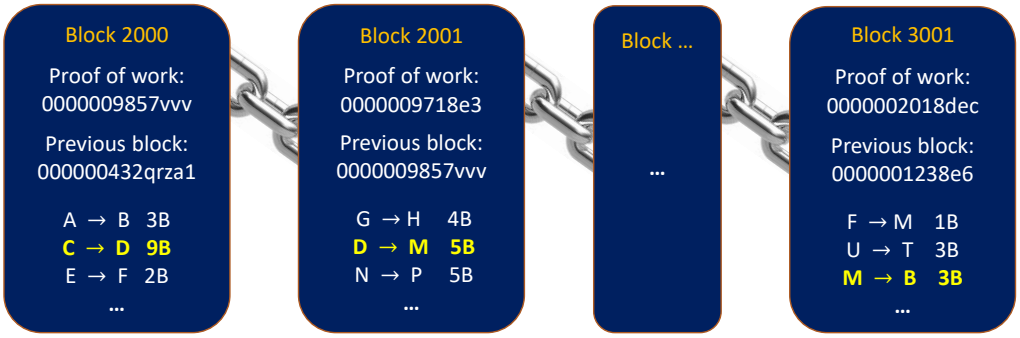


Figure 3. A graphical representation of Bitcoin's ledger

function, except the genesis block. A graphical representation is shown in Fig. 3.

We note that since the network is distributed, there would be cases that local copies of ledgers of nodes are different from each other. This issue is addressed with Nakamoto's consensus protocol that is reviewed in section 2.5. At the end of each transaction, the output is the amount of coins which is sent to a Bitcoin address (wallet) that is named "unspent transaction output" (UTXO). It is worth to mention that sum of UTXOs for a particular address, gives us the balance of that address (the unspent ones).

### 3.1.2 Construction

In summary, each block of Bitcoin's ledger contains the following information,

- $hpb$: Hash of the previous block in ledger, which is sort of a reference to the previous block (intuitively sticks blocks to each other)
- A list of transactions
- $hcb$: Hash of current block
- A solution to a PoW (described in section 2.5) puzzle consists of a value called nonce, s.t. $H(hpb||hcb||nonce) \leq tv$, where $H$ is a cryptographic hash function and $tv$ is a 256-bit *target value* announced to all the miners in the network.

Assuming that the output of cryptographic hash function $H$ is indistinguishable from a random value, there is no way to solve above puzzle better than trying all possible input values for the given nonce, until finding a proper input value which would be the answer for the given puzzle. The difficulty of finding such an answer for the PoW puzzle depends on value of $tv$ which specifies the number of zeros at the beginning of the 256-bit value (e.g. find a $tv$ value that has $5$ zeros at its left bits, e.g. $tv = 2^{251} - 1$).

We note that as the number of zeros increases the set of possible answers decreases and as a result the problem gets more difficult. Indeed this is one of parameters of the Bitcoin network to control the level of difficulty in PoW puzzle. With current setting in Bitcoin (at the time of writing the thesis), the value of $tv$ updates every 2016 blocks which increases the difficulty of the PoW puzzle. Changing this parameter allows to have constant block generation time on average. With current setting, in Bitcoin a new block is constructed roughly every 10 minutes.

### 3.1.3 Security and Privacy Concerns

Since Bitcoin's consensus protocol works based on PoW, so security of Bitcoin and its ledger relies on the assumption that majority of computational power in the network is under the control of honest miners. Attacks such as double spending are prevented if honest miners have more than half of total computation power in the network. Security of Bitcoin is formalized and studied in several works including [GKL15, PSS17, BMTZ17, GKL17]. In one research done in [BGM+18], it is shown that a miner who aims to maximize her rewards (a.k.a. rational miner), should behave honestly, as this case gives the maximum benefit for her. Due to nature of Bitcoin network which is distributed, DoS attacks are applicable by publishing a huge number of transactions, but this attack is mitigated due to transaction fees.

From privacy point of view, as briefly discussed in Sec. 1, different research have shown that transactions on Bitcoin are linkable [RH11]. Strictly speaking, graph analysis of Bitcoin's ledger has shown that, although transactions are between pseudonyms, if a user uses a fixed address for long time, one can easily link their transactions to each other. So, once a monitored address (pseudonym) is linked to an identity (a person), the sources and destinations of their coins, as well as the balance of their wallet, is laid bare. This is a serious loss of privacy, with potentially catastrophic real-world consequences. For instance, suppose Alice has her salary paid in Bitcoin. Later, Alice sends Bob, a coworker, some Bitcoin to pay for coffee. Bob now knows Alice's Bitcoin address. If Bob runs a Bitcoin node, he necessarily knows the total spendable balance of Alice's wallet. In addition, he can see all incoming transactions to Alice's wallet, and their amounts. From this he can surmise the amount of Alice's salary.

To address the discussed security and privacy concerns, several solution have been proposed that use various cryptographic techniques to provide secure and privacy-preserving payments [MGGR13, BCG+14a, Noe15, FMMO18]. In the rest of section, we present a short overview of the Monero [Noe15], Zerocoin [MGGR13] and Zerocash [BCG+14a] that are three prominent alternatives. We will observe that, actually Zerocash is an improved version of Zerocoin. Beside the privacy preserving coins that we briefly explain in this thesis there exists more privacy preserving coins namely, Dash [5], PIVXS [6], NavCoin [7], CloakCoin [8] and DeepOnion [9].

## 3.2 Monero

Monero [Noe15] is a privacy-preserving cryptocurrency that works based on PoW and aims to provide anonymous payments by hiding the source and destination of a transaction along with the value of transferred coins. Although, some research has shown that Monero is not successful in hiding the source and value of transactions, and one can learn information about them; more detailed discussions can be found in [KFTS17, MSH+18] but in the rest we provide a brief overview on the coin [KCA+18].

In order to hide the source and destination of transactions, Monero's protocol uses one-time keys and a ring signature [RST01]. They extended their protocol to hide the value of transactions in 2017. To this end, Monero moved to use Ring Confidential Transaction (Ring

[5] Available at `https://www.dash.org`
[6] Available at `https://pivx.org`
[7] Available at `https://cryptorating.eu`
[8] Available at `https://www.cloakcoin.com`
[9] Available at `https://deeponion.org`

CT) protocol [Noe15]. Hiding source, destination, and values of transactions is something that Bitcoin does not support.

### 3.2.1 MLSAG Signature

The main tools in Ring CT protocol is Multi Layered Linkable Spontaneous Anonymous Group (MLSAG) signature. Intuitively, a ring signature allows an individual user from a big group to sign a message in an anonymous way, such that no polynomial time adversary can distinguish which member of that group has signed the message. They deploy MLSAG signature, but we note that it still is a ring signature. MLSAG has some extra properties in comparison to the common ring signatures, which make them useful for cryptocurrencies. As one of the prominent properties is that MLSAG is linkable which allows to determine whether two signatures are created with identical secret key. As we mentioned before Monero beside hiding origion and destination of transactions, to hide the value of transactions uses Ring CT protocol which provides some extra properties on top of a ring signature and consequently MLSAG signature.

### 3.2.2 Ring Confidential Transaction

This subsection aims to summarize description of a transaction in the Ring CT protocol of Monero. First of all, the receiver generates a new one-time public key for MLSAG (each key is used just once). Inputs and outputs of a transaction are given as a Pedersen commitments which is explained in sec. 2.1.1. As the Pedersern commitment has additive homomorphic property, it allows third parties to verify that transaction is done correctly. For instance, let $\mathsf{Com}(\mathsf{sk}_1, m_1)$ be the input of a transaction, and $\mathsf{Com}(\mathsf{sk}_2, m_2)$, $\mathsf{Com}(\mathsf{sk}_3, m_3)$ are the outputs of the transaction; therefore anyone can verify that $\mathsf{Com}(\mathsf{sk}_2, m_2) \cdot \mathsf{Com}(\mathsf{sk}_3, m_3) = \mathsf{Com}(\mathsf{sk}_1, m_1)$ if $m_1 = m_2 + m_3$ and $\mathsf{sk}_1 = \mathsf{sk}_2 + \mathsf{sk}_3$. The sender of transaction opens the committed values only to the recipients of the transaction. Such that, only the receiver can verify the correctness of received amount. All above arithmetic are done in modulo $p$, the sender also gives some range proofs proving that commitments are in range$\{0, 1, \cdots, k\}$ where $k \ll p$.

At the end, the sender selects a randomly chosen subset of unspent public keys of various users along with her own public key and creates a MLSAG signature of the transaction with those keys. In practice the Ring CT protocol is a bit more complicated thanm our high level explanation, but in general the intuition behind the protocol is described here.

### 3.2.3 Privacy of Ring CT.

As we already mentioned, Ring CT protocol aims to provide an anonymous payments by hiding source, destination and value of each transaction. In the rest, we informally argue how the protocol archives its goals.

**Hiding the Receiver.** By considering the fact that public keys of an individual user are independent of other users, so it is computationally difficult to link various transactions to the same individual account.

**Hiding the Sender.** Due to nature of Ring signature, that hides an individual signer in a group of users, an attacker would not be able to say which of the public keys was used to create the

signature; as a result the sender is hidden among the number of members in the group and we can not distinguish which one of them is the sender (e.g. 1 out of 16)

**Hiding the Amount.** As Ring CT uses Pedersen commitment, and this commitment scheme is perfectly hiding, so the amount of transactions are private.

**Double Spending.** Regard to the double spending attack, in order to do this attack, a malicious sender would need to sign two messages with the same secret key, that due to linkability of MLSAG, it would be detected. Overflow attacks are avoided by Range proofs. e.g. if input of a particular transaction is $m_1 = 1$ and it outputs $m_2 = 2$ and $m_3 = -1$, then it still holds that $m_2 + m_3 = m_1$. But we know that amount of coins cannot be negative, and it does not make sense; on the other hand as arithmetic in Pedersen commitment is done modulo $p$ which is a large prime, then $-1$ can be considered as $p - 1$ that is a huge value. As a result, such an attack would allow to illegally make new currency with huge value. Recall that, range proofs show that each input and output are bounded by some value $k$, which mitigates this attack.

## 3.3 Zerocoin

As briefly mentioned before, Zerocash is a refined version of Zerocoin that provides more security and flexibility. Here we present a short overview on Zerocoin which can help to have better understanding of Zerocash in the rest of thesis.

Zerocoin is a concrete decentralized e-cash scheme that was proposed by Miers et al. in 2013 [MGGR13]. They have shown that their system is secure assuming the hardness of the Strong RSA[10] and Discrete Logarithm assumptions, and the existence of a zero-knowledge proof system. Roughly speaking, Zerocoin beside Bitcoin's transactions has two more transactions named: *mint* and *spend*. A *mint* transaction the same amount of is used to mint a new *zerocoin* by consuming Bitcoin. Zerocoin consists of a commitment value $\mathsf{cm}$ to a random serial number $\mathsf{sn}$. More precisely, in order to spend a coin which was minted in the past and its coin commitment recorded in the coin commitment list. In order to spend a coin, user needs to produce a spend transaction which consists of a destination address (recipient public key), the coin serial number $\mathsf{sn}$, and a non-interactive zero-knowledge proof for a $NP$ statement. The $NP$ statement is "I know secret $\mathsf{cm}$ and $r$ such that (i) $\mathsf{cm}$ can be opened to $\mathsf{sn}$ with commitment randomness $r$, and (ii) $\mathsf{cm}$ was previously minted at some point in the past". One of the very important property of zero-knowledge proofs is being *zero-knowledge* which means the proof does not leak any information about the source of the coin which is spending in the other words, it is not possible to link any two spend and mint transactions to each other. Another important property of zero-knowledge proofs is being *soundness* which means that if the verifier accepts the validity of transaction, it means the coin serial number is valid and has not appeared in the list of spent coins' serial number. Precisely speaking about committed values, Zerocoin uses Pedersen commitments over a prime field $\mathbb{F}_\mathsf{p}$, i.e., $\mathsf{cm} := g^{\mathsf{sn}} h^r$, where $g$ and $h$ are the random generators of $\mathbb{F}_p^*$ subgroup. To spend a Zerocoin a spender gives a zero-knowledge proof that she knows a coin commitment and the corresponding randomness from the list of minted coin commitments. To do so, they use an accumulator based on Strong-RSA [CL02]. Technically

[10]The assumptions states that given a modulus $N$ of unknown factorization, and a ciphertext $C$, it is infeasible to find any pair $(M, e)$ such that $C \equiv Me \mod N$.

speaking, the spender gives a proof that $\prod_i (C_i - \mathsf{cm}_t) = 0$, where $\mathsf{cm}_t$ denotes the commitment of the coin that is going to be spent and $C$ is the set of coin commitments so far. By adding a new coin commitment to the list of commitments the witnesses will be updated. The number of coin commitments can be unlimitedly increase, so it worth to mention that Zerocoin supports unlimited number of coins but later we will discuss that Zerocash supports a limited number of coins (precisely $2^{64}$ coins).

In practical, the proof size in Zerocoin is about $45$ kB at the 128-bit security level, and the proof verification takes around $0.5$ seconds [BCG+14a].

### 3.3.1 Construction

Formally speaking, Zerocoin consists of four algorithms $\Pi_{zerocoin} = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Vf})$ that can be expressed as below,

**Setup,** $pp \leftarrow \mathsf{Setup}(1^n)$**:** Given a security parameter $n$ as an input, algorithm Setup outputs the public parameters pp that are required for the system (including public parameter for the deployed proof system).

**Mint,** $(\mathsf{cm}, \mathsf{sk}_{\mathsf{cm}}) \leftarrow \mathsf{Mint}(pp)$**:** Given public parameters pp as inputs, algorithm Mint returns $(\mathsf{cm}, \mathsf{sk}_{\mathsf{cm}})$, where cm is the Pedersen commitment of coin and $\mathsf{sk}_{\mathsf{cm}}$ is the secret information of coin.

**Spend,** $(\pi, \mathsf{sn}) \leftarrow \mathsf{Spend}(pp, \mathsf{cm}, \mathsf{sk}_{\mathsf{cm}}, C)$**:** Given public parameters pp, a commitment cm, the corresponding secret information $\mathsf{cm}_{\mathsf{sk}}$ and the set of commitments $C$, the algorithm Spend checks if $\mathsf{cm} \notin C$ outputs $\perp$. Otherwise, returns $(\pi, \mathsf{sn})$ where $\pi$ is a zero-knowledge proof to attest that prover knows secret information of the spend coin, and the coin is not spend before (serial number sn is not published before).

**Verify,** $0/1 \leftarrow \mathsf{Vf}(pp, \pi, \mathsf{sn}, C)$**:** Given inputs such as a proof $\pi$, a serial number sn, and a set of coins $C$, the algorithm Vf verifies the validity of proof $\pi$, it returns $1$ if proof verifies successfully and returns $0$ if the proof verification fails.

It is worth to mention that, since Zerocoin uses non-interactive zero-knowledge proofs, they need a trusted third party to generate public parameters honestly. Actually later we will observe that Zerocash also needs a similar trusted setup phase.

### 3.3.2 Security

The security of Zerocoin [MGGR13] depends on two theorems which we will briefly mention below.

**Theorem 1.** Using the property computationally zero-knowledge in the random oracle model of the zero-knowledge signature of knowledge they guarantee anonymity of Zerocoin scheme $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Vf})$.

Intuitively, in order to provide anonymity for users in Zerocoin, the signature proof $\pi$ should at least be computationally zero-knowledge and the commitment cm has to be perfectly hiding. These two requirements give the opportunity to spend a coin anonymously because the adversary can only guess which coin is being spent.

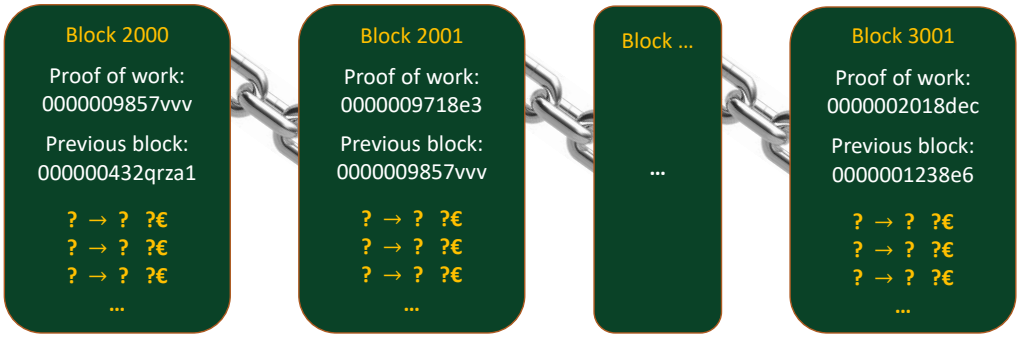


Figure 4. A graphical representation of ledger in privacy-preserving coins (e.g. Zcash)

**Theorem 2.** Using the property of soundness in the random oracle model of the signature proof $\pi$, the hardness of Strong RSA problem and the hardness of Discrete Logarithm problem in $G$, they guarantee Balance property of Zerocoin scheme $\Pi = (\mathsf{Setup}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Vf})$.

Intuitively, In order to provide Balance property for users in Zerocoin, there is need to have the perfect binding coin commitment, the sound and unforgeable zero-knowledge signature of knowledge and the collision resistant accumulator. The advantage of adversary to win Balance game is negligible otherwise the adversary would be able to find a collision for commitment scheme which would lead to solving of the Discrete Logarithm problem or find a collision resistant for accumulator which would lead to solving the Strong RSA problem.

## 3.4 Zerocash

Now we move on to Zerocash [BCG$^+$14a] which is a construction based on Zerocoin and Sander and Ta-Shma's scheme [ST99], by some feature extensions. In a short comparison with current Bitocoin network, Zerocash is more scalable and can handle larger number of transactions per seconds; as each block is generated in $\approx 2.5$ minutes (instead of $\approx 10$ minutes in the Bitcoin) and block capacity is $2.5$MB (compared to $1$MB for Bitcoin). More importantly, Zerocash allows complete anonymous transactions by hiding source, destination and value of each transaction. But, Zerocoin does not hide the value of a transaction and other meta data. Zerocoin uses coins of fixed denomination and does not allow users to do payments of exact values and users need to use a support coin such as Bitcoin, but these issues are solved in Zerocash. As Zerocash completely hides the source, the destination and the amount of transactions, so its ledger has a structure as in Fig. 4 that in comparison with ledger of Bitcoin in Fig. 3, it reveals minimum information about the transactions.

### 3.4.1 Construction

This section aims to present a step-by-step review of the construction of Zerocash. The design of Zerocash, was later developed to a full-fledged cryptocurrency which is known as Zcash these days. As we briefly mentioned in Sec. 2.6, Zerocash is a construction of DAP scheme with a particular instantiation [BCG$^+$14a]. In the rest, with recalling that a DAP scheme $\Pi_{\mathsf{DAP}}$ consists of a tuple of polynomial time algorithms $\Pi_{\mathsf{DAP}} = (\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$, we discuss how the algorithms are constructed in Zerocash. We already have introduced all the primitives including collision-resistant hash functions,

commitments schemes, (one-time strongly-unforgeable) signature schemes, (key-private) public-key encryption schemes, and zk-SNARKs that are deployed in the construction of the scheme. Later we will talk about precise instantiation of these primitives in the construction of Zerocash.

**Algorithms of the DAP Scheme in Zerocash.** The DAP scheme $\Pi_{Zcash}$ consists of the algorithms $(\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive})$ defined as follows.

Setup Algorithm. The Setup algorithm generates public parameter needed in other algorithms (including CRS of the zk-SNARK) which is executed only once by a trusted third party or distributed authority. It returns public parameter $\mathsf{pp}$ which is shared with all nodes. Public parameter $\mathsf{pp}$ contains encryption public parameter $\mathsf{pp}_{\mathsf{Enc}}$, signature public parameter $\mathsf{pp}_{\mathsf{Sig}}$ and a tuple $(\mathsf{pk}_{\mathsf{Pour}}, \mathsf{vk}_{\mathsf{Pour}})$, as the public parameter of the zk-SNARK, which will be used in the following algorithms. Roughly speaking, the algorithm generates public parameters $\mathsf{pp}_{\mathsf{Enc}}$ and $\mathsf{pp}_{\mathsf{Sig}}$ for key generation of the encryption and the signature schemes, respectively; and the keys $(\mathsf{pk}_{\mathsf{Pour}}, \mathsf{vk}_{\mathsf{Pour}})$ which will be used in Pour algorithm to generate and verify the proofs. The algorithm Setup is described in Alg. 1.

**Algorithm 1:** Setup

**Input:** security parameter $(n)$
**Result:** public parameters $\mathsf{pp}$

1 Construct arithmetic circuit $C_{\mathsf{Pour}}$ for Pour transaction at security $n$.
2 Compute $(\mathsf{pk}_{\mathsf{Pour}}, \mathsf{vk}_{\mathsf{Pour}}) := \mathsf{KGen}_{\mathsf{NIZK}}(1^n, C_{\mathsf{Pour}})$.
3 Compute $\mathsf{pp}_{\mathsf{Enc}} := \mathsf{KGen}_{\mathsf{Enc}}(1^n)$.
4 Compute $\mathsf{pp}_{\mathsf{Sig}} := \mathsf{KGen}_{\mathsf{Sig}}(1^n)$.
5 Set $\mathsf{pp} := (\mathsf{pk}_{\mathsf{Pour}}, \mathsf{vk}_{\mathsf{Pour}}, \mathsf{pp}_{\mathsf{Enc}}, \mathsf{pp}_{\mathsf{Sig}})$.
6 Output $\mathsf{pp}$.

CreateAddress Algorithm. Given the public parameters $\mathsf{pp}$, the algorithm generates a tuple of addresses $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$. To do so, it parses $\mathsf{pp} := (\mathsf{pk}_{\mathsf{Pour}}, \mathsf{vk}_{\mathsf{Pour}}, \mathsf{pp}_{\mathsf{Enc}}, \mathsf{pp}_{\mathsf{Sig}})$ and then using parameters $\mathsf{pp}_{\mathsf{Enc}}$ and algorithms $\mathsf{KGen}_{\mathsf{Enc}}$, it generates a pair of public and secret keys $(\mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}})$. Next, it samples a PRF with a seed $\mathsf{a}_{\mathsf{sk}}$ and uses it to generate $\mathsf{a}_{\mathsf{pk}}$. Finally this algorithm returns $\mathsf{addr}_{\mathsf{pk}}$ which consists of $\mathsf{a}_{\mathsf{pk}}$ and $\mathsf{pk}_{\mathsf{Enc}}$ and $\mathsf{addr}_{\mathsf{sk}}$ which consists of $\mathsf{a}_{\mathsf{sk}}$ and $\mathsf{sk}_{\mathsf{Enc}}$. The algorithm CreateAddress is shown in Alg. 2.

**Algorithm 2:** CreateAddress

**Input:** public parameters $\mathsf{pp}$
**Result:** address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

1 Compute $(\mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}) := \mathsf{KGen}_{\mathsf{Enc}}(\mathsf{pp}_{\mathsf{Enc}})$.
2 Randomly sample a $\mathsf{PRF}^{\mathsf{addr}}$ with a randomly selected seed $\mathsf{a}_{\mathsf{sk}}$.
3 Compute $\mathsf{a}_{\mathsf{pk}} = \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a}_{\mathsf{sk}}}(0)$.
4 Set $\mathsf{addr}_{\mathsf{pk}} := (\mathsf{a}_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{Enc}})$.
5 Set $\mathsf{addr}_{\mathsf{sk}} := (\mathsf{a}_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{Enc}})$.
6 Output $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$.

Mint Algorithm. This algorithm is used to create a new coin with a value V. It produces a coin $c$ and a *mint transaction* $\mathsf{Tran}_{\mathsf{Mint}}$ to record the coin $c$ in the ledger. More precisely, the Mint

algorithm gets public parameters, the desired coin value (which is bounded by $\mathsf{V}_{\max}$ initialized in the protocol) and address public key $\mathsf{addr}_{\mathsf{pk}}$ for the user that wants to mind a coin. In order to generate the serial number of the new coin, user picks a $\rho$ randomly as a seed and runs a pseudo random function on her secret address $\mathsf{a}_{\mathsf{sk}}$. On the other hand, to get commitment of the new coin user needs to pick to randomnesses $r$ and $s$ and first produce $k$ as $k := \mathsf{Com}_r(\mathsf{a}_{\mathsf{pk}}||\rho)$ and finally produce coin commitment as $\mathsf{cm} := \mathsf{Com}_s(\mathsf{V}||k)$. The produced new coin $c$ includes public address key of user, coin value $\mathsf{V}$, the secrets $\rho, r, s$ and the coin commitment $\mathsf{cm}$ which will be placed in the commitments' merkle tree. The other product of this algorithm is mint transaction $\mathsf{Tran}_{\mathsf{Mint}}$ which consists of $(\mathsf{cm}, \mathsf{V}, *)$, where $\mathsf{cm}$ is the coin commitment, $\mathsf{V}$ is a value of the new coin and $*$ is $r$ and $s$. The procedure of algorithm is described in Alg. 3.

**Algorithm 3:** Mint

**Input:** public parameters $\mathsf{pp}$, coin value $v \in [0, \ldots, \mathsf{V}_{\max}]$, receiver's $\mathsf{addr}_{\mathsf{pk}}$
**Result:** coin $c$ and mint transaction $\mathsf{Tran}_{\mathsf{Mint}}$

1 Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(\mathsf{a}_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{Enc}})$.
2 Randomly sample a $\mathsf{PRF}^{\mathsf{sn}}$ seed $\rho$.
3 Randomly sample two $\mathsf{Com}$ trapdoors $r, s$.
4 Compute $k := \mathsf{Com}_r(\mathsf{a}_{\mathsf{pk}} \parallel \rho)$.
5 Compute $\mathsf{cm} := \mathsf{Com}_s(\mathsf{V} \parallel k)$.
6 Set $c := (\mathsf{addr}_{\mathsf{pk}}, \mathsf{V}, \rho, r, s, \mathsf{cm})$.
7 Set $\mathsf{Tran}_{\mathsf{Mint}} := (\mathsf{cm}, \mathsf{V}, *)$, where $* := (k, s)$.
8 Output $c$ and $\mathsf{Tran}_{\mathsf{Mint}}$

Pour Algorithm. This is the most important algorithm in the DAP scheme and consequently Zerocash, which is used to spend old coins and produce new coins and also pay the transaction fee. Technically speaking it takes two old coins $c_1^{\mathsf{old}}$ and $c_2^{\mathsf{old}}$ and produces two new coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ for the public address keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$, respectively and it can be generalized to the case which takes $n$ old coins as an input and outputs $m$ new coins. It also outputs some public value which can be spent as the transaction fee. This algorithm allows to merge coins, subdivide coins into smaller denominations and also do public payments. To check the validity of spent coins, the algorithm takes the root of the Merkle tree over all the coin commitments verifies the authentication paths $\mathsf{Path}_1$ and $\mathsf{Path}_2$ which should go to two spent coins. The algorithm also takes values $\mathsf{V}_1^{\mathsf{new}}$ $\mathsf{V}_2^{\mathsf{new}}$ as values of two new anonymous coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ with the corresponding address public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$. The other input of Pour algorithm is the value $\mathsf{V}_{pub}$ that shows the amount which is publicly spent, for instance for transaction fees. The summation of the new coins' value plus the $\mathsf{V}_{pub}$ must be exactly equal to the summation of the old coins' value. The last input of Pour algorithm is an arbitrary string $\mathsf{info}$ which is bound into the output pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$. The output of algorithm consists two new coins $c_1^{\mathsf{new}}$ and $c_2^{\mathsf{new}}$ and a pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$. The transaction $\mathsf{Tran}_{\mathsf{Pour}}$ contains the root $\mathsf{rt}$, two old coins' serial numbers $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$, two commitments' of new coins $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$, the public value $\mathsf{V}_{pub}$, a arbitrary string $\mathsf{info}$ and $*$, where $*$ is some extra information related to the implementation.

Intuitively, this algorithm aims to spend old coins and create new coins and produce a pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$ which will be published in the ledger. This algorithm using zk-SNARK generates a proof for a special statement which is important to provide anonymity and in order

to avoid the man-in-the-middle attack of the proof it uses a one time strong secure signature. The procedure of algorithm Mint is described in Alg. 4.

**Algorithm 4:** Pour

**Input:** public parameters pp, the Merkle root rt, old coins $c_1^{\mathsf{old}}$ and $c_2^{\mathsf{old}}$, old address secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}$ and $\mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$, path $\mathsf{Path}_1$ from commitment $\mathsf{cm}(c_2^{\mathsf{old}})$ to the root rt and path $\mathsf{Path}_2$ from commitment $\mathsf{cm}(c_1^{\mathsf{old}})$ to the root rt, new values $\mathsf{V}_1^{\mathsf{new}}$ and $\mathsf{V}_2^{\mathsf{new}}$, new addresses public keys $\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$ and $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$, public value $\mathsf{V}_{pub}$, transaction string info

**Result:** new coins $c_1^{\mathsf{new}}, c_2^{\mathsf{new}}$ and pour transaction $\mathsf{Tran}_{\mathsf{Pour}}$

**foreach** $i \in \{0,1\}$ **do**
- Parse $c_i^{\mathsf{old}}$ as $(\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{V}_i^{\mathsf{old}}, \rho_i^{\mathsf{old}}, {}_i^{\mathsf{old}}, s_i^{\mathsf{old}}, \mathsf{cm}_i^{\mathsf{old}})$;
- Parse $c_i^{\mathsf{old}}$ as $(\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{V}_i^{\mathsf{old}}, \rho_i^{\mathsf{old}}, {}_i^{\mathsf{old}}, s_i^{\mathsf{old}}, \mathsf{cm}_i^{\mathsf{old}})$;
- Parse $\mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}}$ as $(\mathsf{a}_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{sk}_{\mathsf{Enc},i}^{\mathsf{old}})$;
- Compute $\mathsf{sn}_i^{\mathsf{old}} := \mathsf{PRF}_{\mathsf{a}_{\mathsf{sk}},i}^{\mathsf{sn}}(\rho_i^{\mathsf{old}})$;
- Parse $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}$ as $(\mathsf{a}_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{pk}_{\mathsf{Enc},i}^{\mathsf{new}})$;
- Randomly sample a $\mathsf{PRF}^{\mathsf{sn}}$ seed $\rho_i^{\mathsf{new}}$;
- Randomly sample two Com trapdoors $r_i^{\mathsf{new}}, s_i^{\mathsf{new}}$;
- Compute $k_i^{\mathsf{new}} := \mathsf{Com}_{r_i^{\mathsf{new}}}(\mathsf{a}_{\mathsf{pk},i}^{\mathsf{new}} \parallel \rho_i^{\mathsf{new}})$;
- Compute $\mathsf{cm}_i^{\mathsf{new}} := \mathsf{Com}_{s_i^{\mathsf{new}}}(\mathsf{V}_i^{\mathsf{new}} \parallel k_i^{\mathsf{new}})$;
- Set $c_i^{\mathsf{new}} := (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{V}_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, {}_i^{\mathsf{new}}, s_i^{\mathsf{new}}, \mathsf{cm}_i^{\mathsf{new}})$;
- Set $C_i := \mathsf{Enc}(\mathsf{pk}_{\mathsf{Enc}}^{\mathsf{new}}, (\mathsf{V}_i^{\mathsf{new}}, \rho_i^{\mathsf{new}}, {}_i^{\mathsf{new}}, s_i^{\mathsf{new}}))$

Generate $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) := \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp}_{\mathsf{Sig}})$.

Compute $\mathsf{h}_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{Sig}})$.

Compute $h_1 := \mathsf{PRF}_{\mathsf{a}_{\mathsf{sk},1}^{\mathsf{old}}}^{\mathsf{pk}}(\mathsf{h}_{\mathsf{Sig}})$ and $h_2 := \mathsf{PRF}_{\mathsf{a}_{\mathsf{sk},2}^{\mathsf{old}}}^{\mathsf{pk}}(\mathsf{h}_{\mathsf{Sig}})$.

Set $\vec{x} := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{h}_{\mathsf{Sig}}, (h_1, h_2))$.

Set $\vec{a} := (\mathsf{Path}_1, \mathsf{Path}_2, c_1^{\mathsf{old}}, c_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, c_1^{\mathsf{new}}, c_2^{\mathsf{new}})$.

Compute $\pi_{\mathsf{Pour}} := \mathsf{P}(\mathsf{pk}_{\mathsf{Pour}}, \vec{x}, \vec{a})$.

Set $m := (\vec{x}, \pi_{\mathsf{Pour}}, \mathsf{info}, (C_1, C_2))$.

Compute $\sigma := \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, m)$.

Set $\mathsf{Tran}_{\mathsf{Pour}} := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info}, *)$, where $* := \mathsf{pk}_{\mathsf{Sig}}, (h_1, h_2), \pi_{\mathsf{Pour}}, (C_1, C_2), \sigma)$.

Output $c_1^{\mathsf{new}}, c_2^{\mathsf{new}}$ and $\mathsf{Tran}_{\mathsf{Pour}}$

VerifyTransaction Algorithm. To check if a transaction is valid and well-formed, nodes of the network need to execute the algorithm VerifyTransaction. For instance, to check a mint transaction, it takes $\mathsf{Tran}_{\mathsf{Mint}}$ which contains $(\mathsf{cm}, \mathsf{V}, *)$ where $*$ is randomnesses $k$ and $s$ and sets $\mathsf{cm}' := \mathsf{Com}_s(\mathsf{V}||k)$. Then it returns $b = 1$ if $\mathsf{cm}' = \mathsf{cm}$ otherwise returns $b = 0$.

To check the validity of pour transaction, this algorithm needs to check validity of proof and validity of signature. It takes $\mathsf{Tran}_{\mathsf{Pour}}$ which contains $\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info}, *$ where $*$ is $\mathsf{pk}_{\mathsf{Sig}}, (h_1, h_2), \pi_{\mathsf{Pour}}, (C_1, C_2), \sigma$ and checks some steps. First checks if the old serial numbers already appeared in the spent serial number list or $\mathsf{sn}_1^{\mathsf{old}} = \mathsf{sn}_2^{\mathsf{old}}$, if so then returns $b = 0$ otherwise returns 1. Next checks if this rt root appears on the ledger, if

yes returns $b = 1$, otherwise outputs $b = 0$. In the next step computes $\mathsf{h}_{\mathsf{Sig}}$ which was used in the signing process in the Pour algorithm. After that sets the statement as $x := \mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{h}_{\mathsf{Sig}}, (h_1, h_2)$ and message as $m := x, \pi_{\mathsf{Pour}}, \mathsf{info}, (C_1, C_2)$. Finally first runs signature's verifier and sets $b$ as $b := \mathsf{Vf}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m, \sigma)$. Then running proof verifier on the proof computes $b'$ as $b' := \mathsf{VerifyTransaction}(\mathsf{crs}_v, x, \pi_{\mathsf{Pour}})$. At the end returns $b \bigwedge b$'. If the result is 1 means that both proof and signature are valid otherwise means at least one of them is not valid.

Intuitively, this algorithm has two parts, one is checking validity of mint transaction $\mathsf{Tran}_{\mathsf{Mint}}$ produced by Mint algorithm and the second part checks if the $\mathsf{Tran}_{\mathsf{Pour}}$ generated by Pour algorithm (which means checking both proof and signature) is valid or not. The procedure of this algorithm is described in Alg. 5.

**Algorithm 5:** VerifyTransaction

**Input:** public parameters pp, a (mint or pour) transaction Tran, the current ledger L
**Result:** bit $b$, equal 1 iff the transaction is valid

**if** *given a mint transaction* $\mathsf{Tran} = \mathsf{Tran}_{\mathsf{Mint}}$ **then**
  Parse $\mathsf{Tran}_{\mathsf{Mint}}$ as $(\mathsf{cm}, \mathsf{V}, *)$, and $*$ as $(k, s)$;
  Set $\mathsf{cm}' := \mathsf{Com}_s(\mathsf{V} \parallel k)$;
  Compute $\mathsf{sn}_i^{\mathsf{old}} := \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a}_{\mathsf{sk}},i}(\rho_i^{\mathsf{old}})$;
  Output $b := 1$ if $\mathsf{cm} = \mathsf{cm}'$, else output $b := 0$
**if** *given a pour transaction* $\mathsf{Tran} = \mathsf{Tran}_{\mathsf{Pour}}$ **then**
  Parse $\mathsf{Tran}_{\mathsf{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info}, *)$, where $* := \mathsf{pk}_{\mathsf{Sig}}, (h_1, h_2), \pi_{\mathsf{Pour}}, (C_1, C_2), \sigma)$;
  If $\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}$ appears on L or $(\mathsf{sn}_1^{\mathsf{old}} = \mathsf{sn}_2^{\mathsf{old}})$, output $b := 0$;
  If the Merkle root rt does not appear on L, output $b := 0$;
  Compute $\mathsf{h}_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{Sig}})$;
  Set $\vec{x} := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{h}_{\mathsf{Sig}}, (h_1, h_2))$;
  Set $m := (\vec{x}, \pi_{\mathsf{Pour}}, \mathsf{info}, (C_1, C_2))$;
  Compute $b := \mathsf{Vf}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m, \sigma)$;
  Compute $b' := \mathsf{Vf}(\mathsf{vk}_{\mathsf{Pour}}, \vec{x}, \pi_{\mathsf{Pour}})$, and output $b \wedge b'$

Receive Algorithm. This is the algorithm that when a node with address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ wishes to receive payments sent to the $\mathsf{addr}_{\mathsf{pk}}$, needs to execute to scan the ledger of Zerocash. The algorithms returns the corresponding coins that their serial number have not published on the SNList. In summary, this algorithm scans the ledger and returns all the unspent coins for a special address key. The procedure of the algorithm is described in Alg. 6.

A key point about the DAP scheme in Zerocash is that it uses Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) [BCG+14a, Lip12, BCI+13, ABLZ17a] under some cryptographic assumptions. For all the ledger based digital currency such as Bitcoin one can use their proposed DAP scheme. Currently Zerocash uses the most efficient zk-SNARK that is proposed by Groth in [Gro16].

**Algorithm 6:** Receive

**Input:** public parameters pp, recipient's $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$, the current ledger $\mathsf{L}$
**Result:** set of received coins

Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(\mathsf{a}_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{Enc}})$.
Parse $\mathsf{addr}_{\mathsf{sk}}$ as $(\mathsf{a}_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{Enc}})$.
**foreach** *Pour transaction* $\mathsf{Tran}_{\mathsf{Pour}}$ *on the ledger* **do**
    Parse $\mathsf{Tran}_{\mathsf{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{V}_{pub}, \mathsf{info}, *)$, where $* := \mathsf{pk}_{\mathsf{Sig}}, (h_1, h_2), \pi_{\mathsf{Pour}}, (C_1, C_2), \sigma)$;
    **foreach** $i \in \{1, 2\}$ **do**
        Compute $(\mathsf{V}_i, \rho_i, r_i, s_i) := \mathsf{Dec}_{\mathsf{Enc}}(\mathsf{sk}_{\mathsf{Enc}}, C_i)$;
        **if** $\mathsf{Dec}_{\mathsf{Enc}}$*'s output is not* $\perp$ **then**
            verify that, $\mathsf{cm}_i^{\mathsf{new}}$ equals $\mathsf{Com}_{s_i}(\mathsf{V}_i \parallel \mathsf{Com}_{r_i}(\mathsf{a}_{\mathsf{pk}} \parallel \rho_i))$;
            verify that, $\mathsf{sn}_i := \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a}_{\mathsf{sk}}}(\rho_i)$ does not appear on $\mathsf{L}$;
        **if** *If both checks succeed* **then**
            output $c_i := (\mathsf{addr}_{\mathsf{pk}}, \mathsf{V}_i, \rho_i, r_i, s_i, \mathsf{Com}_i^{\mathsf{new}})$

### 3.4.2 Instantiation

As we already mentioned, the coin Zerocash is an instantiation of the described DAP scheme for particular security parameter. As one can see in the algorithms mentioned above, there are collision resistant hash functions CRH, pseudo random functions PRF and commitment schemes Com that Zerocash instantiate all of them with hash function SHA-256 which takes $512$ bits as an input and outputs $256$ bits. Taking precisely 512-bit inputs in hash function is useful in construction of the Merkle tree.

**Pseudo Random Function Instantiation.** The pseudo random functions are instantiated as $\mathsf{PRF}^{\mathsf{addr}}_x(z) := \mathsf{H}(x||00||z)$, $\mathsf{PRF}^{\mathsf{sn}}_x(z) := \mathsf{H}(x||01||z)$ and $\mathsf{PRF}^{\mathsf{pk}}_x(z) := \mathsf{H}(x||10||z)$.

**Commitment Schemes Instantiation.** Zerocash instantiates the commitment schemes as follows: first instantiate commitment for producing $k$ as $k := \mathsf{Com}_r(\mathsf{a}_{\mathsf{pk}}||\rho) = \mathsf{H}(r||[\mathsf{H}(\mathsf{a}_{\mathsf{pk}}||\rho)]_{128})$. Above, $[\cdot]_{128}$ denotes that we are truncating the 256-bit string to $128$ bits (say, by dropping least significant bits, as in their implementation) Then, using $k$ they instantiate coin commitment as $\mathsf{cm} := \mathsf{Com}_s(\mathsf{V}||k) = \mathsf{H}(k||0^{192}||v)$.

**NP Statement Instantiation.** Zerocash instantiate the NP statement Pour algorithm such that it checks the following holds, for each $i \in \{1, 2\}$:

- $\mathsf{Path}_i$ is an authentication path for leaf $\mathsf{cm}_i^{\mathsf{old}}$ with respect to root $\mathsf{rt}$, in a CRH-based Merkle tree;
- $a_{\mathsf{pk},i}^{\mathsf{old}} = \mathsf{H}(a_{\mathsf{sk},i}^{\mathsf{old}}||0^{256})$;
- $\mathsf{sn}_i^{\mathsf{old}} = \mathsf{H}(a_{\mathsf{sk},i}^{\mathsf{old}}||01||[\rho_i^{\mathsf{old}}]_{254})$;

- $cm_i^{\mathsf{old}} = \mathsf{H}(\mathsf{H}(r_i^{\mathsf{old}}||[\mathsf{H}(a_{\mathsf{pk},i}^{\mathsf{old}}||\rho_i^{\mathsf{old}})]_{128})||0^{192}||\mathsf{V}_i^{\mathsf{old}})$;
- $cm_i^{\mathsf{new}} = \mathsf{H}(\mathsf{H}(r_i^{\mathsf{new}}|[\mathsf{H}(a_{\mathsf{pk},i}^{\mathsf{new}}||\rho_i^{\mathsf{new}})]_{128})||0^{192}||\mathsf{V}_i^{\mathsf{new}})$ and
- $\mathsf{h}_i = \mathsf{H}(a_{\mathsf{sk},i}^{\mathsf{old}}||10||b_i||[\mathsf{h}_{\mathsf{Sig}}]_{253})$ where $b_1 := 0$ and $b_2 := 1$.

Beside the checking which are mentioned above Pour algorithm needs to check $\mathsf{V}_1^{\mathsf{new}} + \mathsf{V}_2^{\mathsf{new}} + \mathsf{V}_{pub} = \mathsf{V}_1^{\mathsf{old}} + \mathsf{V}_2^{\mathsf{old}}$, with $\mathsf{V}_1^{\mathsf{old}}, \mathsf{V}_2^{\mathsf{old}} \geq 0$ and $\mathsf{V}_1^{\mathsf{old}} + \mathsf{V}_2^{\mathsf{old}} < 264$. Zerocash in order to build circuit for Pour algorithm fixed a Merkle-tree depth $d_{tree}$ with $64$ which this depth supports up to $2^{64}$ that is enough for Zerocash currency.

**Signature Scheme Instantiation.** Zerocash instantiate signature scheme with ECDSA signature scheme [Kob98].

**Encryption Function Instantiation.** For instantiating encryption function they use the key-private Elliptic-Curve Integrated Encryption Scheme (ECIES) [Cer00].

### 3.4.3 Security

As we already talked about the security requirements of a DAP scheme in Sec. 2.6.3, since Zerocash is an instantiation of a DAP scheme, it should guarantee the required security. Security proofs for Zerocash are explained with more details in the full version of the paper and we refer reader to the paper [BCG+14a].

**An issue with definition of Completeness.** While we have been studying the structure and security proofs of Zerocash, we noticed that the definition of completeness has some issues and it does not cover all requirements of users in real life. Precisely speaking, we observed a flaw on the construction of zerocash which allows a malicious spender to make the spent coins unspendable to the recipient, while the network will accept the transactions and the Receive algorithm will not detect the issue in the recipient side. Then we got aware that the attack is correct, but 2 years ago Zooko Wilcox had reported the flaw and the issue is solved in current version of Zcash protocol and is described with more details in recent update of the protocol specification, but it still exists in the full version of paper.

In summary, the flaw stems from the construction of new coins in Pour algorithm. More accurately, in constructing the new coins, the sender picks the serial number randomnesses $\rho_1$, $\rho_2$ and sends them securely to the recipients; and later when recipients want to spend the new coins, they will use the randomnesses along with their secret keys to generate the serial numbers of new coins. It is easy to show that generating the randomnesses of new serial numbers by spender, allows a malicious spender to act in a way that some of legitimate coins in recipients side would be unspendable.

### 3.4.4 Efficiency

Setup algorithm in their construction includes constructing $C_{\mathsf{Pour}}$ and CRS generation for $C_{\mathsf{Pour}}$. They instantiate PRFs and Commitment schemes with $SHA256$ (except a Hash function in signing). They constructed an arithmetic circuit $C_{SHA256}$ for verifying SHA256's function from

Table 1. Number of gates needed in for checks in $C_{\mathsf{Pour}}$

| Gate count for circuit $C_{\mathsf{Pour}}$ | |
|---|---|
| Ensure $\mathsf{cm}_1^{\mathsf{old}}$ is in Merkle tree | $1,802,304$ |
| ($1$ layer out of $64$) | $28,161$ |
| Ensure $\mathsf{cm}_2^{\mathsf{old}}$ is in Merkle tree | $1,802,304$ |
| ($1$ layer out of $64$) | $28,161$ |
| Check computation of $\mathsf{sn}_1^{\mathsf{old}}$, $\mathsf{sn}_2^{\mathsf{old}}$ | $2 \times 27,904$ |
| Check computation of $\mathsf{a}_{\mathsf{pk},1}^{\mathsf{old}}$, $\mathsf{a}_{\mathsf{pk},2}^{\mathsf{old}}$ | $2 \times 27,904$ |
| Check computation of $\mathsf{cm}_1^{\mathsf{old}}$, $\mathsf{cm}_2^{\mathsf{old}}$, $\mathsf{cm}_1^{\mathsf{new}}$, $\mathsf{cm}_2^{\mathsf{new}}$ | $4 \times 83,712$ |
| Check computation of $(h_1, h_2)$ | $2 \times 27,904$ |
| Ensure that $\mathsf{V}_1^{\mathsf{new}} + \mathsf{V}_2^{\mathsf{new}} + \mathsf{V}_{pub} = \mathsf{V}_1^{\mathsf{old}} + \mathsf{V}_2^{\mathsf{old}}$ | $1$ |
| Ensure that $\mathsf{V}_1^{\mathsf{old}} + \mathsf{V}_2^{\mathsf{old}} < 2^{64}$ | $65$ |
| Miscellaneous | $2,384$ |
| **Total** | $4,109,330$ |

Table 2. Performance of crs generator

| | |
|---|---|
| Setup (crs generation) | $5$ min $11$ sec, $\mathsf{crs}_p$ : $896$ MB, $\mathsf{crs}_V$ : $750$ B |
| Prover | $1$ min $59$ sec, $288$ B |
| verify | $< 6$ ms |

scratch ($\sim 28.000$ gates). Then they construct a large arithmetic circuit $C_{\mathsf{Pour}}$ that can verify all checks in the Pour transaction (excepts the one-time signature) with $4,100,000$ gates.

Running Setup (CRS generation) takes $5$ minutes $11$ second but it only needs to be run once and the size of keys needed for proof generation is $896$ MB and the size of keys for proof verification is $750$ B. Prover takes $1$ min $59$ sec, to generate a proof $\pi$ with size $288$ B and verifier takes less than $6$ ms to verify the proof. Reported experiments are done on a desktop machine with an Intel Core i7-4770 @3.40GHz and 16GB of RAM. Tab. 1 summarizes the number of gates needed in individual checks in circuit $C_{\mathsf{Pour}}$. One can see that totally the circuit has around 4.100.000 gates and the big part is required to check the validity of spent coins in a Merkele tree that each coin requires around 1.800.000 gates. Based on implementation reports in [BCG+14b], performance of CRS generation for zk-SNARK and other algorithms of Zcash are summarized in Tab. 2 and Tab. 3 [11].

Table 3. Perfornace of all algorithms

| | |
|---|---|
| Setup (crs generation) | $5$ min $12$ sec, $896$ MB params |
| Create Address | $326$ ms/add , $650$ B |
| Mint a coin | $23\ \eta$ s, $72$ B transaction |
| Pour (2-to-2) | $1$ min $59$ sec, $1$ KB transaction |
| Verify Transaction | $< 9$ ms transaction |
| Receive | $< 2$ ms transaction |

[11] We reported all implementation results based on the experiments reported in full version of Zerocahs paper available in [BCG+14b]. It is worth to mention that initial version of Zerocash, was using initial version of Pinocchio zk-SNARKs proposed by Ben-Sasson et al. [BCG+14a], but recently they optimized the system and decreased the number of gates in $C_{\mathsf{Pour}}$ and also started to use Groth's zk-SNARK [Gro16].

# 4 An Efficient Simulation Extractable zk-SNARK

Up to this section we considered Bitcoin and some privacy preserving coins including Zcash that uses zk-SNARKs to generate proof in pour transactions. Due to very efficient verification and succinct proofs of zk-SNARKs, recently they have appeared in various privacy-preserving applications including privacy-preserving smart contract systems such as Hawk [KMS+16] and Gyges [JKS16]. As we discussed in the introduction, majority of known zk-SNARKs [PHGR13, BCTV13, Gro16] are designed to guarantee *completeness*, *zero-knowledge* and (non-black-box) *knowledge soundness* that are defined in Sec. 2.7.1.

Currently the most efficient pairing-based zk-SNARK was proposed by Groth in Eurocrypt 2016 [Gro16]. Groth's zk-SNARK is constructed for Quadratic Arithmetic Programs (QAPs) and works over bilinear groups. While using zk-SNARKs in practical systems, their *knowledge soundness* property is not enough to guarantee non-malleability of proofs. For instance, in verification of some zk-SNARKs (e.g. Groth's zk-SNARK [Gro16]) the verifier checks a pairing equation as $A \bullet B = \cdots$, where $A$ and $B$ are proof elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ with prime orders. One can see that such verification equation will be satisfied also for new proof elements such as $A' = A^r$ and $B' = B^{\frac{1}{r}}$, for arbitrary $r \in \mathbb{Z}_p$. Due to this fact, zk-SNARKs that only guarantee knowledge-soundness cannot be deployed in many of practical applications straightforwardly [BCG+14a, KMS+16, JKS16, Bag19]. For instance, privacy-preserving crypocurrencies such as Zerocash that uses zk-SNARKs [BCTV13, Gro16] as a subroutine, takes extra steps to prevent malleability attacks in the SNARK proofs for *pour* transactions [BCG+14a]. Similarly, privacy-preserving smart contract systems [KMS+16, JKS16] show that knowledge-soundness of zk-SNARKs is not enough for their systems. Technically speaking, their system requires zk-SNARKs that satisfies Universal Composability [Can01] which is stronger security guarantee than knowledge-soundness achieved in majority of zk-SNARKs. *Simulation-knowledge soundness* which also is known as *simulation extractability* (defined in Def. 12), is an amplified version of knowledge-soundness that guarantees non-malleability of proofs.

As mentioned before, Groth's zk-SNARK [Gro16] is the most efficient pairing-based zk-SNARK in the CRS model, but it does not guarantee non-malleability of proofs. In Crypto 2017 Groth and Maller [GM17] have proposed the first SE zk-SNARK that is constructed for Square Arithmetic Programs (SAPs) and guarantees non-malleability of proofs. They also proved that a SE zk-SNARK requires at least two verification equations. Their scheme is constructed in the bilinear groups for Square Arithmetic Programs (SAPs) and achieves the lower bound in the number of verification equations that results smaller number of pairings. To verify a proof, the verifier needs to check two equations that contain with 5 pairings [GM17]. To guarantee non-malleability in proofs, their scheme removes one of the bilinear group generators from the CRS, which can create new challenges in some practical cases (e.g. in CRS generation by multi-party computation protocols [BCG+15, ABL+19] , or in achieving subversion security [BFS16, ABLZ17b, Fuc18]). Above all, Groth and Maller's zk-SNARK is constructed for arithmetic circuits with squaring gates that requires larger number of gates, as each multiplication gate requires two squaring gates ($a \times b = ((a+b)^2 - (a-b)^2)/4$). Implementations have confirmed that for many practical circuits Groth's zk-SNARK [Gro16] (described in Sec. 2) has considerably better efficiency than Groth and Maller's zk-SNARK [GM17]. Based on the currently available implementations in Libsnark library[12], for a Rank-1 Constraint System (R1CS) [13] instance,

[12] Available on `https://github.com/scipr-lab/libsnark/tree/master/libsnark/zk_proof_systems`

[13] The R1CS is the following natural NP-complete problem: given a vector $v \in \mathbb{F}^k$ and three matrices $A, B, C \in$

efficiency metrics of both schemes are compared in Tab. 4. For two mentioned zk-SNARK, we executed already written codes in Libsnark library on a Laptop with 2.50 GHz Intel Core i5-7200U CPU, with 16GB RAM, in single-threaded mode.

Table 4. Empirical performance of zk-SNARKs proposed by Groth and Maller [GM17], Groth [Gro16] and this thesis for arithmetic circuit satisfiability with an R1CS instance with $10^6$ constraints and $10^6$ variables, where $10$ are input variables. SE: Simulation Extractable, KS: Knowledge Soundness.

| SNARK | CRS size, time | Proof size, P time | Verification, V time | Security |
|---|---|---|---|---|
| [GM17] | 376 MB, 103 s | $2\mathbb{G}_1 + 1\mathbb{G}_2$, 120 s | 5 pairings, 2.3 ms | SE |
| [Gro16] | 196 MB, 75 s | $2\mathbb{G}_1 + 1\mathbb{G}_2$, 83 s | 3 pairings, 1.4 ms | KS |
| New scheme | 205 MB, 80.5 s | $3\mathbb{G}_1 + 2\mathbb{G}_2 + 1$BS, 90 s | 4 pairings, 2.0 ms | SE |

By considering efficiency comparisons in Tab. 4, we observe that Groth's zk-SNARK has better efficiency but only achieves knowledge-soundness, and Groth and Maller's zk-SNARK [GM17] ensures simulation extractability but with less efficiency. So an interesting research question can be raised whether we can achieve simulation extractability in Groth's scheme efficiently such that the new scheme will 1) work for QAPs 2) have both generators of bilinear groups in the CRS 3) have a comparable or even better efficiency than Groth and Maller's zk-SNARK.

**A variation of Groth's zk-SNARK.** In this thesis we address the questions discussed above and propose a variation of Groth's zk-SNARK that can achieve simulation extractability with minimal efficiency loss in practical cases.

To this end, we use the known OR technique and define a new language $\mathbf{L}'$ based on the language $\mathbf{L}$ in Groth's zk-SNARK that is inspired by the works of De Santis et al. [DDO+01] and Kosba et al. [KZM+15].

Defining a new language based on original language results some changes in algorithms of the original scheme. Evaluations show that in practical cases, new changes have minimal affects on the efficiency of original scheme which currently is the state-of-the-art. Strictly speaking, the verification of new scheme has two equations as the optimal case, and only adds 1 pairing to the verification of Groth's scheme. As a result, for smaller number of public inputs, verification of new scheme is dominated with 4 pairings which is less than 5 pairings in Groth and Maller's SE zk-SNARK [GM17]. Empirical analysis shows that, for the considered instance in Tab. 4, verification of new scheme takes $2.0$ milliseconds. In the proposed variation, the proof size will be extended by one element from $\mathbb{G}_1$, one element from $\mathbb{G}_2$ plus a 256-bit string, that totally will be 3 elements from $\mathbb{G}_1$, 2 elements from $\mathbb{G}_2$ and one 256-bit string, which for 128-bit security still it is less than 256 bytes. The prover should give a proof for a new circuit that has around $50 \times 10^3$ gates more than before, where in practical scenarios the overload is very small. I.e. Zerocash uses zk-SNARKs to give a proof for a circuit with approximately $2 \times 10^6$ multiplication gates[14]. In comparison with P running times in Tab. 4, prover of new scheme requires $90$ sec to generate a proof; particularly with smaller CRS in comparison with Groth and Maller's [GM17] scheme (with $205$ MB, instead of $376$ MB). Efficiency of the proposed variation is summarized in Tab. 4.

$\mathbb{F}^{m\times n}$, can one augment $v$ to $z\mathbb{F}^n$ such that $Az \circ Bz = Cz$? (We use "$\circ$" to denote the entry-wise product.)

[14]Their initial circuit had $\approx 4 \times 10^6$ gates, but recently they optimized the system and reduced the number of gates to $\approx 2 \times 10^6$, but still it is very larger than $\approx 50 \times 10^3$.

**Discussion and Related Works.** Among different NIZK arguments, zk-SNARKs are the most practically-interesting ones; because of their succinct proofs and very efficient verifications. But as majority of them guarantee knowledge-soundness by default, that is vulnerable to the man-in-the-middle attacks, so they cannot be deployed directly in practical systems. Actually, in constructing large cryptographic systems, this issue can cause misuse from non-expert users. To address this, recently constructing efficient SE zk-SNARKs, that by default can guarantee non-malleability of proofs, has gotten more attention [GM17, BG18b, KLO19, Lip19]. In [BG18b], Bowe and Ariel also proposed a variation of Groth's scheme that achieves simulation extractability but in the Random Oracle (RO) model. In their variation, the proof consists of 3 elements from $\mathbb{G}_1$ and 2 elements from $\mathbb{G}_2$, and verification is dominated by 5 pairings. A good point about their case is that they keep the language as original one, and add some computation to the proof generation and verification with relying on a random oracle that returns group elements [15]. Implementing such random oracle might cause some challenges in practice. But as Groth's zk-SNARK is constructed and proven in the CRS model, so we are interested to achieve simulation extractability in the same model using more practical cryptographic primitives.

## 4.1 A Variation of Groth's zk-SNARK

As briefly discussed in the introduction, Groth's zk-SNARK [Gro16] guarantees knowledge-soundness which is weaker than simulation extractability. Technically speaking, knowledge-sound proofs are not secure against man-in-the-middle attacks. In this section, we present a variation of Groth's zk-SNARK which can achieve (non-black-box) simulation extractabilitythat can guarantee non-malleability of the proofs.

### 4.1.1 New Construction

In construction of new variation, we define a new language $\mathbf{L}'$, using an OR technique [DDO+01, KZM+15], which combines original language $\mathbf{L}$ in Groth's zk-SNARK with a commitment scheme which commits to a secret randomness as a key for a pseudorandom function. Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a one-time signature scheme and $(\mathsf{Com}, \mathsf{ComVerify})$ be a perfectly binding commitment scheme.

Given the language $\mathbf{L}$ with the corresponding NP relation $\mathbf{R_L}$, we define a new language $\mathbf{L}'$ such that $((\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, s, r)) \in \mathbf{R}_{\mathbf{L}'}$ iff:

$$\big((\mathsf{x}, \mathsf{w}) \in \mathbf{R_L} \vee (\mu = f_s(\mathsf{pk}_{\mathsf{Sign}}) \wedge \rho = \mathsf{Com}(s, r))\big)\,,$$

where $\{f_s : \{0,1\}^* \to \{0,1\}^n\}_{s \in \{0,1\}^n}$ is a pseudo-random function family. The intuition for a pseudo-random function $f_s(\cdot)$ is that without the knowledge of the key $s$, $f_s(\cdot)$ behaves like a true random function. However, given $s$, one can compute $f_s(\cdot)$ easily. In new language $\mathbf{L}'$, for a statement-witness pair to be valid, either a witness for $\mathbf{R_L}$ is provided (by honest prover) or an opening to $\rho$ together with the value of $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$ is provided (by simulator), where $s$ is the open value of $\rho$ (in CRS). One may note that in order for a statement to pass the verification without a valid witness, the prover must generate $f_s(\mathsf{pk}_{\mathsf{Sign}})$ without the knowledge of $s$ (thus breaking the pseudo-random function $f_s$).

[15] Intuitively, some part of their changes play the role of a one-time secure signature scheme, but add two pairings to the verification of original scheme.

By considering new language $\mathbf{L}'$, zk-SNARK of Groth for the relation $\mathbf{R}$ constructed from PPT algorithms $(\mathsf{KGen}, \mathsf{P}, \mathsf{V}, \mathsf{Sim})$ can be lifted to a simulation-extractable zk-SNARK $\Psi'$ with PPT algorithms $(\mathsf{KGen}', \mathsf{P}', \mathsf{V}', \mathsf{Sim}')$ as described in Fig. 5. To simplify the description, we assume $\mathsf{Com}$ takes exactly $n$ random bits as randomness and that the witness for original language $\mathbf{L}$ is exactly $n$ bits; it is straight forward to adapt the proof when they are of different lengths [KZM+15]. Note that in the simulation-extractable zk-SNARK $\Psi'$, the algorithms of original scheme will be executed for a new arithmetic circuit which encodes new language $\mathbf{L}'$ and has slightly larger number of gates. Namely, CRS generation algorithm of Groth's zk-SNARK will be executed with a new QAP instance that has larger parameters; $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$. Similarly prover of new variation will execute prover of Groth's zk-SNARK with a new arithmetic circuit that has larger number of gates; namely $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, z_0, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, z_1, z_2))$, where $z_1$ and $z_2$ play the role of witnesses $s$ and $r$ for prover.

**CRS generator** $\mathsf{KGen}'(\mathbf{R}_{\mathbf{L}}, \xi)$**:** Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s, r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, r)))$; where $\mathsf{tr}'$ is new simulation trapdoor and $\xi$ is the auxiliary information returned by relation generator $\mathsf{BGgen}(1^n)$, and is given as an input to the honest parties.

**Prover** $\mathsf{P}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \mathsf{w})$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \rho)$; abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$; generate $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; sample $z_0, z_1, z_2 \leftarrow_u \{0,1\}^n$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, z_0, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, z_1, z_2))$ using the prover of Groth's scheme; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, z_0, \pi))$; and return $\pi' := (z_0, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.

**Verifier** $\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi')$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \rho)$ and $\pi' := (z_0, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; abort if $\mathsf{Vf}(\mathsf{pk}_{\mathsf{Sign}}, (\mathsf{x}, z_0, \pi), \sigma) = 0$; call the verifier of Groth's scheme $\mathsf{V}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, z_0, \mathsf{pk}_{\mathsf{Sign}}, \rho), \pi)$ and abort if it outputs $0$.

**Simulator** $\mathsf{Sim}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{tr}', \mathsf{x})$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \rho)$ and $\mathsf{tr}' := (\mathsf{tr}, (s, r))$; generate $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; and output $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.

Figure 5. A variation of Groth's zk-SNARK.

### 4.1.2 Security Proofs

In the rest we present security proofs of the proposed scheme.

**Theorem 1** (Completeness)**.** *The variation of Groth's zk-SNARK described in Sec. 4.1.1, guarantees completeness.*

*Proof.* In new scheme internal computations of $\mathsf{P}$ and $\mathsf{V}$ are the same as original one, except few extra efficient computations. Precisely, $\mathsf{P}$ needs to do the computation for a new instance that has slightly larger size (e.g. $n = n_{\mathsf{old}} + n_{\mathsf{new}}$, where $n_{\mathsf{new}}$ is the number of multiplication gates added to the old circuit) and sign the proof and statement whit a one-time secure signature scheme. So following the completeness of original scheme, and the fact that the deployed signature scheme is complete, which means $\mathsf{Vf}(\mathsf{pk}_{\mathsf{Sign}}, m, \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{Sign}})) = 1$, one can conclude that the modified construction satisfies *completeness*. □

**Theorem 2** (Zero-Knowledge)**.** *The variation of Groth's zk-SNARK described in Sec. 4.1.1, guarantees computational zero-knowledge.*

*Proof.* We write a series of hybrid experiments which start from an experiment with the simulator and ends with an experiment that uses the real prover. We show that all experiments consecutively are indistinguishable. Changes between successive experiments are shown with highlights. Recall that Groth's zk-SNARK guarantees perfect zero-knowledge and the simulator of the modified scheme is expressed in Fig. 5. Now consider the following experiments,

$\underline{\mathsf{G}_1}$(simulator):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s, r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x}, \mathsf{w})$: Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$; $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), \mathsf{tr}')$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathsf{crs}')$

$\underline{\mathsf{G}_2}$(separate secret key of pseudo random function):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s', s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s', r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, s', r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x}, \mathsf{w})$: Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$; generate $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathsf{crs}')$

**Lemma 1.** *If the underlying commitment scheme is computationally hiding, then for two experiments* $\mathsf{G}_2$ *and* $\mathsf{G}_1$ *we have* $\Pr[\mathsf{G}_2] \approx \Pr[\mathsf{G}_1]$.

*Proof.* Computationally hiding property of a commitment scheme implies that $\mathsf{Com}(m_1, r)$ is computationally indistinguishable from $\mathsf{Com}(m_2, r)$. So this property straightforwardly results the lemma. □

$\underline{\mathsf{G}_3}$(replace pseudo random function):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s'$, ~~$s$~~, $r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s', r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (~~s~~, s', r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x}, \mathsf{w})$: Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$; $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu \leftarrow_u \{0,1\}^n$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s', r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathsf{crs}')$

**Lemma 2.** *If the pseudo random function* $f_s(\cdot)$ *is secure and the underlying one-time signature scheme is unforgeable,we have* $\Pr[\mathsf{G}_3] \approx \Pr[\mathsf{G}_2]$. □

*Proof.* By considering that the signature scheme is secure, we note that the generated $\mathsf{pk}_{\mathsf{Sign}}$ is unique with overwhelming probability, otherwise it would break the one-time signature scheme. Additionally, we can replace the pseudo random function $f_s(\cdot)$ with a true random function that will result $\mathsf{G}_4$. By considering unique values of $\mathsf{pk}_{\mathsf{Sign}}$ and indistinguishability of output of $f_s(\cdot)$ and truly random function, one can conclude the claim. □

$\underline{\mathsf{G}_4}$(prover):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s', r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s', r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s', r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x}, \mathsf{w})$: Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$; $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu \leftarrow_u \{0,1\}^n$ ($\mu$ plays the role of $z_0$ in Fig. 5); sample $z_1, z_2 \leftarrow_u \{0,1\}^n$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, z_1, z_2))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathsf{crs}')$

**Lemma 3.** *If Groth's zk-SNARK guarantees zero-knowledge, then we have* $\Pr[\mathsf{G}_4] \approx \Pr[\mathsf{G}_3]$.

*Proof.* The last experiment exactly models the real prover of construction in Fig. 5, and as already Groth's scheme guarantees zero-knowledge, so one can conclude that the real proof (generated by prover) in experiment $\mathsf{G}_4$ is indistinguishable from the simulated proof (generated by simulator) in $\mathsf{G}_3$. Intuitively this is because all new elements added to the new construction are chosen randomly and independently. □

This results that the construction proposed in Fig. 5, guarantees computationally zero-knowledge. □

**Theorem 3** ((Non-Black-Box) Simulation Extractability)**.** *The variation of Groth's zk-SNARK, described in Sec. 4.1.1, guarantees (non-black-box) simulation extractability.*

*Proof.* Similarly, we write a sequence of hybrid experiments and finally show that the success probability in the last game is negligible. Recall that Groth's scheme is proven to achieve knowledge-soundness. For any NUPPT $\mathcal{A}$ and extractor $\mathsf{Ext}_{\mathcal{A}}$, consider the following game which models the definition of simulation-extractability,
$\underline{\mathsf{G}_1}$(main experiment):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s, r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x})$:
  $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, (s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathsf{crs}')$.
- Parse $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; $\mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}', \mathsf{x}, \pi, \xi)$.
- Return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi') = 1) \wedge ((\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}})$; where $Q$ shows the set of statement-proof pairs generated by $\mathsf{O}(\mathsf{x})$.

$\underline{\mathsf{G}_2}$(relaxing the return checking):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s, r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x})$: $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{w}, (s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathsf{crs}')$.
- Parse $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; $\mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}', \mathsf{x}, \pi, \xi)$.
- Return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi') = 1) \wedge (\mathsf{pk}_{\mathsf{Sign}} \notin \mathcal{PK}) \wedge (\mu = f_s(\mathsf{pk}_{\mathsf{Sign}}))$; where $Q$ is the set of statement-proof pairs and $\mathcal{PK}$ is the set of signature verification keys, both generated by $\mathsf{O}(\mathsf{x})$.

**Lemma 4.** *If the underlying one-time signature scheme is strongly unforgeable, and Groth's scheme guarantees knowledge-soundness, then* $\Pr[\mathsf{G}_2] \leq \Pr[\mathsf{G}_1] + \mathsf{negl}(n)$.

*Proof.* We note that if $(\mathsf{x}, \pi') \notin Q$ and "$\mathsf{pk}_{\mathsf{Sign}}$ from $(\mathsf{x}, \pi')$, has been generated by $\mathsf{O}(\cdot)$", then the $(\mathsf{x}, \mu, \pi)$ is a valid message/signature pairs. Therefore by unforgeability of the signature scheme, we know that $(\mathsf{x}, \pi) \notin Q$ and "$\mathsf{pk}_{\mathsf{Sign}}$ has been generated by $\mathsf{O}(\cdot)$" happens with negligible probability, which allows us to focus on $\mathsf{pk}_{\mathsf{Sign}} \notin \mathsf{pk}$.

Now, due to knowledge-soundness (there is an extractor $\mathsf{Ext}_{\mathcal{A}}$ that can extract unique witness from successful $\mathcal{A}$) of the original scheme, the extracted $\mathsf{w}$ is unique for all valid witnesses. Further, if some witness is valid for $\mathbf{L}'$ and $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}_{\mathbf{L}}$, so we conclude it must be the case that there exists some $s'$, such that $\rho$ is valid commitment of $s'$ and $\mu = f_{s'}(\mathsf{pk}_{\mathsf{Sign}})$, which by perfectly binding property of the commitment scheme, it implies $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$. □

$\underline{\mathsf{G}_3}$(simulator):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s, r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x})$: $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathsf{crs}')$.
- Parse $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; $\mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}', \mathsf{x}, \pi, \xi)$.
- Return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi') = 1) \wedge (\mathsf{pk}_{\mathsf{Sign}} \notin \mathsf{pk}) \wedge (\mu = f_s(\mathsf{pk}_{\mathsf{Sign}}))$; where $Q$ is the set of statment-proof pairs and pk is the set of signature verification keys, both generated by $\mathsf{O}(\mathsf{x})$.

**Lemma 5.** *If Groth's zk-SNARK guarantees zero-knowledge, then for two experiments* $\mathsf{G}_3$ *and* $\mathsf{G}_2$*, we have* $\Pr[\mathsf{G}_3] \leq \Pr[\mathsf{G}_2] + \mathsf{negl}(n)$.

*Proof.* As the original scheme ensures (perfect) zero-knowledge, so it implies no polynomial time adversary can distinguish a proof generated by the simulator from a proof that is generated by the prover. So, as we are running in polynomial time, thus two experiments are indistinguishable. □

$\underline{\mathsf{G}_4}$(separating secret key of pseudo random function):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s', s, r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s', r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (s, s', r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x})$: $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = f_s(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathsf{crs}')$.
- Parse $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; $\mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}', \mathsf{x}, \pi, \xi)$.
- Return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi') = 1) \wedge (\mathsf{pk}_{\mathsf{Sign}} \notin \mathsf{pk}) \wedge (\mu = f_s(\mathsf{pk}_{\mathsf{Sign}}))$; where $Q$ is the set of statment-proof pairs and $\mathsf{pk}$ is the set of signature verification keys, both generated by $\mathsf{O}(\mathsf{x})$.

**Lemma 6.** *If the commitment scheme used in the CRS generation is computationally hiding, then* $\Pr[\mathsf{G}_4] \leq \Pr[\mathsf{G}_3] + \mathsf{negl}(n)$.

*Proof.* Computationally hiding of a commitment scheme implies that $\mathsf{Com}(m_1, r)$ and $\mathsf{Com}(m_2, r)$ are computationally indistinguishable, as in this lemma. □

$\underline{\mathsf{G}_5}$(replace pseudo random function $f_s(\cdot)$ with true random function $F(\cdot)$):

- *Setup:* Sample $(\mathsf{crs}||\mathsf{tr}) \leftarrow \mathsf{KGen}(\mathbf{R}_{\mathbf{L}'}, \xi)$; $s'$, ~~$s$~~, $r \leftarrow_u \{0,1\}^n$; $\rho := \mathsf{Com}(s', r)$; and output $(\mathsf{crs}'||\mathsf{tr}') := ((\mathsf{crs}, \rho)||(\mathsf{tr}, (\not{s}, s', r)))$; where $\mathsf{tr}'$ is simulation trapdoor.
- *Define function* $\mathsf{O}(\mathsf{x})$: $(\mathsf{pk}_{\mathsf{Sign}}, \mathsf{sk}_{\mathsf{Sign}}) \leftarrow \mathsf{KGen}(1^n)$; set $\mu = F(\mathsf{pk}_{\mathsf{Sign}})$; generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi, \mathsf{crs}, (\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho), (\mathsf{tr}||(s, r)))$; sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sign}}, (\mathsf{x}, \mu, \pi))$; return $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$.
- $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathsf{crs}')$.
- Parse $\pi' := (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$; $\mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{crs}', \mathsf{x}, \pi, \xi)$.
- Return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}'(\mathbf{R}_{\mathbf{L}}, \xi, \mathsf{crs}', \mathsf{x}, \pi') = 1) \wedge (\mathsf{pk}_{\mathsf{Sign}} \notin \mathsf{pk}) \wedge (\mu = F(\mathsf{pk}_{\mathsf{Sign}}))$; where $Q$ is the set of statment-proof pairs and $\mathsf{pk}$ is the set of signature verification keys, both generated by $\mathsf{O}(\mathsf{x})$.

**Lemma 7.** *If the underlying truly random function* $F(\cdot)$ *is secure, then* $\Pr[\mathsf{G}_4] \leq \Pr[\mathsf{G}_5]$.

*Proof.* By assuming function $F(\cdot)$ is secure, we can conclude no polynomial time adversary can distinguish an output of the true random function $F(\cdot)$ from an output of the pseudo random function $f_s(\cdot)$. Indeed, experiment $\mathsf{G}_5$ can be converted to an adversary for the game of a *true random function*. □

For experiment $\mathsf{G}_5$, we have $\Pr[\mathsf{G}_5] \leq 2^{-n}$.

*Proof.* From verification we know $\mathsf{pk}_{\mathsf{Sign}} \notin \mathsf{pk}$, therefore $F(\mathsf{pk}_{\mathsf{Sign}})$ has not been queried already. Thus, we will see $F(\mathsf{pk}_{\mathsf{Sign}})$ as a newly generated random string independent from $\mu$, which implies adversary only can guess. □

This completes proof of the main theorem. □

## 4.2 Instantiation and Efficiency Evaluation

As we observed in Sec. 4.1.1, defining the new language $\mathbf{L}'$ led to some changes in the algorithms of original scheme. In this section, we discuss how efficient can be such changes (described in Fig. 5). We first discuss how the used primitives can be instantiated and then evaluate efficiency of the whole protocol.

Recall that in result of new changes, one needed a pseudo random function, a commitment scheme and a one-time secure signature scheme. In similar practical cases, both pseudo random function and commitment scheme are instantiated using an efficient SHA-256 circuit that has around $\approx 25 \times 10^3$ multiplication gates for one block (512-bit input) [BCG+14a, KMS+16][16]. We instantiate the signature scheme with Boneh and Boyen's signature [BB08] where its setup phase, signing and verification for message $m$ can be summarized as follows,

- **Setup:** Given system parameters for a prime-order bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, randomly selects $\mathsf{sk} \leftarrow_u \mathbb{Z}_p^*$, and computes $\mathsf{sk} \cdot [1]_1$ and returns $(\mathsf{pk}, \mathsf{sk}) := ([\mathsf{sk}]_1, \mathsf{sk})$.
- **Signing:** Given system parameters, a secret key $\mathsf{sk}$, and a message $m$, computes $[\sigma]_2 = [1/(m + \mathsf{sk})]_2$ and returns $[\sigma]_2$ as the signature.
- **Verification:** Given a public key $\mathsf{pk}$, a message $m$, and a signature $\sigma$, verifies if $[m + \mathsf{sk}]_1 \bullet [1/(m + \mathsf{sk})]_2 = [1]_T$,

where $\bullet$ denotes the pairing operation. In our case, we use the same bilinear group as in the main scheme and $m$ would be the hash[17] (e.g. with SHA-256) of concatenations of the proof elements with the statement, i.e. $m := H(\mathsf{x}||z_0||\pi)$. As it can be seen, the scheme generates a single-element signature from $\mathbb{G}_2$, its public key is an element from $\mathbb{G}_1$, and above all its verification only requires one pairing. Note that $[1]_T$ can be preprocessed and shared in the CRS.

So by considering the above instantiation, new proof $\pi' = (\mu, \pi, \mathsf{pk}_{\mathsf{Sign}}, \sigma)$ will be as $\pi' = (\mu, \pi, [\mathsf{sk}]_1, [1/(m + \mathsf{sk})]_2)$ where from original scheme $\pi = ([a]_1, [b]_2, [c]_1)$, and $\mu$ is an output of the pseudo random function $f_s(\cdot)$, which is instantiated with SHA-256 hash function [KMS+16]. As a result, the proof in new scheme will be 3 elements from $\mathbb{G}_1$, 2 elements from $\mathbb{G}_2$ and one 256-bit string. Consequently, new changes add only one pairing to the verification of original scheme. To the best of our knowledge, this is the first simulation-extractable zk-SNARK in the CRS model with verification dominated by 4 pairings.

Next, we empirically analyse efficiency of the proposed scheme from different perspectives. Tab. 5 summarizes asymptotic and empirical performance of new scheme and two

[16] It has 25.538 gates in the `xjsnark` library, `https://github.com/akosba/xjsnark`.

[17] As shown in [BB08], by taking hash of input message the signature scheme can be used to sign arbitrary messages in $\{0,1\}^*$. To do so, a collision resistant hash function $H : \{0,1\}^* \to \{0, \ldots, 2^b\}$ such that $2^b < p$ is sufficient [BB08].

Table 5. An efficiency comparison of new scheme with Groth's [Gro16] and Groth and Maller's [GM17] zk-SNARKs for arithmetic circuit satisfiability with $m_0$ elements instance, $m$ wires, $n$ multiplication gates. In [GM17], $n$ multiplication gates translate to $2n$ squaring gates. Implementations (Implem.) are done on a Laptop with 2.50 GHz Intel Core i5-7200U CPU, with 16GB RAM, in single-threaded mode, for an R1CS instance with $n = 10^6$ constraints and $m = 10^6$ variables, of which $m_0 = 10$ are input variables. $\mathbb{G}_1$ and $\mathbb{G}_2$: group elements, $E$: exponentiations, $P$: pairings. In the new scheme, the statement contains $(\mathsf{x}, \mu, \mathsf{pk}_{\mathsf{Sign}}, \rho)$ which has 3 new elements $(\mu, \mathsf{pk}_{\mathsf{Sign}}, \rho)$, so $m'_0 = m_0 + 3$. All asymptotic analysis of new scheme are done based on our particular instantiation of commitment and pseudo random function. So, as new changes add $\approx 50 \times 10^3$ multiplication gates to $n$ and $m$, so $n' = n + 50.000$ and $m' = m + 50.000$.

| SNARK | CRS size & gen. time | Proof size | Comp. & time of P | V | Sec. |
|---|---|---|---|---|---|
| [GM17] | $m + 4n + 5$ $\mathbb{G}_1$ | 2 $\mathbb{G}_1$ | $m + 4n - m_0$ $E_1$ | $m_0$ $E_1$ | |
| & | $2n + 3$ $\mathbb{G}_2$ | 1 $\mathbb{G}_2$ | $2n$ $E_2$ | 5 $P$ | SE |
| Implem. | 376 MB, 103 sec | 127 bytes | 120 sec | 2.3 ms | |
| [Gro16] | $m + 2n - m_0$ $\mathbb{G}_1$ | 2 $\mathbb{G}_1$ | $m + 3n - m_0 + 3$ $E_1$ | $m_0$ $E_1$ | |
| & | $n + 3$ $\mathbb{G}_2$ | 1 $\mathbb{G}_2$ | $n + 1$ $E_2$ | 3 $P$ | KS |
| Implem. | 196 MB, 75 sec | 127 bytes | 83 sec | 1.4 ms | |
| Sec. 4 | $m' + 2n' - m'_0 + 5$ $\mathbb{G}_1$ | $3\mathbb{G}_1$ + $2\mathbb{G}_2$ | $m' + 3n' - m'_0 + 4$ $E_1$ | $m'_0 + 1$ $E_1$ | |
| & | $n' + 3$ $\mathbb{G}_2$ | 1 bit string | $n' + 2$ $E_2$ | 4 $P$ | SE |
| Implem. | 205 MB, 80.5 sec | 254 bytes | 90.1 sec | 2.0 ms | |

zk-SNARKs proposed by Groth's [Gro16] and Groth and Maller's [GM17]. Implementations of Groth's [Gro16] and Groth and Maller's [GM17] zk-SNARKs are available in `libsnark` library [BCTV13] [18], so similarly implementation of new scheme is done in the same library. In Tab. 5, all implementation results are reported for the same R1CS instance.

[18] Available on `https://github.com/scipr-lab/libsnark`

# 5 Conclusion

Beside very nice features that blockchain technology and its by-products, such as digital currencies, provide, guaranteeing strong privacy and security of end-users always has been one of main priorities for society and researchers of the area.

In the first part of the thesis, after a short summary of some cryptocurrencies such as Bitcoin, Monero and Zerocoin, we reviewed Zerocash [BCG$^+$14a] with details which is one of the most well-known cryptocurrencies that is proposed to guarantee users' privacy. Zerocash does not reveal any information about user identities, source or destination addresses, and the account of transferred or spent coins. To have a comprehensive review on Zerocash, we first had an overview of the building blocks of the coin, including PRFs, commitment schemes, hash functions, one-time signature schemes, public-key encryption schemes, and zk-SNARKs. Then we discussed step-by-step the construction of the coin along with the main intuitions behind each step. We observe that the main tool behind indistinguishability properties of the coin is a zk-SNARK that allows a user (a prover) to give very short (succinct) proofs in *pour* transactions and more importantly these proofs can be verified by a low-power verifier very efficiently (in less than a second).

In the second part of the thesis, due to the importance of zk-SNARKs in various privacy-preserving applications, we focused on construction of an efficient simulation-extractable zk-SNARK. More precisely, we presented a new variation of Groth's 2016 zk-SNARK [Gro16] that currently is the most efficient pairing-based scheme. The main difference between the proposed variation and the original one is that unlike the original version, new variation guarantees non-malleability of generated proofs. In the new variation, we used an efficient OR construction to define a new language $\mathbf{L}'$ from the language $\mathbf{L}$ in the original scheme, that led to some changes in the algorithms of original scheme. Analysis and implementation results showed that in practical scenarios, new changes have minimal effect on the efficiency of original scheme which currently is the most efficient pairing-based zk-SNARK in the CRS model [Gro16]. Precisely speaking, evaluations showed that for arithmetic circuits with larger than $\approx 50 \times 10^3$ multiplication gates, the proposed SE zk-SNARK outperforms Groth and Maller's simulation-extractable zk-SNARK [GM17]. We emphasize that in current real-life systems that use zk-SNARKs, their underlying arithmetic circuits have much larger number of gates than $50 \times 10^3$. For instance, in Zerocash cryptocurrency [BCG$^+$14a] their current circuit for *pour* transactions has $2 \times 10^6$ multiplication gates; or similarly in Hawk smart contract system [KMS$^+$16], their circuit for *finalize* operation in an auction with 50 bidders has around $4 \times 10^6$ multiplication gates. In comparison with Groth and Maller's SE zk-SNARK [GM17], however proof of new scheme is extended slightly, but still its total size is less than 256 bytes for 128-bit security; and importantly its verification is dominated by smaller number of pairings that allows very efficient verification.

# References


[AB19] Shahla Atapoor and Karim Baghery. Simulation extractability in Groth's zk-SNARK. Cryptology ePrint Archive, Report 2019/641, 2019. http://eprint.iacr.org/2019/641.

[ABL+19] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-secure CRS generation for SNARKs. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 99–117. Springer, Heidelberg, July 2019.

[ABLZ17a] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, pages 3–33, 2017.

[ABLZ17b] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.

[Bag19] Karim Baghery. On the efficiency of privacy-preserving smart contract systems. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 118–136. Springer, Heidelberg, July 2019.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 566–582, 2001.

[BCG+14a] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCG+14b] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. Cryptology ePrint Archive, Report 2014/349, 2014. http://eprint.iacr.org/2014/349.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Erratum: Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013.

[BCTV13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. `http://eprint.iacr.org/2013/879`.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016.

[BG18a] J Benet and N Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, 2018.

[BG18b] Sean Bowe and Ariel Gabizon. Making groth's zk-snark simulation extractable in the random oracle model. *IACR Cryptology ePrint Archive*, 2018:187, 2018.

[BGM+18] Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 34–65, 2018.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 324–356, 2017.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 61–76, 2002.

[CMP+19] Jean-Paul Calderone, Ramakrishnan Muthukrishnan, Paige Peterson, Liz Steininger, and Chris Wood. P4:private periodic payment protocol. pages 1–23, 2019.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001.

[DFKP13] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *PETShop'13, Proceedings of the 2013 ACM Workshop on Language Support for Privacy-Enhancing Technologies, Co-located with CCS 2013, November 4, 2013, Berlin, Germany*, pages 27–30, 2013.

[FKP15] Michael Fleder, Michael S. Kester, and Sudeep Pillai. Bitcoin transaction graph analysis. *CoRR*, abs/1502.01657, 2015.

[FMMO18] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. *IACR Cryptology ePrint Archive*, 2018:990, 2018.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 181–200. Springer, Heidelberg, April 2007.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310, 2015.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 291–323, 2017.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[JKS16] Ari Juels, Ahmed E. Kosba, and Elaine Shi. The ring of Gyges: Investigating the future of criminal smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 283–295. ACM Press, October 2016.

[KCA+18] Aggelos Kiayias, Michele Ciampi, Behzad Abdolmaleki, Karim Baghery, Janno Siim, Luisa Siniscalchi, Ivan Visconti, Michal Zajac, and Peter Gazi. State of the art of cryptographic ledgers. 2018.

[KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, pages 153–173, 2017.

[KLO19] Jihye Kim, Jiwon Lee, and Hyunok Oh. Qap-based simulation-extractable SNARK with a single verification. *IACR Cryptology ePrint Archive*, 2019:586, 2019.

[KMS+16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press, May 2016.

[Kob98] Neal Koblitz. An elliptic curve implementation of the finite field digital signature algorithm. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 327–337, 1998.

[KZM+15] Ahmed E. Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T.-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs. Technical Report 2015/1093, November 10, 2015. `http://eprint.iacr.org/2015/1093`, last accessed version from 9 Apr 2017.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

[Lip19] Helger Lipmaa. Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612, 2019. `http://eprint.iacr.org/2019/612`.

[Mer] Ralph Charles Merkle. *Secrecy, authentication and public key systems*. PhD thesis, Stanford University.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 397–411, 2013.

[MS18] Izaak Meckler and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. 2018.

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, 2018.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

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

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 643–673, 2017.

[RH11] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, pages 1318–1326, 2011.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, pages 6–24, 2013.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 552–565, 2001.

[ST99] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash extended abstract. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 555–572, 1999.

# II. Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, **Shahla Atapoor**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   **On Privacy Preserving Blockchains and zk-SNARKs**,

   supervised by Helger Lipmaa, Janno Siim and Karim Baghery.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Shahla Atapoor
**14.08.2019**