Abelian (ABEL) – A Quantum-Resistant Cryptocurrency Balancing Privacy and Accountability

Alice, Bob, Eve, and λ

February 18, 2022
Version 1.0

Abstract. We propose to develop a new cryptocurrency Blockchain platform called Abelian (ABEL). It will employ post-quantum cryptographic primitives for ensuring the security against quantum attacks. The platform will also support three different levels of privacy, which we believe will be useful for different usage scenarios, spanning from the conventional use as of most existing cryptocurrencies are supporting to some highly private usage scenarios that a few current cryptocurrencies are promoting. We also propose a new privacy level, namely Full Privacy with Accountability, which will accommodate regulatory needs and enterprise requirements. Users can choose which privacy levels they would like to have for each transaction to be carried out. We would like to provide the cryptocurrency community a new choice and a new platform with a set of tools, which will empower other innovators to build their next disruptive technologies, applications, and businesses using the quantum-resistant infrastructure supported by the Abelian platform.

In this whitepaper, we propose a construction approach and a suite of quantum-

resistant cryptographic techniques for building the Abelian Blockchain platform. We formalize it by proposing definitions and security models with the aim of building a quantum-resistant cryptocurrency platform from provably secure cryptographic schemes.

Our mission is to open up a new page on the evolution of cryptocurrency through enabling the first quantum-resistant cryptocurrency with the options of supporting fully private transactions and accountability. We strongly believe that the success of this project will not only bring to the crypto community a quantum-resistant cryptocurrency with full privacy and accountability, but also the key quantum-resistant cryptographic techniques which will benefit other existing cryptocurrencies including Bitcoin.

Let us all work together, and let us invest today for tomorrow's sustainable and prosperous quantum-resistant cryptocurrency ecosystem with the options of full privacy and accountability.

"When cryptography is outlawed, bayl bhgynjf jvyy unir cevinpl." – John Perry Barlow

Contents

1 Introduction

Bitcoin is a cryptocurrency which literally has changed the world. In one's lifetime, it is not often to experience something like this, a global phenomenon that is both so disruptive and so intriguing. It gives us huge and profound impact on how we transact in the financial world. In less than 15 years, Bitcoin, and the term "cryptocurrency" have evolved from being known by just a few technical geeks deep inside the cyber-world to literally a household name.

Bitcoin is not the first attempt to create a global, decentralized, and secure cryptocurrency. But its underpinning technology conglomerate, called Blockchain, is giving the financial world an innovative and disruptive impact. Blockchain is a genius creation by putting together some well-known primitives, and it is so unique and groundbreaking in both technological contributions as well as application innovativeness.

Blockchain is a magnificent inter-disciplinary technological work, which nicely intertwines some well-studied computer science algorithms and cryptographic techniques to create an ingenious cryptocurrency, which has changed the rules of the global financial market. It is also demonstrating the power of reshaping the economic dominance in the world. The impact and potential of cryptocurrencies are profound, long-lasting and fundamental.

Blockchain is based on the concept of distributed and duplicated ledgers among a tremendously broad collection of intelligent rational "node-runners", which are contending yet cooperating. Blockchain applies a consensus algorithm over a peer-to-peer computer network together with digital signatures under individual crypto-wallet owners' private keys for ensuring some crucial functionalities of a virtual ledger without a centralized moderator. Those crucial functionalities include double spending prevention, time-stamping, and immutability.

Elliptic Curve Cryptography (ECC) is in the core of almost all the current cryptocurrencies for digital signatures, which ensure the ownership of the crypto-

coins, non-repudiation, and transaction integrity. ECC is an invention in the 80s of the last century and the security of ECC relies on the difficulty of computing the discrete logarithm problem over an underlying elliptic curve finite field, or a related Decisional or Computational Diffie-Hellman problem. These mathematical problems are believed to be intractable for our conventional computers while may no longer be the case in front of quantum computers which behave according to laws in quantum mechanics.

When ECC's underlying mathematical problems are no longer considered intractable, for example, against quantum attacks, namely, solving the underlying mathematical problems using quantum computers with some quantum algorithms, the ECC-based digital signatures may become forgeable. The cryptocurrency's fundamental principle, according to Nakamoto's Bitcoin paper, quote *"transactions that are computationally impractical to reverse,"* will collapse, so is the security of the cryptocurrency itself.

The theme of this project is to build a quantum-resistant cryptocurrency with the options of supporting privacy and accountability for individual crypto-coin transactions. By privacy, we mean the protection of the sender and receiver's identities or wallet addresses of a crypto-coin transaction. We also want to protect the transaction amount from prying eyes. By accountability, we consider some practical scenarios that an authority may be able to open up the identities and the transaction amount while the transaction will remain anonymous to all other participants.

In addition to designing, prototyping, developing and deploying a full-fledged quantum-resistant cryptocurrency called Abelian (ABEL), we are also hoping to build up an ecosystem for ABEL. An ABEL community will be built up with developers all over the world to build a more secure and private ABEL ecosystem. Side chains, smart contracts, and inter-operability will be developed for supporting various DeFi, Metaverse, web3 applications and initiatives.

The design idea of Abelian is inspired by CryptoNote, a linkable ring signature

based system. Besides making Abelian a cryptocurrency which is secure against quantum attacks, we aim to support a higher level of privacy protection to all users, also against quantum attacks. Furthermore, we also consider accountability. Accountability refers to the ability to accommodate an authority's needs to open up the privacy protection of the transactions for being able to trace and link the flow of coins over wallet addresses, and also the transaction amount. Accountability is particularly useful when ABEL is used by some organization or consortium as their internal-use-only tokens while the authority of the organization or consortium should have the rights to identify and audit the transactions. Accountability will also be useful for regulatory bodies to perform monitoring duties on the cryptocurrency market.

In this project, we will make all the design documents and codes open source, that will encourage the rapid growth of the Abelian community. This will also help encourage collaboration with other technology partners for this new cryptocurrency. Our mission is to open up a new page on the evolution of cryptocurrency through enabling the first quantum-resistant cryptocurrency with the options of supporting fully private transactions and accountability. We strongly believe that the success of this project will not only bring to the crypto community a quantum-resistant cryptocurrency, but also the key quantum-resistant cryptographic techniques which will benefit all the existing cryptocurrencies including Bitcoin.

Through further development and collaboration with partners, we hope to bring smart contracts, side-chains, special higher-layer protocols, and interoperability techniques to the Abelian ecosystem in the near future. We would like to give the crypto market a new choice and a new platform, which empowers other innovators to build their next disruptive technologies, applications and businesses using quantum-resistant cryptographic techniques brought in by the Abelian community.

1.1 When ECC Becomes Forgeable

Elliptic-Curve Cryptography (ECC) is a class of public key cryptography based on the algebraic structure on elliptic curves over finite fields. It can be used to construct different kinds of cryptographic primitives such as key agreement, digital signature, encryption, zero-knowledge proof, and so on. All the major cryptocurrencies nowadays are based on ECC. In Bitcoin, for example, a wallet address is generated from an ECC public key after going through a series of cryptographic hash operations using SHA256 and RIPEMD-160, and also some error detection encoding.

An ECC public key is typically generated from a random number as the corresponding private key by calculating a scalar multiplication over an elliptic curve defined under a finite field. Simply speaking, the difficulty of compromising the ECC private key given the public key depends of the intractability of the underlying discrete logarithm problem in an additive group on an elliptic curve over a finite field, and this hard problem is usually referred to as the Elliptic Curve Discrete Logarithm Problem (ECDLP).

While ECC with carefully chosen parameters and curves is currently considered secure, it may no longer be true in the future when significant advancements have been made in the technology of quantum computing. In August 2015, the U.S. National Security Agency (NSA) released a major policy statement on the need for Post-Quantum Cryptography (PQC). The NSA had long cuddled up to ECC, swaying standards bodies away from RSA public key cryptosystem and toward ECC in the late 1990s, as well as recommending it as a strong enough solution for sensitive government agencies to use in guarding their secrets. The announcement made in August 2015 by NSA, however, showed a hard stop on recommending Suite-B, a 20-year-old public cryptographic standard that relies on ECC and was certified for top secret data protection, to sensitive government organizations. The agency suggested concerns over the advancement of quantum computing as the reason on stopping their support of Suite-B. This announcement

is a great stimulus to the development, standardization, and commercialization of new quantum-resistant / quantum-safe cryptographic algorithms, and PQC in general. It also symbolizes NSA's moving away from ECC.

Quantum computers exploit the properties of quantum superposition to generate Qubits, which can be used to effectively attack ECC, namely the underlying hard problem ECDLP. Unlike normal bits which can have a state of either 0 or 1, Qubits can exhibit both states 0 and 1 at the same time, and when combined with special algorithms such as Shor's Algorithm [79], can solve mathematical problems that would take traditional computers thousands of years to compute such as Integer Factorization or the Discrete Logarithm Problem in certain groups. While most number-theoretic cryptography, including ECC, relies on the conjectured hardness of these mathematical problems, quantum computers would render number-theoretic cryptosystems insecure in the near future where large-scale quantum computers may become available.

It is also worth noticing that large-scale quantum computers seem to be getting closer, for example, in November 2017, IBM announced a 50-qubit quantum computer, and in March 2018, Google announced a 72-qubit one. Following the reminiscent of Moore's Law as it is currently, quantum computers are expected to produce hundreds or thousands of Qubits in coming years, where ECC may be effectively compromised once millions of Qubits can be reached.

On December 20, 2016, the National Institute of Standards and Technology (NIST) made a Call for Proposal Announcement and initiated a process to solicit, evaluate, and standardize PQC [66]. According to NIST, the purpose of the process is to introduce *"new public-key cryptography standards which will specify one or more additional unclassified, publicly disclosed digital signature, public-key encryption, and key-establishment algorithms that are available worldwide, and are capable of protecting sensitive government information well into the foreseeable future, including after the advent of quantum computers."*

The goal of PQC is to develop cryptosystems that are secure against both

quantum and classical computers. Back to just a few years ago, the NIST already had received around 80 proposals of post-quantum cryptographic algorithms as candidates for the consideration of being a public standard. There are algorithms for digital signatures, public-key encryption, and key establishment. Activities on post-quantum cryptography research and development have been going actively and we are not lacking of elegantly designed, strong and efficient algorithms to start with for building a quantum-resistant cryptocurrency. As well articulated by NIST:

> "*While in the past it was less clear that large quantum computers are a physical possibility, many scientists now believe it to be merely a significant engineering challenge. Some engineers even predict that within the next twenty or so years sufficiently large quantum computers will be built to break essentially all public key schemes currently in use. Historically, it has taken almost two decades to deploy our modern public key cryptography infrastructure. Therefore, regardless of whether we can estimate the exact time of the arrival of the quantum computing era, we must begin now to prepare our information security systems to be able to resist quantum computing*".

We therefore believe that building the next-generation cryptocurrency, which is safe against quantum attacks, namely, a quantum-resistant cryptocurrency, is a strategic mission in our crypto community.

1.2 Call for Privacy

Privacy is another key merit that physical cash notes and coins of fiat currency inherently possess, and has been one of the most important concerns since cryptocurrency was first proposed. In 1983, David Chaum invented a cryptographic primitive called Blind Signature, which can be used for building an untraceable payment system that separated a person's identity from their transactions for anonymous

payments, and based on it, Chaum promoted an electronic cash system called eCash [24]. Then in the last 1980s, self-proclaimed libertarian anarchist group 'Cypherpunks' outlined some tools of modern cryptocurrency (pseudo-anonymous protection of identity, proof-of-work systems, private/public-key encryption and separation from government-backed currency [63]) in their memorandum called "The Crypto Anarchist Manifesto".

A decade later in 1997, Adam Beck introduced the first successful proof-of-work algorithm, such algorithms would become an important means of controlling the token supply of a given cryptocurrency. At the same time, Wei Dai, another member of the Cypherpunks and a researcher at Microsoft [16], released B-money, which highlighted the concepts of decentralization and digital contracts [26].

In 2004, Hal Finney, a computer scientist, and Cypherpunk [77] developed the first successful reusable proof-of-work (RPOW) protocol based on Beck's earlier work. RPOW allows users to transfer digital tokens by destroying and creating tokens during each transfer [34]. This process constituted the first proof-of-work digital cash system. Meanwhile, Nick Szabo, a computer scientist and cryptographer, launched a protocol that combined Wei Dai's concept of decentralization and Hal Finney's RPOW to create Bit Gold, the cryptocurrency that served as the predecessor to Bitcoin [81].

In 2009, the first popularized cryptocurrency – Bitcoin – was launched following the release of a paper titled, Bitcoin: A Peer-to-Peer Electronic Cash System [65], by someone writing under the pseudonym Satoshi Nakamoto. Cryptocurrencies have soared in popularity since 2009 with thousands of different crypto-coins available.

For most cryptocurrencies, however, the wallet addresses and the number of crypto-coins in each wallet are stored on the corresponding cryptocurrency's distributed ledger indefinitely. Exposure of the traces of crypto-coin flows is inevitable. The pseudonymity-based anonymity, which Bitcoin and most of the cryptocurrencies provision, is a weak form of anonymity due to the lack of un-

traceability and unlinkability. As all the transactions that take place between the network's participants are public, any transaction can be unambiguously traced to a unique origin and final recipient. Some results [73, 75, 69] have also shown that a careful or targeted analysis may reveal a connection between users and their transactions. In addition, there are few options for users to keep their identities anonymous or to hide the amount, other than, for example, through self-generating numerous new wallet addresses.

Full privacy means (1) keeping the coin addresses untraceable and unlinkable, and (2) hiding the transaction amount. Monero, Zcash, and Dash are some of the prominent ones. The existence of a cryptocurrency which is quantum-resistant and at the same time provisions full privacy is an open problem, and the cryptocurrency community has been putting great efforts onto solving this challenge, which is both theoretically and practically important.

1.3 Privacy vs. Accountability

From the privacy perspective, anonymity is a desirable feature of cryptocurrency. On the other hand, a strong degree of anonymity would also make cryptocurrency an ideal tool for conducting crimes or illegal transactions. Abusing anonymous cryptocurrencies for illegal purposes are on the rise in recent years. For example, in 2017, the worldwide ransomware attack WannaCry demanded ransom payments in Bitcoin, which were later converted into cryptocurrencies with a higher degree of anonymity, such as Monero.

The European Union's law-enforcement agency, Europol, have also raised alarms that cryptocurrencies with strong anonymity, such as Monero and Zcash, are gaining popularity over Bitcoin by criminals. There is an urgent need to develop technologies that can balance the anonymity and accountability so to prevent a cryptocurrency with full privacy from becoming a handy tool of the criminals.

Provisioning the accountability property in a cryptocurrency with full privacy is also essential for empowering such a cryptocurrency to support enterprise ap-

plications especially for financial services, and regulatory compliance.

1.4 Abelian and its Potential Features

Abelian (ABEL) is a brand new cryptocurrency and a completely new Blockchain system. It is secure against quantum attacks, and supports full privacy with optional accountability. Abelian's core cryptographic systems are provably secure under strong and practical adversarial models. Its implementations follow the design and will be open for public review. We target to build an open community which can continuously improve the security, scalability, and decentralization of Abelian, and also enable the growth of Abelian ecosystem so that everybody will be benefiting the security, privacy and accountability of this next-generation cryptocurrency and Blockchain technology.

1. **Quantum-resistance.** All cryptographic constructions used in Abelian's Blockchain platform are quantum resistant, and are provably secure in strong security models that capture the most practical attacking scenarios.

2. **Full Privacy (with optional accountability).** Abelian supports fine-grained privacy, where users can determine the privacy level for each of their transactions by selecting one from the following three when they make a transaction.

   (a) **Basic privacy:** coin addresses (namely the input and output coin addresses) and transaction amounts are public, while the output coin addresses are always one-time, freshly generated for each transaction. Note that this is the best privacy level that Bitcoin and most of the other conventional cryptocurrencies can achieve today.

   (b) **Full privacy:** no one can break the unlinkability or untraceability of coin addresses, and transaction amounts are hidden.

(c) **Full privacy with accountability:** to a designated authority, it has the same privacy level as 'Basic privacy'; to other participants, it is the same as 'Full privacy'.

Besides hiding the transaction amounts, to achieve full privacy, no one should be able to break the unlinkability or untraceability of coin addresses. We will see in the rest of this manuscript that our idea is to hide the actual input coin address (called 'consumed' coin address) in a group of coin addresses using linkable ring signature, and keep the output coin addresses public while the output coin addresses are always one-time, freshly generated for each transaction. In this way, linkability is not feasible as the destination addresses are always new and one-time, while the input address of each transaction is hidden among a group of other addresses.

With such a fine-grained privacy selection capability delineated above, Abelian can address both individuals' concerns on privacy and also, fulfill the regulatory or enterprise requirements on accountability. For individuals, while transactions with full privacy (with or without accountability) will consume more communication, computation, and storage resources of the system, more transaction fees may be required than those with basic privacy. Also, individuals may consider some external factors such as the fact that accountability is more acceptable to regulatory bodies, and depending on the actual scenarios, the individuals can flexibly choose the privacy level for their transactions from the basic privacy level to the full privacy level, or somewhere in between, namely, the full privacy with accountability level.

3. **Reliable with Proven Track Record PoW-based Consensus.** Abelian builds its consensus on Proof-of-Work (PoW) consensus protocol. For a new cryptocurrency focusing on advancing technologies on post-quantum readiness, the reliability and security is put at the first place. With its growth and continuous development along with Bitcoin, PoW has been proven to have

the merits of trustworthy sustainability, strong robustness against malicious participants, delicate incentive-compatibility, and openness to every participant. Having said that, Abelian may evolve itself from PoW into adopting some other more efficient and potentially environmentally friendly consensus protocols, for example, Proof-of-Stake (PoS). The future of Abelian will entirely be steered by the Abelian community, which will be open and fair.

2 Definition of Decentralized Anonymous Payment Scheme with Optional Accountability

We define a Decentralized Anonymous Payment Scheme with Optional Accountability (DAPOA) by generalizing and combining the ideas of Ring Confidential Transactions in Monero and Decentralized Anonymous Payment Scheme in [9], and adding the functionalities of optional accountability. The generalization enables us to identify the essence of the privacy, security, and accountability problems, so that we can investigate them in the setting of post-quantum cryptography.[1]

2.1 Notions

We study on a transaction-based ledger for a cryptocurrency[2], where transactions consume existing/old coins and generate new coins, i.e. each coin is an output of a transaction, and each coin can be consumed only one time by another transaction. Below, the term *'output coins'* of a transaction is used to denote the generated (new) coins by the transaction, and the term *'input coins'* of a transaction is used to denote the coins (i.e. the output coins of some previous transactions) referenced (and taken as input) by the transaction. A transaction consumes some or all of its input coins, and the term *'consumed coins'* of a transaction is used to denote the really consumed ones among its input coins.

For example, in Bitcoin, the consumed coins are the same as the input coins, while in Monero, the consumed coins are a subset of the input coins. While coins are consumed and generated by transactions, coins are owned by participants, who create and have one or multiple pseudonyms, and receive and spend coins using pseudonyms, trying to hide their identities in reality.

[1] Monero and Zerocash are described and constructed using concrete number-theoretic cryptographic constructions.

[2] Bitcoin and most existing cryptocurrencies use transaction-based ledger, while some others, e.g. Ethereum, use account-based ledger.

2.1.1 Coins, Transactions, Ledger, and Pseudonym

A ***transaction output***, referred to as TXO, is a data object txo, which is associated with the following:

- A *coin address*, denoted $txo.cnaddr$: a string determining who owns the coin implied by txo.
- A *coin value*, denoted $txo.cnvalue$: an integer, between 0 and a maximum value v_{max} (which is a system parameter, representing the total value of coins in the system, for example, 21 million for Bitcoin), that is the denomination of the coin implied by txo.
- A *coin value code*, denoted $txo.cnvaluecode$: a string that implies but hides $txo.cnvalue$.
- A *value hidden flag*, denoted $txo.valuehidden$: a boolean value in $\{\mathsf{True}, \mathsf{False}\}$. If $txo.valuehidden = \mathsf{True}$, txo is referred to as a *value-hidden* TXO, otherwise, txo is referred to as a *public* TXO.

A ***coin*** is a data object cn implied by a TXO txo, and is associated with the following:

- A *coin address*, denoted $cn.addr$: $cn.addr = txo.cnaddr$.
- A *coin secret key*, denoted $cn.sk$: a string known only to the coin owner, who can spend the coin with coin address $cn.addr$ using the coin secret key. We also refer to it as *coin spending key*.
- A *coin value*, denoted $cn.value$: the denomination of cn, as an integer between 0 and a maximum value v_{max}. If $txo.valuehidden = \mathsf{True}$, cn is referred to as a 'value-hidden coin', with $cn.value$ being determined by $txo.cnvaluecode$, otherwise, cn is referred to as a 'public coin', with $cn.value = txo.cnvalue$.

Note that each TXO implies one and only one coin, we may use 'TXO' and 'coin' as the same thing when it doesn't cause confusion.
A ***transaction*** is a data object tx, which is associated with the following:

- A set of *output* TXOs, denoted $tx.outtxo[]$: each element of $tx.outtxo[]$ is a TXO, implying an output coin.
- A set of *input* TXOs, denoted $tx.intxo[]$: a set of references to the output TXOs of previous transactions, each implying an input coin.
- A set of *consumed* TXOs, denoted $tx.csdtxo[]$: a subset of $tx.intxo[]$, representing the input TXOs that are really consumed by tx, each implying a consumed coin.
- A *transaction fee*, denoted $tx.fee$: an integer between 0 and v_{max}. For a transaction, the total value of the output coins implied by $tx.outtxo[]$, say v_{out}, should not exceed that of the consumed coins, say v_{in}, i.e. $0 \leq v_{out} \leq v_{in} \leq v_{max}$, and $tx.fee = v_{in} - v_{out}$.

Remark: For a transaction tx, while the output TXOs, input TXOs, and the transaction fee are public, the consumed TXOs may be hidden and only the transaction issuer knows which elements in $tx.intxo[]$ are consumed. In addition, for each output TXO, only the owner knows the implied coin, i.e. the (coin value, coin secret key) pair.

Ledger. At any given time T, all participants have access to L_T, the ledger at time T, which is a sequence of transactions. The ledger is append-only (i.e., $T < T'$ implies that L_T is a prefix of $L_{T'}$).[3]
A ***pseudonym*** is an object pdn representing an identity of a participant in the cryptocurrency, which is associated with:

[3]In reality, the ledger (such as the one of Bitcoin) is not perfect and may incur temporary inconsistencies.

- A *pseudonym address*, denoted $pdn.addr$: a unique string representing the pseudonym. A pseudonym owner uses his pseudonym address to receive coins via transactions.

- A *pseudonym secret key*, denoted $pdn.sk$: a string known only to the owner of the pseudonym. The owner can use this pseudonym secret key to spend the coins received via the corresponding pseudonym address.

Remark: In a cryptocurrency, a participant may have one or multiple pseudonyms, and through each pseudonym, he can own one or multiple coins, but each coin is owned by only one pseudonym. We define $pdn(cn)$ as the pseudonym through which the coin owner owns the coin.

Note that the above notations are general, and different cryptocurrencies may have different instances. For example,

- Bitcoin: For a pseudonym pdn, $(pdn.addr := H(\mathsf{pk}), pdn.sk := \mathsf{sk})$, where $(\mathsf{pk}, \mathsf{sk})$ is a (public key, secret key) pair for a signature scheme, and H is a cryptographic hash function. For a coin cn, the corresponding TXO has coin address $cn.addr := pdn.addr$ for some pseudonym pdn, the coin secret key is $cn.sk := pdn.sk$ for the same pdn. It is easy for anyone to find $pdn(cn)$, since the coin address of the corresponding TXO is public in the transaction and the coin address is just the pseudonym address.

- Monero: For a pseudonym pdn, the pseudonym address is $pdn.addr := (A, B)$, the pseudonym secret key is $pdn.sk := (a, b)$, where $A = aG, B = bG$. Note that (A, a) (resp. (B, b)) is a (public key, secret key) pair for a signature scheme. For a coin cn, suppose the coin is owned by a participant through a pseudonym pdn with address $pdn.addr = (A, B)$, the *coin address* is $cn.addr = H(rA)G + B$, the coin secret key is $cn.sk = H(aR) + b$, where r is chosen randomly by the participant who sends the coin to pdn and $R = rG$ is received from the sender. Except the coin owner who knows the value of

(a, b), no one can find $pdn(cn)$ from the coin information in the transaction, i.e. $cn.addr = H(rA)G + B$ and R.

2.1.2 Transactions for Protecting Privacy

As the transactions in the ledger are public and consequently are the source of privacy leakage, the following ways have been considered to protect the privacy of participants:

1. cutting the linkage between the transaction output coins and the pseudonyms, i.e. hiding the receiver/owner of a transaction output coin;
2. hiding the transaction consumed coins;
3. hiding the values of the transaction output coins.

Inspired by CryptoNote and Monero, DAPOA hides the coin receiver by using one-time coin address which is generated from the receiver's pseudonym address by the transaction sender, and hides the consumed coins by mixing them in a larger input coin set.

Four types of transaction are defined below to support full privacy with optional accountability: (1) *public transaction* consumes *public coins*, which have public coin address and public coin value, and generates public coins, and the consumed coins are just the input coins; (2) *mask transaction* consumes a *public coin* and generates a *value-hidden coin*, which has public coin address and hidden coin value, and the consumed coin is just the input coin; (3) *private transaction* consumes *value-hidden coins* and generates *value-hidden coins*, and the consumed coins are hidden in a larger input coin set; (4) *unmask transaction* consumes a *value-hidden coin* and generates a *public coin*. From the view of functionalities, as shown in Fig.1, when a coin is initially generated[4], it is a public coin. If the owner of a public

[4]The term 'coinbase transaction' is borrowed from Bitcoin, and is used to denote a transaction that takes no input coins but generates a new coin. Note that any cryptocurrency has a similar mechanism to create coins.

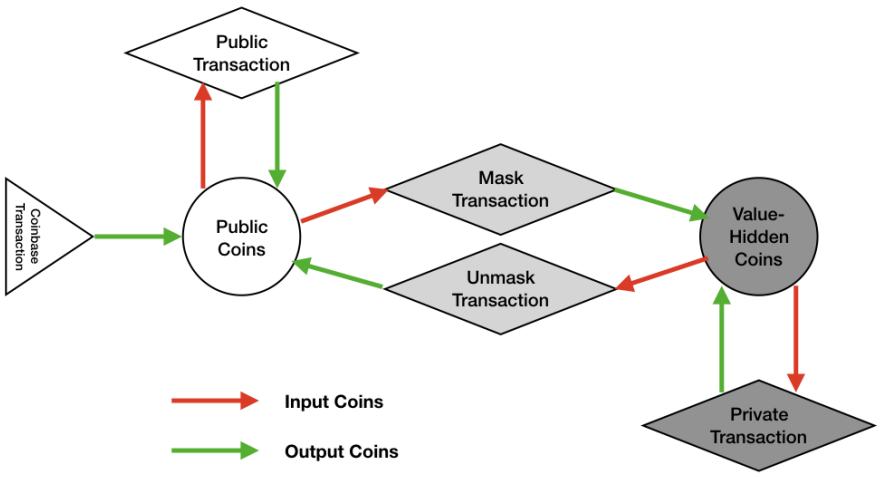

Figure 1: Four Types of Transactions

coin does not care the privacy, he may transact this coin using public transactions, otherwise, he may use a mask transaction to transfer this public coin to a value-hidden coin, and then transact the value-hidden coin using private transactions. When a user wishes to hold public coins, he can use the unmask transaction to transfer a value-hidden coin back to a public coin.

Coinbase Transaction. A *coinbase transaction* CbTx is a tuple $(outtxo; *)$, where *outtxo* is a public TXO, and * denotes other (implementation-dependent) information. The transaction CbTx records that a coin implied by TXO *outtxo* is generated.

Public Transaction. A *public transaction* PubTx is a tuple $(n, intxo[]; m, outtxo[]; spf; *)$, where n is the number of input TXOs, $intxo[]$ is the set of input TXOs, each of which references a public TXO output by some previous transaction, m is the number of output TXOs, $outtxo[]$ is the set of output TXOs, each of which is a public TXO, spf is a string proving that the transaction is valid, and * denotes other (implementation-dependent) information. The transaction PubTx records that n coins implied by the TXOs in $intxo[]$ are consumed and m new coins im-

plied by TXOs in $outtxo[]$ are generated. The transaction fee PubTx.fee can be computed from $intxo[]$ and $outtxo[]$ since the values of the coins implied by $intxo[]$ and $outtxo[]$ are public in $intxo[]$ and $outtxo[]$.

Mask Transaction. A *mask transaction* MsTx is a tuple $(intxo; outtxo, outvalue; spf; *)$, where $intxo$ is an input public TXO output by some previous transaction, $outtxo$ is a value-hidden TXO which implies a value-hidden coin with coin value $outvalue$ such that $0 \leq outvalue \leq intxo.cnvalue$, spf is a string proving that the transaction is valid, and * denotes other (implementation-dependent) information. The transaction MsTx records that a public coin implied by $intxo$ is consumed and a value-hidden coin implied by $outtxo$ has been generated. The value of the new value-hidden coin is $outvalue$ [5] , and the transaction fee is MsTx.$fee = intxo.cnvalue - outvalue$.

Private Transaction. A *private transaction* PriTx is a tuple $(l, n, intxo[]; m, outtxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$, where l is the number of input coins, n is the number of consumed coins, $intxo[]$ is the set of input TXOs, each of which references a value-hidden TXO output by some previous transaction, m is the number of output TXOs, $outtxo[]$ is the set of output TXOs, each of which is a value-hidden TXO, $value_{fee}$ is an integer in $[0, v_{max}]$ for transaction fee, spf and $csdcnsn[]$ form a proof that the transaction is valid, where $csdcnsn[]$ is the set of the serial numbers of the consumed coins, $trackable \in \{\mathsf{True}, \mathsf{False}\}$ implies whether the transaction is accountable, tif is a string that may be used by a designated authority to track the transaction (tif could be null if the transaction is not accountable), and * denotes other (implementation-dependent) information. The transaction PriTx records that n of the l input coins implied by $intxo[]$ are consumed and m new coins implied by $outtxo[]$ are generated. As the TXOs refer-

[5] Note that MsTx.outtxo implies a value-hidden coin, but the value of the coin can be learned from the transaction MsTx. The functionality of mask transaction is to generate value-hidden coins which can be transacted by private transactions.

enced by $intxo[]$ and the TXOs in $outtxo[]$ are value-hidden, so that the transaction fee cannot be computed directly from the input and output coins, transaction fee $value_{fee}$ is explicitly given in the transaction.

Unmask Transaction. An *unmask transaction* UmTx is a tuple $(intxo, invalue; outtxo; spf, csdcnsn; *)$, where $intxo$ is an input value-hidden TXO output by some previous transaction, $invalue$ is the coin value of the coin implied by $intxo$ such that $0 \leq invalue \leq v_{max}$, $outtxo$ is a public TXO which implies a public coin, spf and $csdcnsn$ form a proof that the transaction is valid where $csdcnsn$ is the serial number of the consumed coin, and * denotes other (implementation-dependent) information. The transaction UmTx records that a value-hidden coin implied by $intxo$ is consumed and a public coin implied by $outtxo$ has been generated. The value of the consumed coin is $invalue$ [6] and the transaction fee is $\mathsf{UmTx}.fee = invalue - outtxo.cnvalue$.

Remark: A public transaction consumes public coins and generates public coins, without hiding the coin values or the consumed coins, and provides only pseudonym level of anonymity. Mask transaction transfers a public coin to a value-hidden coin, just preparing value-hidden coins for privacy-protecting transactions – private transactions, and unmask transaction transfers a value-hidden coin back to a public coin. Note that as the coin value of the output (resp. input) value-hidden coin of mask (resp. unmask) transaction is explicitly included in the mask (resp. unmask) transaction, mask and unmask transactions provide only pseudonym anonymity. Private transaction consumes value-hidden coins and generates value-hidden coins, and hides the consumed coins in a larger input coin set, so that it provides full privacy.

[6] Note that the $intxo$ references a value-hidden TXO, and the owner of the consumed coin has to put the coin value of the consumed coin in the unmask transaction.

2.2 Definition of DAPOA

A DAPOA scheme Π is a tuple of polynomial-time algorithms (Setup, CreateTrackingKey, CreatePseudonym, DeriveCoinAddress, VerifyCoinAddress, DeriveCoinSK, HideCoinValue, OpenCoinValue, ComputeCoinSN, Coinbase, TransactPublic, Mask, Unmask, TransactPrivate, VerifyTransaction, Receive, TrackTransaction, VerifyTrack) with the following syntax and semantics.

System Setup. The algorithm Setup generates a list of public parameters.

Setup

- INPUTS: a security parameter λ.
- OUTPUTS: public parameters PP.

PP just contains some common public parameters, and the system does not require a trusted party to run the Setup algorithm. PP are published and made available to all parties (e.g., by embedding them into the protocol's implementation). The setup is done only once.

Create Tracking Key. The algorithm CreateTrackingKey generates a (public key, secret key) pair for accountability.

CreateTrackingKey

- INPUTS: public parameters PP.
- OUTPUTS: a tracking key pair $(\mathsf{PK}_\mathsf{T}, \mathsf{SK}_\mathsf{T})$.

A tracking authority runs the CreateTrackingKey algorithm and obtains the tracking key pair $(\mathsf{PK}_\mathsf{T}, \mathsf{SK}_\mathsf{T})$. The tracking public key PK_T is published, while the tracking secret key SK_T is kept secretly by the tracking authority.

Create Pseudonym. The algorithm CreatePseudonym generates a pseudonym pdn with (pseudonym address, pseudonym secret key).

CreatePseudonym

- INPUTS: public parameters PP.
- OUTPUTS: a pseudonym $(pdnaddr, pdnsk)$.

Each participant generates at least one (pseudonym address, pseudonym secret key) pair $(pdnaddr, pdnsk)$ in order to receive coins. The pseudonym address $pdnaddr$ is published, while the pseudonym secret key $pdnsk$ is used to redeem the coins received through $pdnaddr$.

Derive Coin Address. The algorithm DeriveCoinAddress derives a coin address $cnaddr$ from a pseudonym address.

DeriveCoinAddress

- INPUTS:
  - public parameters PP.
  - pseudonym address $pdnaddr$.
- OUTPUTS:
  - a coin address $cnaddr$.

When a participant wants to issue a transaction, he derives a coin address for each output TXO, from the pseudonym address of the TXO's receiver.

Verify Coin Address. The algorithm VerifyCoinAddress verifies whether a pseudonym is the receiver of a coin address.

VerifyCoinAddress

- INPUTS:

- public parameters PP.
- coin address $cnaddr$.
- pseudonym pdn with (pseudonym address, pseudonym secret key) pair $(pdnaddr, pdnsk)$.

- OUTPUTS:
  - a bit $b \in \{0, 1\}$, equals 1 iff $cnaddr$ is derived from $pdnaddr$.

When a participant receives a transaction, for each TXO of the transaction, he can verify whether the TXO's coin address is derived from his pseudonym address, i.e. whether he is the receiver of the implied coin.

Derive Coin Secret Key. The algorithm DeriveCoinSK derives a coin secret key corresponding to a coin address $cnaddr$.

DeriveCoinSK

- INPUTS:
  - public parameters PP.
  - coin address $cnaddr$.
  - pseudonym with (pseudonym address, pseudonym secret key) pair $(pdnaddr, pdnsk)$.
- OUTPUTS:
  - a coin secret key $cnsk$,
    if VerifyCoinAddress$(cnaddr, pdnaddr, pdnsk) = 1$; $\perp$ otherwise.

For an output TXO txo of a transaction, the corresponding receiver can compute the coin secret key $cnsk$ corresponding to $txo.cnadrr$, using his (pseudonym address, pseudonym secret key).

Hide Coin Value. The algorithm HideCoinValue generates a string *coinvaluecode* from an integer *coinvalue* which represents a coin value.

HideCoinValue

- INPUTS:
  - public parameters PP.
  - coin value *coinvalue*, which is an integer such that $0 \leq coinvalue \leq v_{max}$.
- OUTPUTS:
  - a string *coinvaluecode*.

The string *coinvaluecode* implies but hides a coin value.

Open Coin Value. The algorithm OpenCoinValue verifies whether a string *coinvaluecode* is for a coin value *coinvalue*.

OpenCoinValue

- INPUTS:
  - public parameters PP.
  - coin value *coinvalue*, which is an integer such that $0 \leq coinvalue \leq v_{max}$.
  - coin value code *coinvaluecode*.
- OUTPUTS:
  - a bit $b \in \{0, 1\}$, equals 1 iff *coinvaluecode* and *coinvalue* match.

Given a string *coinvaluecode* and a coin value *coinvalue*, OpenCoinValue can be used to check whether *coinvaluecode* does imply the coin value *coinvalue*.

Compute Coin Serial Number. The algorithm ComputeCoinSN computes a coin serial number for a coin.

ComputeCoinSN

- INPUTS:
  - public parameters PP.
  - coin address $cnaddr$.
  - coin secret key $cnsk$.
- OUTPUTS:
  - a string sn.

For a coin cn, its serial number $sn \leftarrow \mathsf{ComputeCoinSN}(\mathsf{PP}, cn.addr, cn.sk)$ is a unique identifier, i.e. for two coins cn_1 and cn_2, let $sn_i = \mathsf{ComputeCoinSN}(\mathsf{PP}, cn_i.addr, cn_i.sk)$ $(i = 1, 2)$, then $sn_1 = sn_2$ iff $cn_1.addr = cn_2.addr \ AND \ cn_1.sk = cn_2.sk$.

Create Coins. The algorithm Coinbase generates a public TXO $outtxo$.

Coinbase

- INPUTS:
  - public parameters PP.
  - output coin value $outvalue$: the value for the new coin, with $0 \leq outvalue \leq v_{max}$. [7]
  - receiver's pseudonym address $pdnaddr$.
- OUTPUTS:

[7] Note that the value for the new coin may be determined by the coin issue policy of the cryptocurrency, for example, the transaction fees of related transactions, the mining rewards. For simplicity, here we just use coinbase transaction to issue new coin values, without collecting the transaction fee.

– a public TXO $outtxo$: $outtxo.valuehidden = \mathsf{False}$, $outtxo.cnvalue = outvalue$, $outtxo.cnvaluecode$ is null, $outtxo.cnaddr \leftarrow \mathsf{DeriveCoinAddress}(\mathsf{PP}, pdnaddr)$.

– a coinbase transaction $\mathsf{CbTx} = (outtxo; *)$.

Note that Coinbase transaction does not directly use the receiver's pseudonym address as the output coin address. Actually, no transaction in DAPOA directly uses the receiver's pseudonym address as the output coin address.

Transacting Public Coins. The algorithm TransactPublic consumes public coins and transfers the value to new public coins. TransactPublic allows users to subdivide coins into smaller denominations, merge coins, and transfer ownership of public coins.

TransactPublic

- INPUTS:
  - public parameters PP.
  - the number of input TXOs n.
  - input TXOs $intxo[]$: for $i = 1$ to n, $intxo[i]$ references a public TXO output by some previous transaction.
  - consumed coins $csdcn[]$: for $i = 1$ to n, $csdcn[i]$ is the coin implied by $intxo[i]$, with $csdcn[i].sk$ and $csdcn[i].value$ known and $csdcn[i].addr = intxo[i].cnaddr$.
  - the number of output TXOs m.
  - (receiver's pseudonym address, value) pairs $out[]$: for $j = 1$ to m, $out[j]$ is a (pseudonym address, value) pair $(pdnaddr, value)$. The values satisfy the conditions that $0 \leq out[j].value \leq v_{max}$ and $\sum_{j=1}^{m} out[j].value \leq \sum_{i=1}^{n} intxo[i].cnvalue$.
- OUTPUTS:

- a set of new public TXOs $outtxo[]$: for $j = 1$ to m,
  $outtxo[j].valuehidden = \mathsf{False}$, $outtxo[j].cnvalue = out[j].value$,
  $outtxo[j].cnvaluecode$ is null,
  $outtxo[j].cnaddr \leftarrow \mathsf{DeriveCoinAddress}(\mathsf{PP}, out[j].pdnaddr)$.
- a public transaction $\mathsf{PubTx} = (n, intxo[]; m, outtxo[]; spf; *)$.

The TransactPublic algorithm consumes all the n input public TXOs (i.e. the implied coins), and for each (pseudonym address, value) pair it generates a public TXO (implying a coin). The transaction fee does not need to be included in the resulting PubTx, since both the input coins and the output coins are public coins so that the transaction fee can be computed from the coin values by anyone. Note that the order of the TXOs in $outtxo[]$ is determined by the participant who issues this transaction and can randomly arrange the order when it is necessary, i.e. an outsider cannot link a coin address to corresponding pseudonym address.

Masking Coins. The algorithm Mask consumes a public coin and generates a value-hidden coin.

Mask

- INPUTS:
  - public parameters PP.
  - input TXO $intxo$: $intxo$ references a public TXO output by some previous transaction.
  - consumed coin $csdcn$: $csdcn$ is the coin implied by $intxo$, with $csdcn.sk$ and $csdcn.value$ known and $csdcn.addr = intxo.cnaddr$.
  - output coin value $outvalue$: the value for the new coin, with $0 \leq outvalue \leq intxo.cnvalue$.
  - receiver's pseudonym address $pdnaddr$.
- OUTPUTS:

- a value-hidden TXO $outtxo$: $outtxo.valuehidden = \mathsf{True}$,
  $outtxo.value$ is null,
  $outtxo.cnvaluecode \leftarrow \mathsf{HideCoinValue}(\mathsf{PP}, outvalue)$,
  $outtxo.cnaddr \leftarrow \mathsf{DeriveCoinAddress}(\mathsf{PP}, pdnaddr)$.
- a mask transaction $\mathsf{MsTx} = (intxo; outtxo, outvalue; spf; *)$.

The participant (i.e. the owner of the input coin) knows the coin secret key and can issue the mask transaction. The transaction fee is $\mathsf{MsTx}.into.cnvalue - \mathsf{MsTx}.outvalue$.

Unmasking Coins. The algorithm Unmask consumes a value-hidden coin and generates a public coin.

Unmask

- INPUTS:
  - public parameters PP.
  - input TXO $intxo$: $intxo$ references a value-hidden TXO output by some previous transaction.
  - consumed coin $csdcn$: $csdcn$ is the coin implied by $intxo$, with $csdcn.sk$ and $csdcn.value$ known and $csdcn.addr = intxo.cnaddr$.
  - output coin value $outvalue$: the value for the new coin, with $0 \leq outvalue \leq csdcn.value$.
  - receiver's pseudonym address $pdnaddr$.
- OUTPUTS:
  - a public TXO $outtxo$: $outtxo.valuehidden = \mathsf{False}$,
    $outtxo.cnvalue = outvalue$, $outtxo.cnvaluecode$ is null,
    $outtxo.cnaddr \leftarrow \mathsf{DeriveCoinAddress}\ (\mathsf{PP}, pdnaddr)$.

– a unmask transaction $\mathsf{UmTx} = (intxo, incnvalue; outtxo; spf, csdcnsn; *)$, with $invalue = csdcn.value$ and $csdcnsn = \mathsf{ComputeCoinSN}(\mathsf{PP}, csdcn.addr, csdcn.sk)$.

The participant (i.e. the owner of the input coin) knows the coin secret key and coin value of the value-hidden coin implied by $intxo$, and can issue the unmask transaction. The transaction fee is $\mathsf{UmTx}.incnvalue - \mathsf{UmTx}.outtxo.cnvalue$.

Transacting Private Coins. The algorithm TransactPrivate consumes a subset of the input coins and transfers the value to new output coins. Both the input coins and the output coins are value-hidden coins. TransactPrivate allows users to subdivide coins into smaller denominations, merge coins, and transfer ownership of value-hidden coins.

TransactPrivate

- INPUTS:
  - public parameters PP.
  - number of the input TXOs l.
  - input TXOs $intxo[]$: for $i = 1$ to l, $intxo[i]$ references a value-hidden TXO output by some previous transaction.
  - number of the coins to be consumed n: $l = t \times n$ for some $t \geq 1$.
  - coins to be consumed $csdcn[]$: for $k = 1$ to n, $csdcn[k]$ is a coin implied by $intxo[i_k]$, with $csdcn[k].sk$ and $csdcn[k].value$ known and $csdcn[k].addr = intxo[i_k].cnaddr$.
  - the number of output TXOs m.
  - (receiver's pseudonym address, value) pairs $out[]$: for $j = 1$ to m, $out[j]$ is a (pseudonym address, value) tuple $(pdnaddr, value)$, where $0 \leq out[j].value \leq v_{max}$ and $\sum_{j=1}^{m} out[j].value \leq \sum_{k=1}^{n} csdcn[k].value$.

- a transaction fee $value_{fee}$ with $value_{fee} = \sum_{k=1}^{n} csdcn[k].value - \sum_{j=1}^{m} out[j].value$.
- (optional) a tracking public key $\mathsf{PK_T}$.

- OUTPUTS:
  - a set of TXOs $outtxo[]$: for $j = 1$ to m, $outtxo[j].valuehidden = \mathsf{True}$, $outtxo[j].value = null$, $outtxo[j].valuecode \leftarrow \mathsf{HideCoinValue}(\mathsf{PP}, out[j].value)$, and $outtxo[j].cnaddr \leftarrow \mathsf{DeriveCoinAddress}(\mathsf{PP}, out[j].pdnaddr)$.
  - a private transaction $\mathsf{PriTx} = (l, n, intxo[]; m, outxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$, where $csdcnsn[k] = \mathsf{ComputeCoinSN}(\mathsf{PP}, csdcn[k].addr, csdcn[k].sk)$, and $trackable = \mathsf{True}$ if the optional input tracking public key is not null, otherwise $tracklable = \mathsf{False}$.

The TransactPrivate algorithm takes in a set of coins, say l value-hidden coins, but only consumes a subset of the input coins, say n value-hidden coins. Thus the consumed coins are hidden in the set of input coins. The 'tracking public key' is an optional input, either $\mathsf{PK_T}$ or null, determined by the participant who issues the private transaction. If 'tracking public key' is null, the tif in PriTx is also null. The transaction fee needs to be explicitly given in the transaction, since the input and output TXOs are all value-hidden, so that the transaction fee cannot be computed from the input and output TXOs. Note that order of the elements in $outtxo[]$ is determined by the participant who issues this transaction and can randomly arrange the order when it is necessary.

Verifying Transactions. The algorithm VerifyTransaction checks the validity of a transaction.

VerifyTransaction

- INPUTS:

  - public parameters PP
  - a (coinbase, public, mask, private, or unmask) transaction Tx.
  - the current ledger L.
- OUTPUTS:
  - a bit $b \in \{0, 1\}$, equals 1 iff the transaction is valid.

The coinbase, public, mask, private, and unmask transactions must be verified before being considered well-formed. In practice, transactions can be verified by the nodes in the distributed system maintaining the ledger, as well as by participants who rely on these transactions.

Receiving Coins. The algorithm $\mathsf{Receive}$ scans the ledger and retrieves coins paid to a particular pseudonym address.

$\mathsf{Receive}$

- INPUTS:
  - public parameters PP.
  - recipient (pseudonym address, pseudonym secret key) pair ($pdnaddr$, $pdnsk$).
  - a (coinbase, public, mask, private, or unmask) transaction Tx on current ledger L.
  - an output TXO txo of Tx.
- OUTPUTS:
  - a coin cn, or $\perp$ implying that txo is not for the pseudonym address $pdnaddr$.

When a participant with (pseudonym address, pseudonym secret key) pair ($pdnaddr$, $pdnsk$) wishes to receive payments sent to him via $pdnaddr$, he uses the $\mathsf{Receive}$ algorithm to scan the ledger. [8]

[8] Actually, the participant only needs to scan the incremented transactions since last scanning.

Tracking Transactions. Given a valid private transaction (verified by the algorithm VerifyTransaction), the algorithm TrackTransaction obtains the consumed TXOs and the coin value for each output value-hidden TXO txo.

TrackTransaction

- INPUTS:
  - public parameters PP.
  - a private transaction $\mathsf{PriTx} = (l, n, intxo[]; m, outtxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$ on current ledger L, with $trackable =$ True.
  - a tracking secret key $\mathsf{SK_T}$.
- OUTPUTS:
  - set of consumed TXOs $csdtxo[]$: $csdtxo[]$ is a subset of $\mathsf{PriTx}.intxo[]$, having n elements.
  - set of public TXOs $pubtxo[]$: for $j = 1$ to m,
    $pubtxo[j].valuehidden =$ False,
    $pubtxo[j].cnaddr = \mathsf{PriTx}.outtxo[j].cnaddr$,
    $pubtxo[j].cnvaluecode = \mathsf{PriTx}.outtxo[j].cnvaluecode$,
    $pubtxo[j].cnvalue$ satisfies
    $\mathsf{OpenCoinValue}(\mathsf{PP}, \mathsf{PriTx}.outtxo[j].cnvaluecode, pubtxo[j].cnvalue) = 1$.

When the authority wishes to track a private transaction which is set to be trackable by the transaction issuer, he uses the TrackTransaction algorithm with his tracking secret key as input, to identify the consumed TXOs of the transaction and reveal the coin values of the output value-hidden TXOs of the transaction. Note that if he also wants to know the coin values of the consumed TXOs, he just needs to track the previous transactions which generated these consumed TXOs.

Verifying Track. The algorithm VerifyTrack checks the validity of the tracking result of VerifyTransaction algorithm on a private transaction.

VerifyTrack

- INPUTS:
  - public parameters PP.
  - tracking public key PK_T.
  - a private transaction PriTx on current ledger L.
  - set of consumed TXOs $csdtxo[]$.
  - set of public TXOs $pubtxo[]$.
- OUTPUTS:
  - a bit $b \in \{0, 1\}$, equals 1 iff the tracking result $(csdtxo[], pubtxo[])$ is valid.

For a valid private transaction PriTx, only if the tracking result $(csdtxo[], pubtxo[])$ is verified valid, the tracking result is accepted.

Remark: Above we only consider the simplest tracking mechanism, i.e. one authority, one tracking key. We can extend and improve this part, for example, multiple authorities cooperate to generate the tracking key, and transaction issuer specifies the tracking public key in a flexible manner, so that an authorized auditor cannot track all transactions in the ledger, instead, he can track only the transactions with particular tracking tag.

Note that a public transaction consumes all input coins and generates public coins, a mask (resp. unmask) transaction consumes a public (resp. value-hidden) coin and generates a value-hidden (resp. public) coin, but the coin value of the value-hidden coin in mask (resp. unmask) transaction can be obtained from the transaction. In other words, public, mask, and unmask transactions do not provide privacy except the pseudonym anonymity. Private transactions provide full privacy by hiding the consumed coins in a larger input coin set and generating value-hidden coins of which the values are not available from the transaction. Even when a private

transaction is tracked by an authority, it still remains pseudonym anonymity to the authority, like the public transaction.

It is worth mentioning that the pseudonym level of anonymity of Abelian is a still slightly stronger than that of Bitcoin. In Bitcoin, a coin address is just a pseudonym address of the receiver, and it is provided by the receiver to the transaction issuer, so that a receiver may use one pseudonym (address) to receive multiple coins and as a result multiple coins are linked to the same pseudonym. As well known, to get stronger pseudonym anonymity, it is suggested to use one-time coin addresses, i.e. one address for one coin. In Bitcoin, whether the pseudonym anonymity of one-time coin address is achieved depends on whether the participant generates a new pseudonym for each coin to be received. In Abelian, one-time coin address is compulsory, i.e. for each coin sent to a pseudonym address, the sender derives a new one-time coin address from the pseudonym. A malicious sender may attempt to use the same coin address multiple times for a receiver, but the receiver can detect such a malicious behavior easily and reject the transactions with repeated output addresses. Also as defined later, the derived coin address cannot be linked to the corresponding pseudonym. Note that as the coin address is generated on-the-fly by the transaction issuer, the coin address does not leak its owner's pseudonym.

2.3 Correctness of DAPOA

The correctness of DAPOA includes the verification of the derived coin address, the verification of coinbase/public/mask/private/unmask transactions, the receiving of the output coins of coinbase/public/mask/private/unmask transaction, the tracking of private transactions, and the verification of the tracking results.

For any $\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda)$, $(\mathsf{PK_T}, \mathsf{SK_T}) \leftarrow \mathsf{CreateTrackingKey}(\mathsf{PP})$, current ledger L,

- for any $value$ such that $0 \leq value \leq v_{max}$, and any pseudonym address

$pdnaddr$ such that $(pdnaddr, pdnsk) \leftarrow \mathsf{CreatePseudonym}(\mathsf{PP})$, let

$(outtxo, \mathsf{CbTx}) \leftarrow \mathsf{Coinbase}(\mathsf{PP}, value, pdnaddr)$,

$b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{CbTx}, L)$,

$b' \leftarrow \mathsf{VerifyCoinAddress}(\mathsf{PP}, \mathsf{CbTx}.outtxo.cnaddr, pdnaddr, pdnsk)$,

$cn \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnaddr, pdnsk, \mathsf{CbTx}, \mathsf{CbTx}.outtxo)$,

where $\mathsf{CbTx} = (outtxo; *)$, it holds that

- $b = 1$, implying CbTx is a valid transaction.
- $b' = 1$, implying the coin address $\mathsf{CbTx}.outtxo.cnaddr$ is valid for pseudonym address $pdnaddr$.
- cn is a coin with $cn.addr = \mathsf{CbTx}.outtxo.cnaddr$, $cn.value = value$, and $cn.sk = \mathsf{DeriveCoinSK}(\mathsf{PP}, \mathsf{CbTx}.outtxo.cnaddr, pdnaddr, pdnsk)$.

- for any unspent public TXO set $intxo[]$ with size n and the corresponding coins $csdcn[]$, any (pseudonym address, value) pairs $out[] = \{(pdnaddr_j, value_j)\}_{j=1}^{m}$ with size m such that $\{(pdnaddr_j, pdnsk_j) \leftarrow \mathsf{CreatePseudonym}(\mathsf{PP})\}_{j=1}^{m}$ and $0 \leq out[j].value \leq v_{max}$, $\sum_{j=1}^{m} out[j].value \leq \sum_{i=1}^{n} intxo[i].cnvalue$, let

  $(outtxo[], \mathsf{PubTx}) \leftarrow \mathsf{TransactPublic}(\mathsf{PP}, n, intxo[], csdcn[], m, out[])$,

  $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{PubTx}, L)$,

  $\{b'_{k,j} \leftarrow \mathsf{VerifyCoinAddress}(\mathsf{PP}, \mathsf{PubTx}.outtxo[j].cnaddr, pdnaddr_k, pdnsk_k)$,

  $cn_{k,j} \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnaddr_k, pdnsk_k, \mathsf{PubTx}, \mathsf{PubTx}.outtxo[j])\}_{k,j=1}^{m}$,

  where $\mathsf{PubTx} = (n, intxo[]; m, outtxo[]; spf; *)$, it holds that

  - $b = 1$, implying PubTx is a valid transaction.
  - if $k = j$, $b'_{k,j} = 1$ and $cn_{k,j}$ is a coin with $cn_{k,j}.addr = \mathsf{PubTx}.outtxo[j].cnaddr$, $cn_{k,j}.value = out[j].value$, $cn_{k,j}.sk = \mathsf{DeriveCoinSK}\ (\mathsf{PP}, \mathsf{PubTx}.outtxo[j].\ cnaddr, pdnaddr_k, pdnsk_k)$;

otherwise $b'_{k,j} = 0$ and $cn_{k,j} = \perp$, implying that $pdnaddr_k$ is not the intended receiver of $\mathsf{PubTx}.outtxo[j]$.

- for any unspent public TXO $intxo$ and the corresponding coin $csdcn$, any $outvalue$ such that $0 \leq outvalue \leq v_{max}$, $outvalue \leq intxo.cnvalue$, and any pseudonym address $pdnaddr$ such that $(pdnaddr, pdnsk) \leftarrow \mathsf{CreatePseudonym}(\mathsf{PP})$, let

  $(outtxo, \mathsf{MsTx}) \leftarrow \mathsf{Mask}(\mathsf{PP}, intxo, csdcn, outvalue, pdnaddr)$,

  $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{MsTx}, L)$,

  $b' \leftarrow \mathsf{VerifyCoinAddress}(\mathsf{PP}, \mathsf{MsTx}.outtxo.cnaddr, pdnaddr, pdnsk)$,

  $cn \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnaddr, pdnsk, \mathsf{MsTx}, \mathsf{MsTx}.outtxo)$,

  where $\mathsf{MsTx} = (intxo; outtxo, outvalue; spf; *)$, it holds that

  - $b = 1$, implying MsTx is a valid transaction.
  - $b' = 1$, implying the coin address $\mathsf{MsTx}.outtxo.cnaddr$ is valid for pseudonym address $pdnaddr$.
  - cn is a coin with $cn.addr = \mathsf{MsTx}.outtxo.cnaddr$, $cn.value = outvalue$, and $cn.sk = \mathsf{DeriveCoinSK}(\mathsf{PP}, \mathsf{MsTx}.outtxo.cnaddr, pdnaddr, pdnsk)$.

- for any unspent value-hidden TXO $intxo$ and the corresponding coin $csdcn$, any $outvalue$ such that $0 \leq outvalue \leq csdcn.value$, and any pseudonym address $pdnaddr$ such that $(pdnaddr, pdnsk) \leftarrow \mathsf{CreatePseudonym}(\mathsf{PP})$, let

  $(outtxo, \mathsf{UmTx}) \leftarrow \mathsf{Unmask}(\mathsf{PP}, intxo, csdcn, outvalue, pdnaddr)$,

  $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{UmTx}, L)$,

  $b' \leftarrow \mathsf{VerifyCoinAddress}(\mathsf{PP}, \mathsf{UmTx}.outtxo.cnaddr, pdnaddr, pdnsk)$,

  $cn \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnaddr, pdnsk, \mathsf{UmTx}, \mathsf{UmTx}.outtxo)$,

  where $\mathsf{UmTx} = (intxo, incnvalue; outtxo; spf, csdcnsn; *)$ with $incnvalue = csdcn.value$, it holds that

- $b = 1$, implying UmTx is a valid transaction.
- $b' = 1$, implying the coin address $\mathsf{UmTx}.outtxo.cnaddr$ is valid for pseudonym address $pdnaddr$.
- cn is a coin with $cn.addr = \mathsf{UmTx}.outtxo.cnaddr$, $cn.value = outvalue$, and $cn.sk = \mathsf{DeriveCoinSK}(\mathsf{PP}, \mathsf{UmTx}.outtxo.cnaddr, pdnaddr, pdnsk)$.

- for any value-hidden TXO set $intxo[]$ with size l, let $csdcn[]$ is a set of unspent coins with size n such that for $k \in \{1, \ldots, n\}$ $csdcn[k]$ is the coin implied by TXO $intxo[i_k]$ for some $i_k \in \{1, \ldots, l\}$, and let $csdidx = \{i_1, \ldots, i_n\}$. For any (pseudonym address, value) pairs $out[] = \{(pdnaddr_j, value_j)\}_{j=1}^{m}$ with size m such that $\{(pdnaddr_j, pdnsk_j) \leftarrow \mathsf{CreatePseudonym}(\mathsf{PP})\}_{j=1}^{m}$ and $0 \leq out[j].value \leq v_{max}$, $\sum_{j=1}^{m} out[j].value \leq \sum_{k=1}^{n} csdcn[k].value$, let transaction fee be $value_{fee} = \sum_{k=1}^{n} csdcn[k].value - \sum_{j=1}^{m} out[j].value$,

  $(outtxo[], \mathsf{PriTx}) \leftarrow \mathsf{TransactPrivate}(\mathsf{PP}, l, n, intxo[], csdcn[], m, out[], value_{fee}, [\mathsf{PK_T}])$,
  $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{PriTx}, L)$,
  $\{b'_{k,j} \leftarrow \mathsf{VerifyCoinAddress}(\mathsf{PP}, \mathsf{PriTx}.outtxo[j].cnaddr, pdnaddr_k, pdnsk_k)$,
  $cn_{k,j} \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnaddr_k, pdnsk_k, \mathsf{PriTx}, \mathsf{PriTx}.outtxo[j])\}_{k,j=1}^{m}$,

  where $\mathsf{PriTx} = (l, n, intxo[]; m, outtxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$, it holds that
  - $b = 1$, implying PriTx is a valid transaction.
  - if $k = j$, $b'_{k,j} = 1$ and $cn_{k,j}$ is a coin with $cn_{k,j}.addr = \mathsf{PriTx}.outtxo[j].cnaddr$, $cn_{k,j}.value = out[j].value$, $cn_{k,j}.sk \leftarrow \mathsf{DeriveCoinSK}(\mathsf{PP}, \mathsf{PriTx}.outtxo[j].cnaddr, pdnaddr_k, pdnsk_k)$; otherwise $b'_{k,j} = 0$ and $cn_{k,j} = \perp$, implying that $pdnaddr_k$ is not the intended receiver of $\mathsf{PriTx}.outtxo[j]$.
  - if $trackable = \mathsf{True}$, let $(csdtxo[], pubtxo[]) \leftarrow \mathsf{TrackTransaction}(\mathsf{PP}, \mathsf{PriTx}, L, \mathsf{SK_T})$,

* $csdtxo[]$ is a subset of $intxo[]$ with size n, let $trcsdidx = \{j_1, \ldots, j_n\}$ be the index set of $csdtxo[]$'s elements in $intxo[]$, it holds that $trcsdidx = csdidx$.
* pubtxo[] is a set of public TXOs with size m, for $j = 1$ to m, $pubtxo[j].valuehidden = \mathsf{False}$,
  $pubtxo[j].cnvalue$ satisfies
  $\mathsf{OpenCoinValue}(\mathsf{PP}, \mathsf{PriTx}.outtxo[j].cnvaluecode, pubtxo[j].cnvalue) = 1$,
  $pubtxo[j].cnvaluecode$ is null,
  and $pubtxo[j].cnaddr = \mathsf{PriTx}.outtxo[j].cnaddr$.
* let $b_T \leftarrow \mathsf{VerifyTrack}(\mathsf{PP}, \mathsf{PK}_T, \mathsf{PriTx}, L, csdtxo[], pubtxo[])$, it holds that $b_T = 1$.

2.4 Security of DAPOA

A DAPOA scheme is secure, if it is ensured that

- Only the participant who knows the coin secret key can issue a transaction to consume the coin.
- Each coin can be spent only once.
- The total value of the new generated coins and the transaction fee is equal to the total value of the consumed coins.
- The value of each new generated coin and the transaction fee is in a certain range $[0, v_{max}]$.
- For a transaction output TXO, only using the secret key for the target pseudonym can compute the secret key for the coin implied by the TXO.

The core of these security requirements is the VerifyTransaction() algorithm, once an attacker can issue transactions that pass the verification and break the above

requirements, the scheme is not secure. Below, we formally define the security of DAPOA.

Definition 1 *Given a DAPOA scheme* (Setup, CreateTrackingKey, CreatePseudonym, DeriveCoinAddress, VerifyCoinAddress, DeriveCoinSK, HideCoinValue, OpenCoinValue, ComputeCoinSN, Coinbase, TransactPublic, Mask, Unmask, TransactPrivate, VerifyTransaction, Receive, TrackTransaction, VerifyTrack)*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. Setup.

   (a) PP *are generated using* $\mathsf{Setup}(1^\lambda, \omega)$ *where* ω *is the randomness used to run* Setup*,* PP *and* ω *are given to* $\mathcal{A}$*.*

   (b) $(\mathsf{PK_T}, \mathsf{SK_T})$ *is generated using* $\mathsf{CreateTrackingKey}(\mathsf{PP})$*, and* $(\mathsf{PK_T}, \mathsf{SK_T})$ *is given to* $\mathcal{A}$*.* [9]

   (c) *An empty ledger* $L = \emptyset$ *is initialized and given to* $\mathcal{A}$*.*

   (d) *Below some internal states are initialized. Note that the ledger* L *is a global status, and* $\mathcal{A}$ *would make queries adaptively according to this global status, and the response given to* $\mathcal{A}$ *is also based on this global status. The following internal status could be obtained from the global status, and we define them explicitly just to describe the game clearly and easily.*

   - *A total value* $v_{total} = 0$ *is initialized.*
   - *An empty pseudonym list* $PdnL = \emptyset$ *is initialzied, where each element in* $PdnL$ *is a (pseudonym address, pseudonym secret key, is corrupted) tuple* $(pdnaddr, pdnsk, iscorrupted)$*.*
   - *An empty pseudonym list* $PdnaddrL_{\mathcal{A}}$ *is initialzied, which is used to store the pseudonym address created by* $\mathcal{A}$*.*

[9] The security requires that the tracking authority cannot forgeable a transaction or double spend.

- *An empty* TXO *list* $TxoL = \emptyset$ *is initialized, where each element in* $TxoL$ *is a (*TXO*, is consumed, coin, is corrupted, owner pseudonym address) tuple* $(txo, isconsumed, coin, iscorrupted, pdnaddr)$.
- *A flag* $b_{\mathcal{A}win} = 0$ *initialized.*

2. **Phase 1.** $\mathcal{A}$ *is given the oracles:*

- $OCreatePseudonym()$*:* This captures that an adversary can observe other particpiants' pseudonym addresses.

  $OCreatePseudonym()$ *runs* $(pdnaddr, pdnsk) \leftarrow$ CreatePseudonum(PP), *adds* $(pdnaddr, pdnsk,$ False$)$ *into* $PdnL$*, and returns* $pdnaddr$ *to* $\mathcal{A}$.
- $OAddPseudonymAddr(\cdot)$*:* This captures that an adversary can create a pseudonym and publishes the pseudonym address.

  $OAddPseudonymAddr(pdnaddr)$ *first checks if* $PdnL$ *contains a tuple* $pdnL$ *such that* $pdnL.pdnaddr = pdnaddr$*. If such a* $pdnL$ *exists, the oracle returns* $\perp$ *to* $\mathcal{A}$*, implying this is a not valid query. Otherwise, if* $PdnaddrL_{\mathcal{A}}$ *does not contain* $pdnaddr$*, the oracle adds* $pdnaddr$ *to* $PdnaddrL_{\mathcal{A}}$.
- $OCoinbase()$*:* This captures that an adversary can observe the coinbase transactions created by other participants.

  $OCoinbase()$ *first checks whether* $pdnL$ *is empty or* $v_{total} = v_{max}$*. If* $PdnL$ *is empty or* $v_{total} = v_{max}$*, returns* $\perp$ *to* $\mathcal{A}$*, implying this is not a valid query, since no pseudonym address could be used to receive the coins or the total value of the existing coins have meet the upper bound of the cryptocurrency. The oracle chooses random value such that* $0 < value \leq v_{max} - v_{total}$ *and random tuple* $pdnL \in PdnL$*, runs* $(outtxo,$ CbTx$) \leftarrow$ Coinbase$(PP, value,\ pdnL.pdnaddr)$*, where* CbTx $= (outtxo; *)$*. Then the oracle runs* $cn \leftarrow$ Receive$(PP, pdnL.pdnaddr, pdnL.pdnsk,$ CbTx, CbTx.$outtxo)$ *and adds tuple* (CbTx.outtxo, False, $cn, pdnL.iscorrupted, pdnaddr)$ *into* $TxoL$*. Note that here* $pdnL.iscorrupted$ *is*

used to initialize whether the coin secret key has been corrupted by the adversary, as if $\mathcal{A}$ has corrupted the corresponding pseudonym secret key, he also knows all the coin secret keys derived from this pseudonym. At last the oracle adds CbTx *into L and sets $v_{total} = v_{total} + value$, and gives L to $\mathcal{A}$.*

- *$OAddCbTx(\cdot)$:* This oracle captures that the adversary can produce a coinbase transaction $\mathsf{CbTx} = (outtxo; *)$ and ask this oracle to add CbTx to the ledger.

  $OAddCbTx(\mathsf{CbTx})$ runs $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{CbTx}, L)$. If $b = 0$ or $\mathsf{CbTx}.outtxo.cnvalue + v_{total} > v_{max}$, the oracle returns $\perp$ to $\mathcal{A}$, otherwise, the oracle calls the subroutine Subroutine-ProceedOutTXO$(\cdot, \cdot)$ *in Alg. 1 with $tx = \mathsf{CbTx}$ and $txo = \mathsf{CbTx}.outtxo$ as input.*

Algorithm 1 Subroutine-ProceedOutTXO(tx, txo)

Require: tx is a transaction, and txo is a TXO of tx.

1: Find a tuple $pdnL$ in $PdnL$ such that
   $cn \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnL.pdnaddr, pdnL.pdnsk, tx, txo)$ and $cn \neq \perp$.
2: **if** Such a $pdnL$ exists in $PdnL$ **then**
3: Add $(txo, \mathsf{False}, cn, pdnL.iscorrupted, pdnL.pdnaddr)$ into $TxoL$.
4: **else**
5: Form a new coin cn by setting $cn.addr = txo.cnaddr, cn.sk = null, cn.value = txo.cnvalue$, and add $(txo, \mathsf{False}, cn, \mathsf{True}, null)$ into $TxoL$. *Note that only the adversary $\mathcal{A}$ knows the coin secret key for this coin and only the adversary knows the pseudonym address for this txo, and if $txo.valuehidden = \mathsf{True}, cn.value = txo.cnvalue = null$, i.e. only the adversary knows the coin value.*
6: **end if**

The oracle adds CbTx *into L and sets $v_{total} = v_{total} + value$, and returns L to $\mathcal{A}$.*

- *$OTransactPublic()$:* This oracle captures that an adversary can observe the public transactions created by other participants, and also captures that an adversary acts as receivers of the transaction output coins.

  The oracle first interacts with the adversary to form the inputs for the transaction:

  (a) The oracle choose n random tuples from $TxoL$, say $\{txoL_1, \ldots, txoL_n\}$, such that $n \geq 1$ and $txoL_i.isconsumed = \mathsf{False}$ AND $txoL_i.coin.sk \neq null$ AND $txoL_i.txo.valuehidden = \mathsf{False}$ $(1 \leq i \leq n)$. Set $intxo[i] = txoL_i.txo, csdcn[i] = txoL_i.coin$ $(1 \leq i \leq n)$. Let $value_{in} = \sum_{i=1}^{n} csdcn[i].value$, the oracle chooses random $value_{out}$ such that $0 < value_{out} < value_{in}$, and gives $value_{out}$ to $\mathcal{A}$. [10]

  (b) The adversary specifies a set of (pseudonym address, value), say $out[]$, with size $m_0 > 1$, such that for $j = 1$ to m_0, $out[j].pdnaddr = pdnL.pdnaddr$ for some $pdnL \in PdnL$, or $out[j].pdnaddr = pdnaddrL_{\mathcal{A}}$ for some $pdnaddrL_{\mathcal{A}} \in PdnaddrL_{\mathcal{A}}$, $0 < out[j].value \leq value_{out}$, and $out[j].pdnaddr \neq out[j'].pdnaddr$ for $1 \leq j \neq j' \leq n$, and $\sum_{j=1}^{m_0} out[j].value = value_{out}$.

  The oracle chooses a random tuple $pdnL_m \in PdnL$ and a random value $value_m$ such that $0 < value_m < value_{in} - value_{out}$, adds $(pdnL_m.pdnaddr, value_m)$ to $out[]$, and sets $m = m_0 + 1$. This is used to receive the change.

  The oracle randomly arranges the order of the elements in $out[]$.

  The oracle then runs $(outtxo[], \mathsf{PubTx}) \leftarrow \mathsf{TransactPub}(\mathsf{PP}, n, intxo[],$

[10] From the view of practice, an attacker can observe the transactions, but cannot specify the inputs of a transaction, unless he issues a transaction which case is captured by the later oracle $OAddPubTx(\cdot)$. As an attacker may drive a transaction to transfer values to some particular pseudonym address, below we allow the adversary to specify the output pseudonym addresses, as well as the value for each pseudonym addresses. But here we do not allow the adversary to specify the total output value, since the total output value are determined by two parties, the sender and the receiver.

$csdcn[],\ m, out[])$, *where* PubTx $= (n, intxo[]; m, outtxo[]; spf; *)$, *and performs*

(a) *for* $i = 1$ *to* n,
updates $txoL_i$ *(in* $TxoL$*) by setting* $txoL_i.isconsumed =$ True.

(b) *for* $j = 1$ *to* m, *the oracle calls the* Subroutine-ProceedOutTXO$(\cdot, \cdot)$ *in Alg. 1 with* $tx =$ PubTx *and* $txo =$ PubTx.$outtxo[j]$ *as input. Note that, for the* TXO*s in* PubTx.$outtxo[]$ *which are received by the pseudonym addresses in* $PdnaddrL_{\mathcal{A}}$*, although the oracle knows the coin value and the pseudonym address, the* Subroutine-ProceedOutTXO *sets the two values to null. This does not weaken the adversary's capacity or our security model.*

At last, the oracle adds PubTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OAddPubTx(\cdot)$*:* This oracle captures that the adversary can produce a public transaction PubTx and ask this oracle to add PubTx to the ledger.
$OAddPubTx($PubTx$)$ *first runs* $b \leftarrow$ VerifyTransaction(PP, PubTx, L), *where* PubTx $= (n, intxo[];\ m, outtxo[]; spf; *)$. *If* $b = 0$, *the oracle returns* $\perp$ *to* $\mathcal{A}$, *otherwise*

(a) *For* $i = 1$ *to* n*: find a tuple* $txoL_i \in$ TxoL *such that* $txoL_i.txo =$ PubTx.$intxo[i]$, *if such a* $txoL_i$ *does not exist, set* $b_{\mathcal{A}win} = 1$, *otherwise*

- *if* $txoL_i.coin.sk \neq null$
  - *if* $txoL_i.txo.valuehidden =$ False AND $txoL_i.iscorrupted =$ True AND $txoL_i.isconsumed =$ False, *set* $txoL_i.isconsumed =$ True.
  - *otherwise, set* $b_{\mathcal{A}win} = 1$.
- *otherwise,*
  - *if* $txoL_i.txo.valuehidden =$ False AND $txoL_i.isconsumed =$ False, *set* $txoL_i.isconsumed =$ True.

$*$ *otherwise, set* $b_{\mathcal{A}win} = 1$.

(b) for $j = 1$ *to* m*, the oracle calls the* Subroutine-ProceedOutTXO$(\cdot, \cdot)$ *in Alg. 1 with* $tx =$ PubTx *and* $txo =$ PubTx.$outtxo[j]$ *as input.*

At last, the oracle adds PubTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OMask()$*:* This oracle captures that the adversary can observe the mask transactions created by other participants.

  The oracle first chooses the input parameters randomly:

  (a) Choose a random tuple from $TxoL$*, say* $intxoL$*, such that* $intxoL.txo.valuehidden =$ False AND $intxoL.isconsumed =$ False AND $intxoL.coin.sk \neq null$*. Set* $intxo = txoL.txo, csdcn = txoL.coin$.

  (b) Choose a random tuple from $PdnL$*, say* $pdnL$*. Choose a random value* $outvalue$ *such that* $0 < outvalue \leq csdcn.value$.

  The oracle then runs $(outtxo[], \mathsf{MsTx}) \leftarrow \mathsf{Mask}(\mathsf{PP}, intxo, csdcn, outvalue, pdnL.pdnaddr)$*, where* $\mathsf{MsTx} = (intxo; outtxo, outvalue; spf; *)$*, and*

  (a) update $txoL$ *(in* $TxoL$*) by setting* $txoL.isconsumed =$ True.

  (b) run $cn \leftarrow \mathsf{Receive}(\mathsf{PP}, pdnL.pdnaddr, pdnL.pdnsk, \mathsf{MsTx}, \mathsf{MsTx}.outtxo)$*, and adds* $(\mathsf{MsTx}.outtxo, \mathsf{False}, cn, pdnL.iscorrupted, pdnL.pdnaddr)$ *into* $TxoL$.

  At last, the oracle adds MsTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OAddMsTx(\cdot)$*:* This oracle captures that the adversary can produce a mask transaction MsTx and ask this oracle to add MsTx to the ledger.

  $OAddMsTx(\mathsf{MsTx})$ *first runs* $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{MsTx}, L)$*, where* $\mathsf{MsTx} = (intxo; outtxo, outvalue; spf; *)$*. If* $b = 0$*, the oracle returns* $\perp$ *to* $\mathcal{A}$*, otherwise*

  (a) the oracle finds a tuple $txoL \in$ TxoL *such that* $txoL.txo =$ MsTx.$intxo$*, if such a* $txoL$ *does not exist, set* $b_{\mathcal{A}win} = 1$*, otherwise*

  – *if* $txoL.coin.sk \neq null$

* *if* $txoL.txo.valuehidden =$ False AND $txoL.iscorrupted =$ True AND $txoL.isconsumed =$ False, *set* $txoL.isconsumed =$ True.
* *otherwise, set* $b_{\mathcal{A}win} = 1$.

– *otherwise,*

* *if* $txoL_i.txo.valuehidden =$ False AND $txoL_i.isconsumed =$ False, *set* $txoL_i.isconsumed =$ True.
* *otherwise, set* $b_{\mathcal{A}win} = 1$.

(b) the oracle calls the subroutine Subroutine-ProceedOutTXO$(\cdot,\cdot)$ *in Alg. 1 with* $tx =$ MsTx *and* $txo =$ MsTx.$outtxo$ *as input.*

At last, the oracle adds MsTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OTransactPrivate()$*:* This oracle captures that an adversary can observe the private transactions created by other participants, and also captures that an adversary acts as receivers of the transaction output coins.

  The oracle first interacts with the adversary to form the inputs for the transaction:

  (a) The oracle choose n *random tuples from* $TxoL$, *say* $\{csdtxoL_1, \ldots, csdtxoL_n\}$, *such that* $n \geq 1$ *and* $csdtxoL_i.isconsumed =$ False AND $csdtxoL_i.coin.sk \neq null$ AND $csdtxoL_i.txo.valuehidden =$ True, $(1 \leq i \leq n)$. *Set* $intxo[i] = csdtxoL_i.txo, csdcn[i] = csdtxoL_i.coin$ $(1 \leq i \leq n)$. *Let* $value_{in} = \sum_{i=1}^{n} csdcn[i].value$, *the oracle chooses random* $value_{out}$ *such that* $0 < value_{out} < value_{in}$, *and gives* $value_{out}$ *to* $\mathcal{A}$.

  (b) The adversary specifies a set of (pseudonym address, value), say $out[]$, *with size* $m_0 > 1$, *such that for* $j = 1$ *to* m_0, $out[j].pdnaddr = pdnL.pdnaddr$ *for some* $pdnL \in PdnL$, *or* $out[j].pdnaddr = pdnaddrL_{\mathcal{A}}$ *for some* $pdnaddrL_{\mathcal{A}} \in PdnaddrL_{\mathcal{A}}$, $0 < out[j].value \leq value_{out}$, *and* $out[j].pdnaddr \neq out[j'].pdnaddr$ *for* $1 \leq j \neq j' \leq m_0$, *and*

$\sum_{j=1}^{m_0} out[j].value = value_{out}$.

The oracle chooses a random tuple $pdnL_m \in PdnL$ *and a random value* $value_m$ *such that* $0 < value_m < value_{in} - value_{out}$*, adds* $(pdnL_m.pdnaddr, value_m)$ *to out[], and sets* $m = m_0 + 1$*. This is used to receive the change.*

The oracle sets $value_{fee} = value_{in} - value_{out} - value_m$.

The oracle randomly arranges the order of the elements in $out[]$.

(c) The oracle chooses $(t-1) \times n$ *random tuples from* $TxoL$*, say* $\{txoL_i\}_{i=n+1}^{tn}$*, such that* $t \geq 1$ *and* $txoL_i.txo.valuehidden = \mathsf{True}$*,* $(n+1 \leq i \leq tn)$*. Set* $intxo[i] = txoL_i.txo$ $(n+1 \leq i \leq tn)$*. Let* $In_1 = (intxo[1], \ldots, intxo[n])$*,* $In_i = (intxo[(i-1)n+1], \ldots, intxo[i \times n])$ $2 \leq i \leq t$*, the oracle randomly arrange the order of* $In_1, \ldots, In_t$ *in* $intxo[]$.

The adversary specifies the tracking option by setting $trackable \in \{\mathsf{True}, \mathsf{False}\}$. *The oracle then runs* $(outtxo[], \mathsf{PriTx}) \leftarrow \mathsf{TransactPrivate}(\mathsf{PP}, tn, n, intxo[], csdcn[], m, out[], value_{fee}, [\mathsf{PK}_T])$*, where* $\mathsf{PriTx} = (tn, n, intxo[]; m, outtxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$*, and* $[\mathsf{PK}_T] = \mathsf{PK}_T$ *if* $trackable = \mathsf{True}$*. The oracle performs*

(a) for $i = 1$ *to* n*,*

updates $csdtxoL_i$ *(in* $TxoL$*) by setting* $csdtxoL_i.isconsumed = \mathsf{True}$.

(b) for $j = 1$ *to* m*, the oracle calls the* Subroutine-ProceedOutTXO$(\cdot, \cdot)$ *in Alg. 1 with* $tx = \mathsf{PriTx}$ *and* $txo = \mathsf{PriTx}.outtxo[j]$ *as input.*

At last, the oracle adds PriTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OAddPriTx(\cdot)$*:* This oracle captures that the adversary can produce a private transaction PriTx and ask this oracle to add PriTx to the ledger.

  $OAddPriTx(\mathsf{PriTx})$ *first runs* $b \leftarrow \mathsf{VerifyTransaction}(\mathsf{PP}, \mathsf{PriTx}, L)$*, where* $\mathsf{PriTx} = (tn, n, intxo[]; m, outtxo[], value_{fee}; spf, csdcnsn[]; trackable, tif; *)$. *If* $b = 0$*, the oracle returns* $\perp$ *to* $\mathcal{A}$*, otherwise,*

(a) *for* $i = 1$ *to* tn*: find a tuple* $txoL_i \in \mathsf{TxoL}$ *such that* $txoL_i.txo = \mathsf{PriTx}.intxo[i]$*, if such a* $txoL_i.txo$ *does not exist, set* $b_{\mathcal{A}win} = 1$*, otherwise*

- *If* $txoL_i.coin.sk \neq null$*, compute* $sn = \mathsf{ComputeCoinSN}(\mathsf{PP}, txoL_i.coin.addr, txoL_i.coin.sk)$.
  - *If* $sn \in \mathsf{PriTx}.csdcnsn[]$,
    - *if* $txoL_i.txo.valuehidden = \mathsf{TRUE}$ AND $txoL_i.iscorrupted = \mathsf{TRUE}$ AND $txoL_i.isconsumed = \mathsf{False}$, *update* $txoL_i.isconsumed = \mathsf{True}$ *in* $TxoL$. Note that this captures that one consumed coin is from the corrupted ones.
    - *otherwise, set* $b_{\mathcal{A}win} = 1$.
  - *otherwise, do nothing.*
- *otherwise, do nothing.*

(b) *for* $j = 1$ *to* m*, the oracle calls the* $\mathsf{Subroutine\text{-}ProceedOutTXO}(\cdot, \cdot)$ *in Alg. 1 with* $tx = \mathsf{PriTx}$ *and* $txo = \mathsf{PriTx}.outtxo[j]$ *as input.*

At last, the oracle adds PriTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OUnmask()$*:* This oracle captures that the adversary can observe the unmask transactions created by other participants.

  The oracle first chooses the input parameters randomly:

  (a) *Choose a random tuple from* $TxoL$*, say* $intxoL$*, such that* $intxoL.txo.valuehidden = \mathsf{True}$ AND $intxoL.isconsumed = \mathsf{False}$ AND $intxoL.coin.sk \neq null$*. Set* $intxo = txoL.txo, csdcn = txoL.coin$.

  (b) *Choose a random tuple from* $PdnL$*, say* $pdnL$*. Choose a random value* $outvalue$ *such that* $0 < outvalue \leq csdcn.value$.

  The oracle then runs $(outtxo, \mathsf{UmTx}) \leftarrow \mathsf{Unmask}(\mathsf{PP}, intxo, csdcn, outvalue, pdnL.pdnaddr)$*, where* $\mathsf{UmTx} = (intxo, incnvalue; outtxo; spf, csdcnsn; *)$,

$incnvalue = csdcn.value$, $csdcnsn =$ ComputeCoinSN(PP, $csdcn.addr$, $csdcn.sk$), *and*

(a) updates $intxoL$ *(in* $TxoL$*) by setting* $intxoL.isconsumed =$ True.

(b) runs $cn \leftarrow$ Receive(PP, $pdnL.pdnaddr, pdnL.pdnsk$, UmTx, UmTx.$outtxo$), *and adds* (UmTx.$outtxo$, False, $cn, pdnL.iscorrupted, pdnL.pdnaddr$) *into* $TxoL$.

At last, the oracle adds UmTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OAddUmTx(\cdot)$*:* This oracle captures that the adversary can produce a unmask transaction UmTx and ask this oracle to add UmTx to the ledger.

  $OAddUmTx$(UmTx) *first runs* $b \leftarrow$ VerifyTransaction(PP, UmTx, L), *where* UmTx $= (intxo, incnvalue;\ outtxo; spf, csdcnsn; *)$. *If* $b = 0$, *the oracle returns* $\perp$ *to* $\mathcal{A}$, *otherwise,*

  (a) finds a tuple $txoL \in$ TxoL *such that* $txoL.txo =$ UmTx.$intxo$, *if such a* $txoL.txo$ *does not exist, set* $b_{\mathcal{A}win} = 1$, *otherwise*

  – *If* $txoL.coin.sk \neq null$, *compute* $sn =$ ComputeCoinSN(PP, $txoL.coin.addr, txoL_i.coin.sk$).

  * *If* $sn = csdcnsn$,

  · *if* $txoL.txo.valuehidden =$ TRUE AND $txoL.iscorrupted =$ TRUE AND $txoL.isconsumed =$ False, *update* $txoL.isconsumed =$ True *in* $TxoL$.

  · *otherwise, set* $b_{\mathcal{A}win} = 1$.

  * *otherwise, set* $b_{\mathcal{A}win} = 1$.

  – *otherwise, do nothing.*

  (b) calls the subroutine Subroutine-ProceedOutTXO$(\cdot,\cdot)$ *in Alg. 1 with* $tx =$ UmTx *and* $txo =$ UmTx.$outtxo$ *as input.*

  At last, the oracle adds UmTx *into* L *and returns* L *to* $\mathcal{A}$.

- $OCorruptPseudonym(\cdot)$: This oracle captures that an adversary may corrupt other participants' pseudonym secret keys, and as a result, all the derived coin secret keys are compromised.

  $OCorruptPseudonym(pdnaddr)$ looks up $PdnL$ and finds the tuple corresponding to $pdnaddr$, say $(pdnaddr, pdnsk, \mathsf{False})$. If the tuple does not exist, the oracle returns $\perp$ to A, otherwise,

  - *The oracle updates the tuple to $(pdnaddr, pdnsk, \mathsf{Ture})$ and returns $pdnsk$ to $\mathcal{A}$;*
  - *The oracle finds all the $txoL \in TxoL$ such that $txoL.pdnaddr = pdnaddr\ AND\ txoL.coin.sk \neq null$, updates $txoL$ by setting $txoL.iscorrupted = \mathsf{True}$, and gives $txoL.coin.sk$ to $\mathcal{A}$.*

- $OCorruptCoin(\cdot)$: This oracle captures that an adversary may corrupt the coin secret key of a particular TXO.

  $OCorruptCoin(txo)$ looks up $TxoL$ and finds the tuple corresponding to txo, say $txoL$, if $txoL.coin.sk \neq null$, then sets $txoL.iscorrupted =$ True and returns $txoL.coin.sk$ to $\mathcal{A}$. If such a $txoL$ does not exist or $txoL.coin.sk = null$, the oracle returns $\perp$ to A.

  Remark: *Giving the adversary the oracle $OCorruptCoin(\cdot)$ is to capture the security requirement that the coin addresses derived from the same pseudonym address should be independent from each other, and even one of them is compromised, the others are still in safe. CryptoNote and Monero did not achieve this security, since if a transaction sender corrupts a coin for a pseudonym address (say pdnaddr) in a transaction, then he can compute the value b, and as he also knows the random r for other coin address he derived for the same pseudonym address, he can compute the coin secret key $H(rA) + b$ and can spend the coin.*

3. **Output Phase.** *At some time T, $\mathcal{A}$ declares the end of the game, and the current ledger is L_T. The algorithm 2 CheckLedger() is run. $\mathcal{A}$ succeeds if*

$b_{\mathcal{A}win} = 1$, *or* $rangerror = 1$, *or* $v_T > v_{total}$. Note that $b_{\mathcal{A}win} = 1$ captures the adversary's successful attacks that can be detected directly, while $rangerror = 1$ and $v_T > v_{total}$ captures that without the adversary's secret keys, we cannot identity what attacks the adversary launched, but only if (as long as) it breaks the range or total value rules, we know it launched some attack successfully.

A DAPOA scheme is secure if, for any PPT $\mathcal{A}$*, the success probability of* $\mathcal{A}$ *in the above game is negligible.*

Note that the TXOs received and controlled by adversary's pseudonyms are not used to compute the $rangerror$ and v_T. The thoughts behind this is that only the values transferred to the pseudonyms that the adversary does not control are 'effective' values.

Note that the security definition also include resistance against a tracking authority.

2.5 Fine-grained Privacy of DAPOA

DAPOA provisions fine-grained privacy, with three different levels:

- **Basic Privacy**: Coinbase, mask, unmask, and public transactions use one-time coin address to hide the pseudonyms of the receivers, i.e. provides pseudonym-anonymity, which is similar but stronger than the pseudonym-anonymity of Bitcoin.

- **Full Privacy**: Private transaction generated without using the tracking public key provides full privacy, where the consumed coins, the values of the generated coins, and the pseudonyms of the coin receivers are completely hidden.

- **Full Privacy with Accountability**: Private transactions generated with the tracking public key provides full privacy against the participants who

Algorithm 2 *CheckLedger*()

1: $v_T \leftarrow 0$, $rangerror \leftarrow 0$
2: **for** each $txoL \in TxoL$ **do**
3: **if** $txoL.coin.sk \neq null\ AND\ txoL.isconsumed =$ False **then**
4: **if** $txoL.coin.value < 0\ OR\ txoL.coin.value > v_{max}$ **then**
5: $rangerror \leftarrow 1$
6: **else**
7: $v_T \leftarrow v_T + txoL.coin.value$
8: **end if**
9: **end if**
10: **end for**
11: **for** each transaction $tx \in L_T$ **do**
12: **if** tx is mask, unmask, public, or private transaction **then**
13: let fee_{tx} be the transaction fee of tx
14: $v_T \leftarrow v_T + fee_{tx}$
15: **end if**
16: **end for**
17: returns v_T and $rangerror$

do not know the tracking secret key, and basic privacy to the authority who knows the tracking secret key.

Below we present the definitions for the basic privacy, full privacy, and full privacy with accountability.

2.5.1 Pseudonym-anonymity

Definition 2 *Given a secure*[11] *DAPOA scheme* (Setup, CreateTrackingKey, CreatePseudonym, DeriveCoinAddress, VerifyCoinAddress, DeriveCoinSK, HideCoinValue, OpenCoinValue, ComputeCoinSN, Coinbase, TransactPublic, Mask, Unmask, TransactPrivate, VerifyTransaction, Receive, TrackTransaction, VerifyTrack)*, and a PPT adversary $\mathcal{A}$, consider the following game:*

1. **Setup.** *Same as that of Def. 1.*
2. **Phase 1.** *Same as that of Def. 1.*
3. **Challenge Phase.** *$\mathcal{A}$ outputs a transaction type $TxType \in \{Coinbase, Mask, Unmask, Public, Private\}$ and two different pseudonym address $pdnaddr_0$, $pdnaddr_1$ such that there are corresponding tuples $pdnL_0, pdnL_1 \in PdnL$ such that $pdnL_0.iscorrupted = pdnL_1.iscorrupted =$ False.*
   - *if $TxType = Coinbase$: random bit $b \in \{0,1\}$ is chosen, $OCoinbase()$ is performed, where the $pdnaddr_b$ is used as the receiver pseudonym address.*
   - *if $TxType = Mask$: random bit $b \in \{0,1\}$ is chosen, $OMask()$ is performed, where the $pdnaddr_b$ is used as the receiver pseudonym address.*
   - *if $TxType = Unmask$: random bit $b \in \{0,1\}$ is chosen, $OUnmask()$ is performed, where the $pdnaddr_b$ is used as the receiver pseudonym address.*

[11] Here we assume the DAPOA is secure as Def. 1.

- *if $TxType = Public$: random bit $b \in \{0,1\}$ is chosen, $OTransactPublic()$ is performed, where $pdnaddr_b$ is used as the pseudonym address for receiving the change.*
- *if $TxType = Private$: random bit $b \in \{0,1\}$ is chosen, $OTransactPrivate()$ is performed, where $pdnaddr_b$ is used as the pseudonym address for receiving the change.*

4. **Phase 2.** *Same as* Phase 1*, except that*

   - *$OCorruptPseudonym(\cdot)$ cannot be queried on $pdnaddr_0$ or $pdnaddr_1$.*
   - *$OCorruptCoin(\cdot)$ cannot be queried on the challenge* Txo*:* [12]
     - *if $TxType = Coinbase$: suppose the transaction is* CbTx^**, $OCorruptCoin(\cdot)$ cannot be queried on* $\mathsf{CbTx}^*.outtxo$.
     - *if $TxType = Mask$: suppose the transaction is* MsTx^**, $OCorruptCoin(\cdot)$ cannot be queried on* $\mathsf{MsTx}^*.outtxo$.
     - *if $TxType = Unmask$: suppose the transaction is* UmTx^**, $OCorruptCoin(\cdot)$ cannot be queried on* $\mathsf{UmTx}^*.outtxo$.
     - *if $TxType = Public$: suppose the transaction is* PubTx^**, $OCorruptCoin(\cdot)$ cannot be queried on* $\mathsf{PubTx}^*.outtxo[j^*]$*, where* $\mathsf{PubTx}^*.outtxo[j^*]$ *is the* TXO *for receiving the change.*
     - *if $TxType = Private$: suppose the transaction is* PriTx^**, $OCorruptCoin(\cdot)$ cannot be queried on* $\mathsf{PriTx}^*.outtxo[j^*]$*, where* $\mathsf{PriTx}^*.outtxo[j^*]$ *is the* TXO *for receiving the change.*

5. **Output Phase.** *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, and succeeds if $b' = b$.*

A DAPOA scheme achieves pseudonym-anonymity if, for any PPT $\mathcal{A}$, the success probability of $\mathcal{A}$ in the above game is negligibly close to $1/2$.

[12] If we remove this restriction, the privacy is stronger, i.e. even given the coin secret key, the adversary cannot tell which pseudonym address the coin public key is derived from.

2.5.2 Value Hiding

Note that when an adversary corrupts the related coins of a target coin, he can guess the value range of the target coin. The value-hiding definition below requires that unless an adversary corrupts all the related coins, he cannot break the value-hiding.

In the following security model, we model a private transaction, where the adversary controls all the input and consumed TXOs. Let v_{in} be the total value of the consumed coins, the adversary specifies three values v_0, v_1, v_{fee}. A random bit $b \in \{0, 1\}$ is chosen, and the private transaction has two output TXOs, one with coin value v_b and one with coin value $v_{in} - v_b - v_{fee}$, and the target of the adversary is to guess the value of b. Note that wlog. it is enough for the adversary to learn information from one of the two output TXOs. The adversary is allowed to make any queries to the oracles, except it cannot consume the target TXO, say txo^*, to generate new coins that are all corrupted by the adversary, i.e., when the adversary tries to consume the target TXO, it is required that at least one output TXO is not corrupted by the adversary, and this new TXO becomes the target TXO, since once the adversary can learn enough information from this new TXO, he can track back and win the game trivially. In addition, as the values of v_0 and v_1 are specified by the adversary, the adversary can learn range information by issuing transactions to consume the target TXO, and win the game trivially. For example, the adversary may specify $v_0 = 10, v_1 = 100$, then it makes a query to produce a transaction which will contain an output TXO with coin value 90. If the oracle returns $\perp$, then the adversary can conclude that the target TXO has coin value 10, i.e. $b = 0$, otherwise it can conclude that $b = 1$. The following game captures this case.

Definition 3 *Given a secure*[13] *DAPOA scheme* (Setup, CreateTrackingKey, CreatePseudonym, DeriveCoinAddress, VerifyCoinAddress, DeriveCoinSK, HideCoinValue,

[13]Here we assume the DAPOA is secure as Def. 1.

OpenCoinValue, ComputeCoinSN, Coinbase, TransactPublic, Mask, Unmask, TransactPrivate, VerifyTransaction, Receive, TrackTransaction, VerifyTrack), *and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. **Setup.** *Same as that of Def. 1.*

2. **Phase 1.** *Same as that of Def. 1.*

3. **Challenge Phase.**

   (a) $\mathcal{A}$ *outputs a set of* TXO*s in current ledger* L*, say* $\{txo[i]\}_{i=1}^{n}$*, and the corresponding coins, say* $\{csdcn[i]\}_{i=1}^{n}$*, such that for each* txo_i *there is a tuple* $txoL_i \in TxoL$*, such that* $txoL_i.txo.valuehidden =$ True *and* $sn_i =$ ComputeCoinSN$(PP, csdcn[i].addr, csdcn[i].sk)$ *does not appear in any private transaction of* L*.*

   (b) $\mathcal{A}$ *outputs three values* v_0, v_1, v_{fee} *such that* $0 < v_0, v_1, v_{fee} < v_{max}$ *and* $v_0 + v_{fee} < \sum_{i=1}^{n} csdcn[i].value, v_1 + v_{fee} < \sum_{i=1}^{n} csdcn[i].value$*.*

   (c) $\mathcal{A}$ *outputs two different pseudonym address* $pdnaddr_1, pdnaddr_2$ *such that there are corresponding tuples* $pdnL_1, pdnL_2 \in PdnL$ *such that* $pdnL_1.iscorrupted = pdnL_2.iscorrupted =$ False*.*

   (d) *a random bit* $b \in \{0, 1\}$ *is chosen, and* $OTransactPrivate()$ *is performed, where*

      i. *t=1, i.e.* $l = n$*, and* $intxo[i] = txo[i]$ $(i = 1, \ldots, n)$ *are the input* TXO*s,* $\{csdcn[i]\}_{i=1}^{n}$ *are the input consumed coins.*

      ii. *The* $out[]$ *is set as* $(out[1].pdnaddr = pdnaddr_1, out[1].value = v_b)$*,* $(out[2].pdnaddr = pdnaddr_2, out[2].value = \sum_{i=1}^{n} csdcn[i].value - (v_b + v_{fee}))$*.*

   (e) *Let* txo_1^*, txo_2^* *be the two output* TXO*s of the private transaction of above step. Wlog. assume* txo_1^* *is the one receiving value* v_b*, and this information is given to* $\mathcal{A}$*.*

(f) *Initialize incorruptible pseudonym address list as* $IcPdnAddrL = \{pdnaddr_1, pdnaddr_2\}$. *Initialize incorruptible* TXO *list as* $IcTxoL = \{txo_1^*, txo_2^*\}$. *Initialize the target* TXO txo^* *to be* txo_1^*. *Note that the target coin value is* $v^* = v_b$, *initialize the range information of the target coin value, namely* $v_{min}^* = min(v_0, v_1), v_{max}^* = max(v_0, v_1)$, *so that* v^* *satisfies* $v_{min}^* \leq v^* \leq v_{max}^*$.

4. **Phase 2.** *Same as* **Phase 1**, *except that*

- $OCorruptPseudonym(\cdot)$ *cannot be queried on* $pdnaddr \in IcPdnAddrL$,
- $OCorruptCoin(\cdot)$ *cannot be queried on* $txo \in IcTxoL$,
- $OUnmask()$ *does not choose* $txo \in IcTxoL$ *as consumed* TXO.
- $OTransactprivate()$ *does not choose* $txo \in IcTxoL$ *as consumed* TXO.
- *If the adversary wants a private transaction to consume the* txo^*, [14]

  (a) $\mathcal{A}$ *outputs a set of* TXO*s in current ledger* L, *say* $\{txo[i]\}_{i=1}^{n_0}$, *and the corresponding coins, say* $\{csdcn[i]\}_{i=1}^{n_0}$, *such that for each* txo_i *there is a tuple* $txoL_i \in TxoL$, *such that* $txoL_i.txo.valuehidden =$ True *and* $sn_i =$ ComputeCoinSN$(PP, csdcn[i].addr, csdcn[i].sk)$ *does not appear in any private transaction of* L. *Let* $v_{\mathcal{A},in} = \sum_{i=1}^{n_0} csdcn[i].value$.

  (b) $\mathcal{A}$ *outputs two values* $v_{\mathcal{A},out}, v_{fee}$ *such that* $0 < v_{\mathcal{A},out}, v_{fee} < v_{max}$ *and* $v_{max}^* + v_{\mathcal{A},in} - v_{max} \leq v_{\mathcal{A},out} + v_{fee} < v_{\mathcal{A},in} + v_{min}^*$. [15] As shown below, the private transaction will consume the target TXO txo^* (with value v^*) and the coins specified above by the adversary (with total value $_{\mathcal{A},in}$), and generates two TXOs, one received by pseudonym address corrupted by the adversary (with value $v_{\mathcal{A},out}$), one received by

[14] We did not consider the situation where the adversary asks a private transaction to spend txo_2^*, as it is similar, but just requires a somewhat complicated description.

[15] This is to ensure that the adversary cannot trivially win the game using the range requirement of coin value.

pseudonym address not-corrupted by the adversary (with value $v^* + v_{\mathcal{A},in} - v_{\mathcal{A},out} - v_{fee}$), and the later will be the new target TXO.

(c) *$\mathcal{A}$ outputs a pseudonym address $pdnaddr^*$ such that there is a corresponding tuple $pdnL^*$ such that $pdnL^*.iscorrupted =$ False. $\mathcal{A}$ outputs a different pseudonym address $pdnaddr_{\mathcal{A}}$.*

(d) *$OTransactPrivate()$ is performed, where*

i. *t=1, $l = n = n_0 + 1$, and $\{intxo[i] = txo[i]\}_{i=1}^{n_0} \cup \{txo^*\}$ are the input TXOs, $\{csdcn[i]\}_{i=1}^{n_0} \cup \{csdcn^*\}$ are the input consumed coins, where $csdcn^*$ is the coin implied by txo^*.*

ii. *The $out[]$ is set as $(out[1].pdnaddr = pdnaddr^*, out[1].value = v^* + v_{\mathcal{A},in} - v_{\mathcal{A},out} - v_{fee})$, $(out[2].pdnaddr = pdnaddr_{\mathcal{A}}, out[2].value = v_{\mathcal{A},out})$.*

(e) *Let txo_1, txo_2 be the two output TXOs of the private transaction of above step, where txo_1 is the one received through $pdnaddr^*$, and this information is given to $\mathcal{A}$.*

(f) *$pdnaddr^*$ is added into $IcPdnAddrL$. txo_1 is added into $IcTxoL$. txo^* is set to txo_1. Note that the value of the target TXO is $v^* = v^* + v_{\mathcal{A},in} - v_{\mathcal{A},out} - v_{fee}$, set $v^*_{min} = v^*_{min} + v_{\mathcal{A},in} - v_{\mathcal{A},out} - v_{fee}$, $v^*_{max} = v^*_{max} + v_{\mathcal{A},in} - v_{\mathcal{A},out} - v_{fee}$.*

5. **Output Phase.** *$\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, and succeeds if $b' = b$, under the restrictions that*

- *In* **Setup** *phase, the tracking secret key SK_T is not given to $\mathcal{A}$; or*
- *In* **Challenge Phase**, *the $OTransactPrivate()$ oracle runs the* TransactPrivate *algorithm with optional parameter tracking public key $\mathsf{PK}_T = null$, and in* **Phase 2**, *when the adversary makes a query to consume the txo^*, the $OTransactPrivate()$ oracle runs the* TransactPrivate *algorithm with optional parameter tracking public key $\mathsf{PK}_T = null$.*

A DAPOA scheme achieves value-hiding if, for any PPT $\mathcal{A}$, the success probability of $\mathcal{A}$ in the above game is negligibly close to $1/2$.

2.5.3 Consumed-Coins-Hiding

Definition 4 *Given a secure*[16] *DAPOA scheme* (Setup, CreateTrackingKey, CreatePseudonym, DeriveCoinAddress, VerifyCoinAddress, DeriveCoinSK, HideCoinValue, OpenCoinValue, ComputeCoinSN, Coinbase, TransactPublic, Mask, Unmask, TransactPrivate, VerifyTransaction, Receive, TrackTransaction, VerifyTrack)*, and a PPT adversary $\mathcal{A}$, consider the following game:*

1. **Setup.** *Same as that of Def. 1.*
2. **Phase 1.** *Same as that of Def. 1.*
3. **Challenge Phase.**
   (a) *A copy of $TxoL$ is made, say $TxoLCP$. For each $txoLCP \in TxoLCP$, if $txoLCP.iscorrupted =$ False, set $txoLCP.coin.sk = null$. $TxoLCP$ is given to $\mathcal{A}$.*
   (b) *$\mathcal{A}$ outputs two set of tuples selected from $TxoLCP$, say $(txoLCP_1, \ldots, txoLCP_n)$ and $(txoLCP_{n+1}, \ldots, txoLCP_{2n})$, such that $txoLCP_i.txo.valuehidden =$ True AND $txoLCP_i.iscorrupted =$ False AND $txoLCP_i.coin.sk = null$ $(i \in \{1, \ldots, n\})$ AND $\sum_{i=1}^{n} txoLCP_i.coin.value = \sum_{i=n+1}^{2n} txoLCP_i.coin.value$.*
   (c) *$\mathcal{A}$ outputs $t \times n$ tuples selected from $TxoLCP$, say $txoLCP_{2n+1}, txoLCP_{2n+2}, \ldots, txoLCP_{n(t+2)}$ such that $txoLCP_i.valehidden =$ True.*
   (d) *a random bit $b \in \{0, 1\}$ is chosen, and $OTransactPrivate()$ is performed, where*

[16] Here we assume the DAPOA is secure as Def. 1.

i. $l = (t+2)n$. *For* $i = 1$ *to* $(t+2)n$, *let* $txoL_i \in TxoL$ *be the tuple corresponding to* $txoLCP_i$, *set* $intxo[i] = txoL_i.txo$, $csdcn[i] = txoL_{b\times n+i}.coin$. *Let* $intxo[]$ *be the input* TXO*s, and* $csdcn[]$ *be the consumed coins.*

4. **Phase 2.** *Same as* **Phase 1**, *except that*

- $OCorruptPseudonym(\cdot)$ *cannot be queried on* $txoL_i.pdnaddr$ $(i \in \{1, \ldots, 2n\})$,
- $OCorruptCoin(\cdot)$ *cannot be queried on* $txoL_i.txo$ $(i \in \{1, \ldots, 2n\})$,
- $OUnmask()$ *does not choose* $txoL_i.txo$ $(i \in \{1, \ldots, 2n\})$ *as consumed* TXO.
- $OTransactprivate()$ *does not choose* $txoL_i.txo$ $(i \in \{1, \ldots, 2n\})$ *as consumed* TXO.

5. **Output Phase.** $\mathcal{A}$ *outputs a bit* $b' \in \{0, 1\}$, *and succeeds if* $b' = b$, *under the restrictions that*

- *the tracking secret key* SK_T *is not given to* $\mathcal{A}$; *or*
- *In* **Challenge Phase**, *the* $OTransactPrivate()$ *oracle runs the* TransactPrivate *algorithm with the optional parameter tracking public key* $\mathsf{PK}_T = null$.

A DAPOA scheme achieves consumed-coins-hiding if, for any PPT $\mathcal{A}$, *the success probability of* $\mathcal{A}$ *in the above game is negligibly close to* $1/2$.

2.6 Accountability of DAPOA

First we define accountability which captures that no PPT adversary can produce a valid private transaction that can escape from tracking, even if it corrupts the tracking secret key.

Definition 5 *Given a secure*[17] *DAPOA scheme* ($\mathsf{Setup}, \mathsf{CreateTrackingKey}, \mathsf{CreatePseudonym}, \mathsf{DeriveCoinAddress}, \mathsf{VerifyCoinAddress}, \mathsf{DeriveCoinSK}, \mathsf{Coinbase}, \mathsf{TransactPublic}, \mathsf{Mask}, \mathsf{Unmask}, \mathsf{TransactPrivate}, \mathsf{VerifyTransaction}, \mathsf{Receive}, \mathsf{TrackTransaction}, \mathsf{VerifyTrack}$)*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. **Setup.** *Same as that of Def. 1.*
2. **Phase 1.** *Same as that of Def. 1.*
3. **Output Phase.** *At some time* T*,* $\mathcal{A}$ *declares the end of the game. Let the current ledger is* L_T*.* $\mathcal{A}$ *outputs a private transaction* $\mathsf{PriTx}^* \in L_T$*.* [18] $\mathcal{A}$ *succeeds, if*
   $\mathsf{PriTx}^*.trackable = \mathsf{True}$ *and*
   $\mathsf{VerifyTrack}(\mathsf{PP}, \mathsf{PK}_T, \mathsf{PriTx}^*, L_T, \mathsf{TrackTransaction}\ (\mathsf{PP}, \mathsf{PriTx}^*, L_T, \mathsf{SK}_T)) = 0.$

A DAPOA scheme is accountable if, for any PPT $\mathcal{A}$*, the success probability of* $\mathcal{A}$ *in the above game is negligible.*

Now we define the undeniable accountability which captures that no PPT adversary can propose two different tracking results for a valid private transaction, even if it corrupts the tracking secret key. This implies that if a tracking result is verified to be valid, then it is undeniable.

Definition 6 *Given a secure*[19] *DAPOA scheme* ($\mathsf{Setup}, \mathsf{CreateTrackingKey}, \mathsf{CreatePseudonym}, \mathsf{DeriveCoinAddress}, \mathsf{VerifyCoinAddress}, \mathsf{DeriveCoinSK}, \mathsf{HideCoinValue}, \mathsf{OpenCoinValue}, \mathsf{ComputeCoinSN}, \mathsf{Coinbase}, \mathsf{TransactPublic}, \mathsf{Mask}, \mathsf{Unmask}, \mathsf{TransactPrivate}, \mathsf{VerifyTransaction}, \mathsf{Receive}, \mathsf{TrackTransaction}, \mathsf{VerifyTrack}$)*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. **Setup.** *Same as that of Def. 1.*

[17] Here we assume the DAPOA is secure as Def. 1.
[18] Note that $\mathsf{PriTx}^* \in L_T$ implies $\mathsf{VerifyTransaction}\ (\mathsf{PP}, \mathsf{PriTx}^*, L_T) = 1$.
[19] Here we assume the DAPOA is secure as Def. 1.

2. **Phase 1.** *Same as that of Def. 1.*

3. **Output Phase.** *At some time T, $\mathcal{A}$ declares the end of the game. Let the current ledger is L_T. $\mathcal{A}$ outputs a private transaction* $\mathsf{PriTx}^* \in L_T$ *and two tuples* $(csdtxo_1[], pubtxo_1[]) \neq (csdtxo_2[], pubtxo_2[])$. *$\mathcal{A}$ succeeds, if* $\mathsf{PriTx}^*.trackable = \mathsf{True}$ *and* $\mathsf{VerifyTrack}\ (\mathsf{PP}, \mathsf{PK}_T, \mathsf{PriTx}^*, L_T, csdtxo_i[], pubtxo_i[]) = 1\ (i = 1, 2)$. [20]

A DAPOA scheme achieves undeniable accountability if, for any PPT $\mathcal{A}$, the success probability of $\mathcal{A}$ in the above game is negligible.

[20] Note that $\mathsf{PriTx}^* \in L_T$ implies $\mathsf{VerifyTransaction}\ (\mathsf{PP}, \mathsf{PriTx}^*, L_T) = 1$.

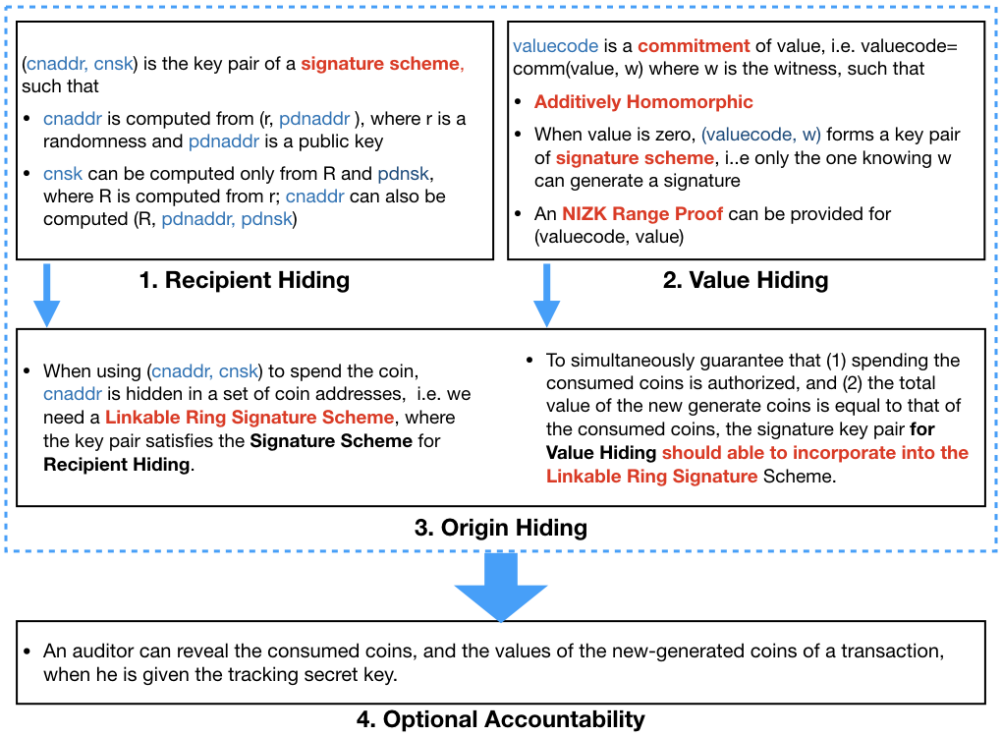

Figure 2: Abelian Cryptographic Freamework

3 Building Blocks

Based on previous retrospect, Abelian is a DAPOA system which (1) borrows Monero's ideas on privacy, (2) designs and implements the cryptographic primitives using lattice-based cryptography.

3.1 Abelian Cryptographic Framework

As shown in Fig. 2,

- A Signature Scheme, denoted Π_{rhsig}, whose (public key, secret key) pair can be generated from a (pseudonym public key, pseudonym secret key) pair, is used to achieve recipient hiding, i.e. cut the relation between the coin address and pseudonym.

- An additively Homomorphic Commitment Scheme, denoted by Π_{vhcm}, is used to achieve value hiding, due to the hiding and binding properties of commitment scheme. In addition, when the message committed is zero, the commitment and the witness forms a (public key pair, secret key pair) of a signature scheme, denoted by $\Pi_{vhcmsig}$, so that equality of consumed coin value and the generated coin value can be proved. Also, a NIZK-Proof scheme is used to provide range proof for the generated coin values, and the NIZK-Proof should be built on the commitment scheme Π_{vhcm} as well.

- A Linkable Ring Signature Scheme $\Pi_{ohlrsig}$ is used to achieve origin hiding, where the consumed coins are hidden in a larger set of coins. As Π_{rhsig} is used to authorize consuming a coin, $\Pi_{ohlrsig}$ is built on Π_{rhsig}. Also, authorizing consuming coins, equality proof, and range proof should be achieved simultaneously, the linkable ring signature scheme $\Pi_{ohlrsig}$ also incorporates $\Pi_{vhcmsig}$.

- The signature schemes $\Pi_{rhsig}, \Pi_{vhcmsig}$ and the linkable ring signature scheme $\Pi_{ohlrsig}$ have a built-in tracking mechanism, so that an auditor, given the tracking secret key, can reveal the consumed coins and value of the generated coins.

3.2 Lattice-based Cryptographic Building Blocks

The state-of-the-art privacy-preserving cryptosystems with post-quantum security belong to two main classes. The first class relies entirely on symmetric-key primitives. Protocols in this class demand zero-knowledge proofs for circuit-like statements [40, 22, 2], leading to constructions such as ring signature [74] and group signature [25] systems with increasingly improved efficiency [28, 17, 43]. Despite these recent advancements, this approach seems unsuitable to be used for the development of Abelian. The main reason is that Abelian requires to work with primitives possessing rich algebraic structures, such as signature schemes

with key derivation and additively homomorphic commitments, while the existing symmetric-key-based zero-knowledge protocols are not known to interact well with algebraic statements [21].

The second class relies on lattice-based public-key primitives. Earlier lattice-based zero-knowledge protocols, e.g., [57, 44, 52, 53], were quite impractical, but the field has been rapidly developed, and some protocols [61, 27, 7] can already be considered for practical use. These protocols naturally interact smoothly with lattice-based statements: in fact, their newly obtained efficiency is thanks to the algebraic structures of lattices and clever treatments of Lyubashevsky's rejection-sampling techniques [57, 58, 59].

Abelian is built on lattice-based cryptography. In the following, we review the necessary ingredients for instantiating our system from lattices, and discuss the known results, the remaining research problems and our approach to address each of these problems.

3.2.1 Module-SIS and Module-LWE

Lattice-based cryptography relies on variants of the Short Integer Solution (SIS) problem [1] and the Learning With Errors (LWE) problem [72]. The original, unstructured variants of SIS and LWE provide very strong security guarantees from general lattices [1, 72], but typically lead to systems with relatively large key sizes. Their variants Ring-SIS [60, 71] and Ring-LWE [62], enable systems with significantly smaller key sizes, but provide less security confidence [18] due to the reliance on ideal lattices.

Many recently proposed systems [19, 31, 7, 27] instead rely on the Module-SIS and Module-LWE problems [46] - variants of SIS and LWE that enjoy the best of both worlds (i.e., high security guarantees and practicality).

[21] Chase et al. [23] proposed an interesting method to combine non-algebraic and algebraic statements based on garbled circuits, but it is inherently interactive and thus unsuitable for the design of signatures.

Rings and norms. Let n, q be positive integers and denote by $\mathbb{Z}_q$ the integers modulo q, which will be represented between $-\lfloor \frac{q-1}{2} \rfloor$ and $\lfloor \frac{q+1}{2} \rfloor$. Let R and R_q be the rings $\mathbb{Z}[X]/(X^n+1)$ and $\mathbb{Z}_q[X]/(X^n+1)$, respectively. For $w = w_0 + w_1 X + \ldots + w_{n-1} X^{n-1} \in R$, define the infinity norm and the Euclidean norm of w as follows:

$$\|w\|_\infty = \max_i \|w_i\|_\infty, \quad \|w\| = \sqrt{\|w_0\|_\infty^2 + \ldots + \|w_{n-1}\|_\infty^2}.$$

Similarly, for $\mathbf{w} = (w_1, \ldots, w_k) \in R^k$, define:

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty, \quad \|w\| = \sqrt{\|w_1\|^2 + \ldots + \|w_k\|^2}.$$

Let S_η denote the set of all elements $w \in R$ such that $\|w\|_\infty \leq \eta$.

(Inhomogeneous) Module-SIS. The Inhomogeneous Module-SIS problem with parameters (n, q, k, ℓ, β) consists in finding $\mathbf{x} \in R^{k+\ell}$ such that $\|\mathbf{x}\| \leq \beta$ and $[\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{x} = \mathbf{t}$, for uniformly random $\mathbf{A} \in R_q^{k \times \ell}$, $\mathbf{t} \in R_q^k$ and $k \times k$ identity matrix $\mathbf{I}$. The problem can be adapted straightforwardly into its infinity-norm version, where $\mathbf{x}$ must satisfy $\|\mathbf{x}\|_\infty \leq \beta$. The homogeneous version is defined with $\mathbf{t} = \mathbf{0}$ and $\mathbf{x} \neq \mathbf{0}$.

Module-LWE. The Module-LWE problem with parameters (n, q, k, ℓ, η) is as follows. Let $\mathbf{A} \in R_q^{k \times \ell}$ be a uniformly random matrix. Let $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \in R_q^k$, where $\mathbf{s} \in S_\eta^\ell$, $\mathbf{e} \in S_\eta^k$ have entries chosen according to some distribution over S_η (e.g., the uniform distribution or a Gaussian distribution). The search variant of Module-LWE asks to recover $\mathbf{s}$ given $(\mathbf{A}, \mathbf{b})$. The decision variant asks to distinguish $(\mathbf{A}, \mathbf{b})$ from a uniformly random pair over $R_q^{k \times \ell} \times R_q^k$.

As shown in [46], the Module-SIS and Module-LWE problems enjoy worst-case to average-case reductions from hard problems in module lattices. Concrete parameters of these problems that provide high post-quantum security against the best known attacks are given in Dilithium [31] and Kyber [19].

3.2.2 Lattice-based Signature with Key-Derivation

To support anonymous payments, we use a mechanism similar to that of CryptoNote, at the heart of which is a key exchange protocol between the sender and the receiver that allows the latter to derive a random-looking one-time key pair for a signature scheme. Since a key exchange protocol can be obtained from a key encapsulation scheme (KEM) in a modular manner, let us recall the definition of KEM.

A key encapsulation scheme $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ is a triple of polynomial-time algorithms together with a key space $\mathcal{K}$.

- The randomized key generation algorithm KeyGen outputs a key pair (pk, sk), where pk is public key and sk is secret key.
- The randomized encapsulation algorithm Encap takes as input a public key pk and outputs a ciphertext $\mathbf{c}$ and a key $\rho \in \mathcal{K}$.
- The deterministic decapsulation algorithm Decap takes as input a secret key sk and a ciphertext $\mathbf{c}$, and outputs either a key $\rho \in \mathcal{K}$ or a symbol $\perp$ indicating rejection.

For a concrete example, we consider Kyber [19], which is one of the most well studied lattice-based KEM schemes and also was included in the NIST post-quantum cryptography project. As in [19, Section 5], Kyber allows Alice and Bob to agree on a key $\rho \in \{0,1\}^{256}$ as follows.

1. Bob runs $\mathsf{Kyber.KeyGen}$ to generate (pk, sk), and sends pk to Alice.
2. Alice runs $\mathsf{Kyber.Encap}(pk)$ to obtain $(\mathbf{c}, \rho)$, and sends $\mathbf{c}$ to Bob.
3. Bob runs $\mathsf{Kyber.Decap}(sk, \mathbf{c})$ to obtain ρ.

The decapsulation algorithm of Kyber never returns $\perp$. Instead, in case the algorithm fails, it outputs a random element of the key space $\mathcal{K} = \{0,1\}^{256}$. According to [19, Table 3], a Kyber variant achieving 161 bits of post-quantum security

against best known attacks features sk size 2368 bytes, pk size 1088 bytes, and ciphertext $\mathbf{c}$ of size 1184 bytes.

Deriving random-looking keys for lattice-based signatures. Fiat-Shamir-with-abort, put forward by Lyubashevsky [58, 59], is a common approach to construct a practical lattice-based signature scheme. For example, Dilithium [31], which was included in the NIST post-quantum cryptography project, is one of the most efficient schemes, thanks to various dedicated optimization techniques. The basic ideas underlying the scheme are as follows.

The secret signing key is a pair $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^\ell \times S_\eta^k$, where η, ℓ, k are small positive integers and S_η is as defined earlier. The public verification key is vector $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \in R_q^k$, where $\mathbf{A} \in R_q^{k \times \ell}$ is a publicly computable matrix. To sign a message, the signer generates a proof of knowledge of a valid secret key, which is made non-interactive via the Fiat-Shamir transformation. With appropriate settings of parameters, the scheme is proven secure in the quantum random oracle model assuming the hardness of the Module-SIS and Module-LWE problems. For the recommended parameters given in [31, Table 1], where $(n, q, \eta, \ell, k) = (256, 8380417, 5, 4, 5)$, Dilithium features public key size 1472 bytes and signature size 2700 bytes.

As in CryptoNote, we use a mechanism in which the sender can use the receiver's long-term public key $\mathbf{t}_0$ to generate a random-looking public key $\mathbf{t}$. It is required that only the receiver can recognize that $\mathbf{t}$ is his one-time public key and can compute the corresponding secret key $(\mathbf{s}_1, \mathbf{s}_2)$. Such a mechanism can be realized based on Kyber and Dilithium, as follows. Each potential receiver has two key pairs:

- A Kyber key pair $(pk, sk) \leftarrow \mathsf{Kyber.KeyGen}$.
- A Dilithium key pair $\big(\mathbf{t}_0, (\mathbf{s}_{1,0}, \mathbf{s}_{2,0})\big)$, where $(\mathbf{s}_{1,0}, \mathbf{s}_{2,0}) \in S_\eta^\ell \times S_\eta^k$ and $\mathbf{t}_0 = \mathbf{A}\mathbf{s}_{1,0} + \mathbf{s}_{2,0} \in R_q^k$.

Then, a sender and a receiver proceed as follows.

- The sender runs $\mathsf{Kyber.Encap}(pk)$ to obtain $(\mathbf{c}, \rho)$. Then she uses $\rho \in \{0,1\}^{256}$ as the random seed to deterministically generate $(\mathbf{s}_{1,1}, \mathbf{s}_{2,1}) \in S_\eta^\ell \times S_\eta^k$ (see [31] for the implementation of this "extendable output function"), and computes $\mathbf{t}_1 = \mathbf{A}\mathbf{s}_{1,1} + \mathbf{s}_{2,1} \in R_q^k$ and $\mathbf{t} = \mathbf{t}_0 + \mathbf{t}_1 \in R_q^k$. Next, she publishes $(\mathbf{c}, \mathbf{t})$.

- Given a transaction associated with $(\mathbf{c}, \mathbf{t})$, the receiver first runs the decapsulation algorithm $\mathsf{Kyber.Decap}(sk, \mathbf{c})$ to obtain a key $\rho' \in \{0,1\}^{256}$, which allows him to generate $(\mathbf{s}'_{1,1}, \mathbf{s}'_{2,1}) \in S_\eta^\ell \times S_\eta^k$. Then he checks whether it holds that $\mathbf{t} = \mathbf{t}_0 + \mathbf{A}\mathbf{s}'_{1,1} + \mathbf{s}'_{2,1}$. If this is the case, then he determines that he is indeed the target receiver, and computes $(\mathbf{s}_1, \mathbf{s}_2) \in S_{2\eta}^\ell \times S_{2\eta}^k$, where $\mathbf{s}_1 = \mathbf{s}_{1,0} + \mathbf{s}'_{1,1}$ and $\mathbf{s}_2 = \mathbf{s}_{2,0} + \mathbf{s}'_{2,1}$. Note that

$$\mathbf{t} = \mathbf{A}\mathbf{s}_{1,0} + \mathbf{s}_{2,0} + \mathbf{A}\mathbf{s}'_{1,1} + \mathbf{s}'_{2,1} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2.$$

The above mechanism allows the receiver to obtain a key pair $\big(\mathbf{t}, (\mathbf{s}_1, \mathbf{s}_2)\big)$, for an instance of Dilithium with parameters $(n, q, 2\eta, \ell, k)$. This key pair later enables the receiver to issue a Dilithium signature and spend the associated coins. We stress that, for appropriate parameter settings, $\mathbf{t}_1 = \mathbf{A}\mathbf{s}_{1,1} + \mathbf{s}_{2,1}$ is pseudorandom over R_q^k (based on the hardness of the decision Module-LWE problem). It is infeasible to learn $\mathbf{t}_0$ given $(\mathbf{c}, \mathbf{t})$. The security of the resulting signature scheme is then based on those of Kyber and Dilithium.

The derived signature scheme is slightly less efficient than the original Dilithium. That is because here we work with secret keys of infinity norm bounded by 2η instead of η. As a consequence, we would have to slightly increase the scheme parameters, which would lead to a small growth in public key size and signature size.

3.2.3 Lattice-Based Additively Homomorphic Commitments

In the last decade, several additively homomorphic lattice-based commitment schemes with companion zero-knowledge proofs have been proposed. For example,

one of the most practical ones is by Baum et al. [7], and its variant by del Pino et al. [27] in the context of lattice-based e-voting. Let us recall the commitment scheme from [27], which is based on the hardness of Module-SIS and Module-LWE.

Let d be a positive integer, and B_r a positive real bound. Define a commitment scheme with key space $\mathcal{R}_q^{(d+1)\times(2d+1)}$, message space $\mathcal{R}_q$, opening space $\left\{\mathbf{r} \in \mathcal{R}^{2d+1}, \|\mathbf{r}\| \leq B_r\right\}$. The scheme works as follows.

KeyGen The key generation algorithm performs the following steps.

- Choose uniformly random matrices $\mathbf{A}' \in \mathcal{R}_q^{d\times(d+1)}$ and $\mathbf{B} \in \mathcal{R}_q^{1\times(2d+1)}$.
- Let $\mathbf{A} = \left[\mathbf{A}' \mid \mathbf{I}_d\right] \in \mathcal{R}_q^{d\times(2d+1)}$, where $\mathbf{I}_d$ denotes the $d \times d$ identity matrix.
- Output $\mathbf{C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \in \mathcal{R}_q^{(d+1)\times(2d+1)}$.

Commit To commit to message $m \in \mathcal{R}_q$, proceed as follows.

- Sample Gaussian vector $\mathbf{r} \in \mathcal{R}^{2d+1}$, satisfying $\|\mathbf{r}\| \leq B_r$.
- Output $\mathsf{Com}(m, \mathbf{r}) = \mathbf{C} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0} \\ m \end{bmatrix} \in \mathcal{R}_q^{d+1}$.

Open To open $\mathbf{c} \in \mathcal{R}_q^{d+1}$ given $\mathbf{r} \in \mathcal{R}^{2d+1}$, performs the following steps.

- If $\|\mathbf{r}\| \leq B_r$ and there exists $\mathbf{m}' \in \mathcal{R}_q$ such that $\mathbf{c} - \mathbf{C} \cdot \mathbf{r} = \begin{bmatrix} \mathbf{0} \\ m' \end{bmatrix}$, then output m'.
- Else output $\perp$.

According to [27, Theorem 3.1], the scheme above is computationally hiding under the Module-LWE assumption and computationally binding under the Module-SIS

assumption. Moreover, the scheme is additively homomorphic, i.e., the sum of $\mathsf{Com}(m_1, \mathbf{r}_1)$ and $\mathsf{Com}(m_2, \mathbf{r}_2)$ is $\mathsf{Com}(m_1 + m_2, \mathbf{r}_1 + \mathbf{r}_2)$.

However, one cannot sum arbitrary number of commitments: as the sum of the underlying randomness $\mathbf{r}_i$ grows, breaking the binding property becomes easier. To handle the situations where one has to sum a large number of commitments, del Pino et al. suggested a "refreshing" technique: whenever the randomness underlying the sum of commitments grows sufficiently large, one replaces it by a fresh commitment to the same message (with small randomness) and proves in zero-knowledge that the two commitments contain the same committed value.

In terms of efficiency, according to the parameters provided in [27, 7], a commitment has size around 8 KB, and its companion zero-knowledge proof of knowledge of a valid opening has size around 7.5 KB.

3.2.4 Lattice-Based Zero-Knowledge Range Proof

Another cryptographic primitive that we use in our Abelian construction is zero-knowledge proofs. In particular, we choose to use zero-knowledge proofs where an integer, which is committed via a lattice-based commitment scheme, belongs to a given range. For example, the committed value m satisfies $m \in [0, 2^t - 1]$, for some positive integer t. Below are two main zero-knowledge techniques which can achieve this requirement.

- Introduced by Ling et al. [52], which operates in the framework of Stern's protocol [80], this technique is expressive (it is applicable to all known lattice-based commitment schemes). It, however, yields proofs with large sizes because in each protocol execution, it admits some soundness error, and therefore, multiple repetitions are required in order to make the soundness error negligible. For example, [51] has a proof size of over 2 MB for a committed integer in the range of $[0, 2^{64} - 1]$.

- Another technique, initiated by Lyubashevsky [57, 59], employs the Fiat-

Shamir-with-abort approach. This technique is less expressive while it yields more practical sizes in proofs.

Based on Lyubashevsky's technique, del Pino et al. [27] proposed some OR-proofs for their commitment scheme. Their OR-proofs efficiently prove that a committed value (when viewed as an integer) is either 0 or 1, with proof size less than 15KB. This gives us a method to show in the context of privacy-preserving cryptocurrency that a commitment contains either 0 coin or 1 coin.

A method to prove that a committed integer m belongs to the range $[0, 2^t - 1]$ consists of reducing the given task to constructing t OR-proofs for committed values in $\{0, 1\}$. If $m \in [0, 2^t - 1]$, we can write m as:

$$m = m_0 + 2^1 \cdot m_1 + \ldots + 2^{t-1} \cdot m_{t-1},$$

where $m_i \in \{0, 1\}$, for all $i \in [0, t-1]$. Instead of committing to integer m directly (which is impractical in the lattice setting because it requires extremely large modulus q), we commit to its binary representation $(m_0, \ldots, m_{t-1}) \in \{0, 1\}^{64}$. Note that this method was also suggested in [27, Appendix B]. They showed that their proposed commitment scheme can be adapted straightforwardly to handle message space R_q^t, and then an amortized proof that a committed message belongs to $\{0, 1\}^t$ can be done via t parallel atomic proofs. This approach produces a proof size to be less than 1 MB for $t = 64$. We remark that, in practice, one may mainly deal with much smaller number of coins, e.g., $t < 25$. In these cases, we would only require proof size of a few hundred KBs.

3.2.5 Lattice-Based Linkable Ring Signature

A linkable ring signature scheme can be constructed from an ordinary ring signature scheme in a modular manner, by letting the signer output a deterministic function of his secret signing key[22].

[22]This function is called *key image* in CryptoNote. A similar function serves as the serial number in Zerocash [9].

In [50], Libert et al. suggested a blueprint for constructing post-quantum ring signatures based on Merkle hash trees supported by zero-knowledge proofs of a hash chain. They instantiated this framework and proposed the first lattice-based scheme with short secret key and with signature size logarithmic in the cardinality of the underlying ring. Yang et al. [85] then extended the construction to a linkable ring signature. Several other post-quantum ring signature schemes [28, 17] have been proposed based on the blueprint from [50]. However, existing constructions following this approach have not been sufficiently efficient for practical use, due to the current lack of truly practical zero-knowledge proofs for statements related to Merkle hash trees.

Another approach employs Lyubashevsky's techniques [59]. Baum et al. [8] proposed one with security relies on the Module-SIS and Module-LWE assumptions, but the scheme is not scalable. It produces signatures with size linear to the cardinality of the underlying ring. For example, a signature is 82.5KB in size for a ring of 8 signers, while it grows to 1.17MB for a ring with 128 users. Torres et al. [82] proposed a scheme based on the Ring-SIS assumption. Similarly, the scheme is not scalable enough.

Our design can be considered as the mixture of Libert et al.'s blueprint and Lyubashevsky's zero-knowledge techniques.

3.2.6 Lattice-Based Verifiable Encryption

To enable the optional accountability feature, we choose to deploy an efficient lattice-based verifiable encryption scheme. If a transaction is supposed to be tracked by an authority, the user will be constrained from encrypting some crucial information (e.g., the values of consumed coins, the identifying information of sender/receiver) under the authority's public key, and to prove in zero-knowledge that he honestly does so. Should a need arises, the authority can use his secret decryption key to recover the encrypted information. To this end, a lattice-based encryption system supported by zero-knowledge proof of ciphertext well-formedness,

i.e., a verifiable encryption system, is needed.

The first lattice-based verifiable encryption systems were proposed by Ling et al. [53], with respect to Gentry et al.'s LWE-based scheme [38] and the LPR Ring-LWE-based scheme [62]. Their techniques were further employed in subsequent lattice-based constructions [50, 48, 47, 49, 54]. These techniques, however, have not yielded practical systems, since the sizes of the underlying proofs are more than 5 MB.

A practical lattice-based verifiable encryption system was introduced by Lyubashevsky and Neven [61], that produces proof size and ciphertext size as small as 9KB (for 128-bit post-quantum security). Due to technical reasons, in its basic version, their system requires the authority to carry out a considerably large number of decryption attempts - which may be undesirable in practice. Fortunately, as the authors pointed out, the number of decryption attempts can be significantly reduced thanks to an elegant idea that works smoothly in the random oracle model. Their system was subsequently employed by Boschini et al. [20] to construct the most practical lattice-based group signature scheme known to date.

4 Abelian Data Structures and Rules

This section gives an overview to Abelian's data structures and rules on transaction, block, and blockchain.

Abelian is a cryptocurrency system based on transaction-based ledger, where ledger L consists of a series of transactions. Each transaction generates one or multiple transaction-output(TXO)s, and after a transaction is verified and added into ledger L, each TXO of the transaction represents a coin of the cryptocurrency, with the coin-owner and coin-value specified in the TXO. On the input side of a transaction, there are two cases and the transaction is categorized into two types: coinbase transaction and transfer transaction.

A transfer transaction consumes some existing coins (i.e. the TXOs of some previous transactions in L) and transfers the value of the consumed coins to the new coins generated by it (i.e. its TXOs). Note that each TXO can be consumed only once by some transaction, as its value is transferred into other new TXOs once it is consumed, and an attempt to consume a TXO multiple times, referred to as *double spending*, is regarded as an attack that a secure cryptocurrency system should defend against. Also, for a transfer transaction, the total value of the generated TXOs must not exceed that of the consumed TXOs, and an attempt to violate this is referred to as *over spending*. A transfer transaction is valid if and only if it is non-double-spending, non-over-spending, and is well authorized and authenticated by the owner(s) of the consumed TXOs.

A coinbase transaction, in contrast to a transfer transaction, does not consume any existing TXOs. In other words, a coinbase transaction 'creates' coins (its TXOs) and the corresponding value from thin air. The total value of the created coins of a coinbase transaction and the validity criterion are determined by the design of the cryptocurrency.

As a privacy-preserving cryptocurrency, Abelian aims at hiding the source, the destination, and the amount of transactions. Except the payer and payee of a

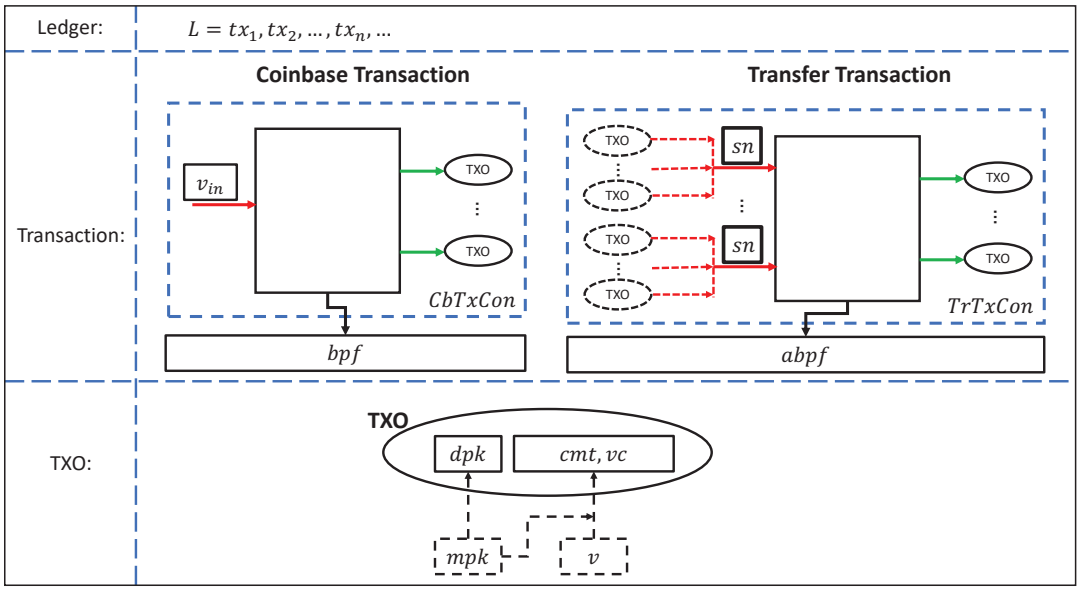

Figure 3: Data Structures of Transactions.

TXO/coin, no one should be able to identify the recipient (i.e. the coin-owner) or the coin-value of the TXO. On the input side, each consumed TXO (in transfer transactions) is hidden in a set of TXOs, so that except the payer (i.e. the owner of that consumed TXO), no one can identify the consumed TXO from the set of TXOs. Fig. 3 shows the outline of the data structures of transactions.

4.1 System Parameters

Below are the system parameters of Abelian:

- Let V be the total supply of coins. As of this writing, V is set to $2^{51} - 1$, which means the total supply is $2^{51} - 1$ Neutrinos, where Neutrino is the minimum unit in Abelian, and 1 Neutrino $= 10^{-7}$ ABEL, where ABEL is the currency unit in Abelian.

- J_{max} specifies the maximum number of generated TXOs in a transfer or coinbase transaction. At this moment, J_{max} is set to 5.

- I_{max} specifies the maximum number of consumed TXOs in a transfer transaction. At this moment $I_{max} = 5$.

Remark: Special attention has to be put on if I_{max} and J_{max} are to be changed, since this will cause the change of the parameters of underlying cryptographic schemes.

4.2 Transaction Data Structures

4.2.1 TXO

Master Key and Derived Public Key. Each user generates at least one master key pair (mpk, msk). The *master public key* mpk is published and enables others to directly transfer to the user. For each transfer to mpk, the payer derives a *fresh* ***derived public key*** dpk from mpk, and uses dpk to specify the owner of TXO. The *master secret key* msk will be used to receive payments sent to mpk and spend them later. The underlying cryptographic scheme of Abelian ensures that each honestly generated derived public key is fresh, which means each derived public key is unique. This is the foundation that Abelian can achieve one-coin-per-address and subsequently hide the coin-owners.

Value and Commitment. Each coin value v is an integer in $[0, V]$. A payer (actually anyone) can generate a commitment cmt for value v.

Transaction Output. TXO is a composite object consisting of derived pubic key and commitment. In particular, each TXO is a (derived public key, commitment, value's ciphertext) tuple $\underline{\mathsf{txo} := (\mathsf{dpk}, \mathsf{cmt}, \mathsf{vc})}$, where dpk acts as the coin-address to specify the owner of the coin represented by this TXO, cmt commits (i.e. specifies but hides) the coin-value, and vc is an encryption of the value so that only the owner (i.e. the owner of the master public key from which the derived public key was derived) can obtain the value, in a non-interactive way. *From now on,*

if a TXO txo*'s derived public key* dpk *is derived from a master public key* mpk*, for simplicity, we say "* txo *belongs to* mpk*", which means that* txo *belongs to the owner of* mpk*.*

Remark: With such a data structure, intuitively, for each coin/TXO, the owner (i.e., the master public key and its owner) is hidden, and the value is hidden.

4.2.2 Coinbase Transaction

A *coinbase-transaction* is a composite object built on TXOs, which does not consume any TXO but generates new TXOs from thin air. In particular, a coinbase-transaction is a (transaction content, balance proof) pair
$\mathsf{cbTx} := (\mathsf{cbTxCon}, \mathsf{bpf})$ with $\mathsf{cbTxCon} := (v_{in}, \{\mathsf{txo}_{out,j}\}_{j=1}^{J})$ where

- $v_{in} \in [0, V]$ is the total value of the new TXOs generated by cbTx, say $\mathsf{txo}_{out,1}, \ldots, \mathsf{txo}_{out,J}$.
- bpf is a proof for the balance, proving that the total value of the generated TXOs is equal to v_{in}.

4.2.3 Transfer Transaction

A *transfer-transaction* is a composite object built on TXOs, which consumes existing TXOs and generates new TXOs, transferring the value in the consumed (old) TXOs to the generated (new) TXOs. In particular, a transfer-transaction is a (transaction content, authorization/authentication and balance proof) pair
$\mathsf{trTx} := (\mathsf{trTxCon}, \mathsf{abpf})$ with $\mathsf{trTxCon} := (\{(\mathsf{inTxoL}_i, \mathsf{sn}_i)\}_{i=1}^{I}, \{\mathsf{txo}_{out,j}\}_{j=1}^{J}, fee)$
where

- $(\mathsf{inTxoL}_1, \mathsf{sn}_1), \ldots, (\mathsf{inTxoL}_I, \mathsf{sn}_I)$ specify the I consumed TXOs. In particular, each inTxoL_i specifies a list (ordered set) of existing TXOs of previous transactions [23] and the corresponding serial number sn_i specifies the consumed

[23] Each existing TXO can be specified by a (transaction hash, output index) pair. Later, when introducing the data structures of block and blockchain, more details will be given.

TXO in inTxoL_i in a privacy-preserving manner. In other words, for each $i \in I$, $(\mathsf{inTxoL}_i, \mathsf{sn}_i)$ implies that "one TXO in inTxoL_i (with serial number sn_i)" is consumed by trTX.

- $\mathsf{txo}_{out,1}, \ldots, \mathsf{txo}_{out,J}$ are the new TXOs generated by trTX.
- $fee \in [0, V]$ is the transaction fee, i.e., the difference between the total value of the generated TXOs and that of the consumed ones [24].
- abpf includes (1) the authorization and authentication proof for trTx, authorizing the spending of the consumed TXOs (specified by $\{\mathsf{sn}_i\}_{i=1}^{n}$) and authenticating the transaction content $\mathsf{trTxCon}$, which says that values in the consumed TXOs are transferred to the generated (new) TXOs $\{\mathsf{txo}_{out,j}\}_{j=1}^{J}$ and the transaction fee fee; and (2) the balance proof for trTx, proving that the total value of the consumed TXOs is equal to that of the generated TXOs and the transaction fee.

Remark: The serial number serves as TXO's secret unique identifier, namely, (1) each TXO has a unique serial number (corresponding to its unique derived public key) and the revealed serial numbers will be used to detect double-spending; and (2) only the owner of a TXO (i.e. the owner of the master public key from which the TXO's derived public key is generated) can compute/reveal the TXO's serial number; and (3) no one except the owner can link a revealed serial number to corresponding TXO, i.e. the revealed serial numbers do not leak the corresponding TXOs.

4.3 Data Structures of Blocks and the Blockchain

In Abelian, transactions are organized into blocks, and the ledger is a (hash) chain of blocks, as shown in Fig. 4.

[24] As the transferred values are hidden, the transaction fee has to be given explicitly.

- Each block consists of an ordered set of transactions, including one coinbase transaction and zero or multiple transfer transactions.

- Besides transactions, each block contains a *block head*, which has four fields (previousblockhash, merkletreeroot, difficulty, nonce):

  - *PreviousBlockHash*: The field *PreviousBlockHash* stores the hash of its previous block, which is computed by hashing the block head of its previous block.

  - *MerkleTreeRootHash*: A Merkle Tree is built on the transactions in the block, and the hash of the Merkle Tree Root is stored in the field *MerkleTreeRootHash*.

  - *Difficulty*: The system parameter *Difficulty* is used to control the block generation rate (e.g. one block per 512 seconds on average) and will be adjusted for every 2000 blocks.

  - *Nonce*: The field *Nonce* does not have any business means, and just makes the hash of the block fall into a valid scope specified by *Difficulty*.

Remark: The blocks do not store the balance proof for any coinbase transaction or the authorization/authentication and balance proof for transfer transaction, and store only the transaction content, say $\mathsf{cbTxCon} := (v_{in}, \{\mathsf{txo}_{out,j}\}_{j=1}^{J})$ for coinbase transaction and $\mathsf{trTxCon} := (\{(\mathsf{inTxoL}_i, \mathsf{sn}_i)\}_{i=1}^{I}, \{\mathsf{txo}_{out,j}\}_{j=1}^{J}, fee)$ for transfer transaction. The Merkle tree is based on the content of the transactions.

As each block's hash value, say H(*PreviousBlockHash, MerkleTreeRootHash, Difficulty, Nonce*) [25] can be and actually is used as the unique identifier of the block, with the $PreviousBlockHash$ field, the blocks are chained together and form a blockchain. For a transaction, only after it is contained in some block and the block is chained in the blockchain, it is accepted as a part of the ledger of the system.

[25] Below, we use H as a secure cryptographic hash function.

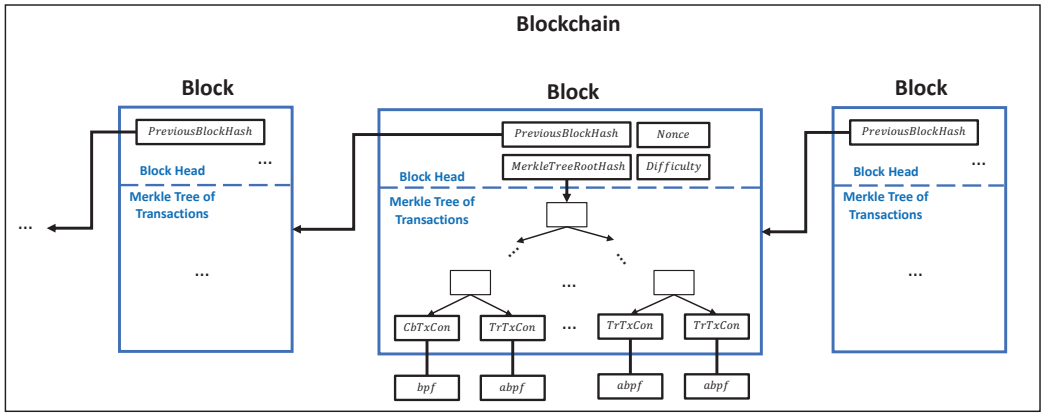

Figure 4: Data Structures of Blocks and Blockchain.

Below, we use $\mathsf{tx} \in B$ to denote that a transaction tx is contained in a block B, and use $B \in L$ to denote that a block B is chained in a blockchain L.

4.4 Rules on Transactions, Blocks, and the Blockchain

We first introduce the roles in the Abelian system and then describe the rules of the system, i.e., how the system runs and what rules data should obey.

4.4.1 Roles

A participant may play as a *user* and/or a *miner*, where *user* is used to describe participants who own coins and create transactions to consume and generate coins, while *miner* is used to describe participants who build and propose valid blocks to store transactions into the ledger (i.e. form the blockchain).

User. Each user can generate one or multiple master key pairs. For each master key pair, say $(\mathsf{mpk}, \mathsf{msk} := (\mathsf{msvk}, \mathsf{mssk}))$, the owner can publish the master public key mpk and his identity, so that anyone can transfer coins to him by using his master public key. On the other side, for each coin (i.e. a TXO) in the ledger, say $\mathsf{txo} := (\mathsf{dpk}, \mathsf{cmt}, \mathsf{vc})$, a user needs to use a part of his master secret key, say msvk,

referred to as *master secret view key*, to check whether the coin belongs to him, and if it does, the user will recover the coin-value (also by using msvk) and add the coin to his wallet. And for each coin in his wallet, the user can use his *master secret key*, $\mathsf{msk} := (\mathsf{msvk}, \mathsf{mssk})$ where mssk is referred to as *master secret spend key*, to create a transaction consuming the coin. Neither the coin/TXO itself nor the transaction consuming the coin leaks the corresponding master public key (i.e. the coin-owner) or the coin-value.

Miner. The generation of block in Abelian is similar to that of Bitcoin. More specifically, a miner collects transfer transactions issued by users, mine to find valid blocks, propose valid blocks, verify received blocks, and maintain its local blockchain. As of this writing, Abelian uses Bitcoin-like PoW-based (Proof-of-Work-based) consensus protocol.

4.4.2 Rules

Genesis Block. There is an initial block, referred to as the genesis block, being hardcoded in Abelian source code so that all participants share the same genesis block through running the source code.

- Genesis block's *PreviousBlockHash* is set to all-zero, implying that it is the first block of the blockchain and does not have a previous block.
- Genesis block contains only one coinbase transaction, $\mathsf{CbTx} := (\mathsf{CbTxCon}, \mathsf{bpf})$ with $\mathsf{CbTxCon} = (v_{in}, \mathsf{txo})$, where v_{in} is set according to the token release schedule and txo and bpf are generated using a standalone subroutine outside the system source code. Genesis block's *MerkleTreeRootHash* is computed based on the transaction list, which contains only this coinbase transaction.
- Genesis block's *Difficulty* is set according to the expected/evaluated initial computation power of the system, such that one block is released per 512 seconds on average.

- Genesis block's *Nonce* is computed using a standalone subroutine outside the system source code, so that the hash of the genesis block falls in the target scope specified by *Difficulty*.

Note that while the coinbase transaction and the nonce are generated/computed by standalone subroutines outside the system source code, any participant can run the code to verify the validity of the block, including the validity of the coinbase transaction (the coin-value of txo equals v_{in}) and that the hash of the block falls in the target scope. Abelian source codes are public and open source, will always be the case.

Rule 1: The Block Height of the genesis block is 0. Each block has a block height one greater than its previous block.

For simplicity, when we say a blockchain has height h, we mean the height of the latest block in the blockchain is h.

4.4.3 Maturity of Coins in Coinbase Transaction

At the very beginning, the coins in the system are only the coins generated by the coinbase transaction in the genesis block, and the owner of these coins may want to spend them. However, the coins cannot be spent until they become mature.

Rule 2: For a coin generated by the coinbase transaction in a block with height h, say B_h, it is regarded as being mature with respect to a blockchain L if and only if $B_h \in L$ and L has height at least $h +$ $\mathsf{maturity}_{sys}$, where $\mathsf{maturity}_{sys}$ is a system parameter. On the other side, for a blockchain L, it would not accept any transaction consuming immature coins.

Currently, $\mathsf{maturity}_{sys}$ is set to 99. This means before the block height reaches 99, there is not transfer transaction in the system. This also means that for the first 100 blocks, each block can have one coinbase transaction only.

4.4.4 Block Generation: Mining

Initially, all miners in the system set up their local ledgers with one block, the genesis block. From the view of a miner, the miner mines based on its local blockchain and its local *memory pool*, which stores received transfer transactions that have not been included in any block in its local blockchain. Here we suppose that the transfer transactions in the memory pool are valid, while referring the validity check of transfer transactions to Sec. 4.4.8. A miner performs the mining as follows:

1. Initialize a new block B.
2. Set $B.PreviousBlockHash$ to be the hash of the latest block of its local blockchain.
3. Set $B.Difficulty$ according to difficulty setting rules.
4. Choose transfer transactions (from memory pool) that will be contained in B, and compute the total transaction fee of the selected transactions. Note that the set of selected transfer transactions could be empty, and is determined by the miner's local policy.
5. Generate coinbase transaction:
   - Compute v_{in}, by summing up the total transaction fee and the current *block reward*.
   - Set the (master public key, coin-value) pair(s) that will be used to generate TXO(s) for the coinbase transaction.
   - Generate coinbase transaction CbTx by calling the Coinbase Transaction Generation subroutine of the underlying cryptographic scheme, on input v_{in} and the (master public key, coin-value) pair(s).
6. Build Merkle tree on the generated coinbase transaction and selected transfer transactions, and set the value for $B.MerkleTreeRootHash$.

7. Find *Nonce* such that
   H(*B.PreviousBlockHash, B.MerkleTreeRootHash, B.Difficulty, Nonce*)
   falls into the target scope determined by *B.Difficulty*, then set *B.Nonce* and broadcast the block B to the Abelian network.

4.4.5 The Validity Rule of Block

When a block is received from other miners, a miner needs to check the validity the block.

Rule 3: A block is valid if and only if

- ***The block's*** $PreviousBlockHash$ ***is the hash of some block in local memory or local blockchain, so that the height of the received block is determined.***
- ***The block's*** $Difficulty$ ***satisfies the difficulty rule.***
- ***The hash value of the block falls in the scope specified by the difficulty.***
- ***All transfer transactions in the block are valid (Sec. 4.4.8).***
- ***The coinbase transaction is the first transaction in the block's transaction list.***
- ***The*** v_{in} ***of the coinbase transaction is the sum of the total transaction fee and the block reward.***
- ***The coinbase transaction's*** bpf ***is valid.***
- ***The block's*** $MerkleTreeRootHash$ ***is valid with respect to the block's transaction list.***

4.4.6 The Rule of TXO Ring

As mentioned, for a transfer transaction in Abelian, each consumed coin/TXO is hidden in a set of TXOs. To resist some privacy-analyzing attacks, as well as to improve efficiency, Abelian adopts the *fixed-ring* policy, where the TXO rings are formed by system protocol/rules.

Before describing the rule, we would like to highlight that for each TXO in Abelian, it is uniquely identified by a (TxHash, index) pair, where TxHash is the hash of the transaction which generates this TXO and index is the position of this TXO among all the TXOs generated by the transaction.

Rule 4: Coins/TXOs are divided into rings as below:

- ***Starting from the genesis block, each three successive blocks form a block group. For any*** $i = 0, 1, 2, \ldots$ ***, blocks with height*** $3i, 3i+1, 3i+2$ ***form a block group.***

- ***For each block group, all*** TXO***s generated by the coinbase transactions form a*** TXO ***group, referred to as Coinbase-TXO group, and all the*** TXO***s generated by the transfer transactions form a*** TXO ***group, referred to as Transfer-TXO group. Suppose the hash of the three blocks are*** $hash_0$***,*** $hash_1$***, and*** $hash_2$***, respectively,***

  - ***for each*** TXO***, a hash value*** $TxoHash = H(hash_0, hash_1, hash_2, BlockHash, TxHash, index)$ ***is computed, where*** $BlockHash$ ***is the hash of the block that contains the transaction generating this*** TXO***,*** $TxHash$ ***is the hash of the transaction generating this*** TXO***, and*** $index$ ***is the position of this*** TXO ***among all the*** TXO***s generated by the transaction.***

  - ***All the*** TXO***s in Coinbase-TXO group (resp. Transfer-TXO group) are ordered by their*** $TxoHash$***, they are divided into (ordered) lists with size*** $\mathsf{RingSize}_{sys}$***, and each list is referred***

to as a ring. If the last list has size smaller than $\mathsf{RingSize}_{sys}$ ***and if there is a ring before it, the*** TXO***s in the last two lists will be put together and divided into two rings with almost the same size. Accordingly, the ring is referred to as Coinbase-TXO Ring (resp. Transfer-TXO Ring). For example, we may set*** $\mathsf{RingSize}_{sys} = 7$***.***

- ***For each TXO ring, the Ring Height is defined as the largest height of the corresponding block group. Suppose the height of the three blocks is*** $3i, 3i+1, 3i+2$***, all the rings generated from this block group have ring height*** $3i + 2$***.***

The rule above is described in a static manner. In fact, in the system, starting from the genesis block, when a new block with height h is appended into the blockchain, if $h\%3 = 2$, then the previous two successive blocks and this new block will form a new block group, and a set of new TXO rings will be generated.

On the other side, when a user wants to spend a coin/TXO, he actually spends a member TXO of a ring. The following rule defines the maturity of TXO ring.

Rule 5: For a Coinbase-TXO Ring with ring height $3i + 2$***, it becomes mature with respect to a blockchain when the blockchain contains a block with height*** $3i + 2 + \mathsf{maturity}_{sys}$***. For a Transfer-TXO Ring, it becomes mature immediately.***

From the view of a user, he can spend a coin successfully only after the ring containing the coin becomes mature. For example, suppose a user owns a coin generated by a transfer transaction contained in a block with height $3i$ for some i, then only after blocks with height $3i+1$ and $3i+2$ are appended into the blockchain and the rings of this block group are generated, the user can spend this coin.

4.4.7 Transfer Transaction Generation

From the view of a user, his wallet needs to fetch complete blocks of the blockchain to its local memory.Suppose that a TXO/coin txo belongs to a wallet, the wallet needs to fetch the other two complete blocks of the block group, so that the wallet can run the above TXO ring generation procedure and get the ring containing txo. When a user wants to spend some of his coins by a transfer transaction,

- on the input side, for each coin to be consumed, the user/wallet needs to provide the corresponding master secret key, the corresponding TXO ring and the index of the coin in the ring, and the coin-value;
- on the output side, for each target recipient (including the change for the user himself), the wallet needs to specify a (master public key, coin-value) pair, which will be used to generate the corresponding TXO,
- and the wallet needs to specify the transaction fee, so that the total value of the consumed coins equals the total value of the generated coins and the transaction fee.

The wallet will then call the Transfer Transaction Generation subroutine of the underlying cryptographic scheme and obtain a transfer transaction $\mathsf{TrTx} = \big(\mathsf{TrTxCon}, \mathsf{abpf}\big)$ with $\mathsf{TrTxCon} = \big(\{\mathsf{txoRing}_i, \mathsf{sn}_i\}_{i=1}^{I}, \{\mathsf{txo}_j\}_{j=1}^{J}, fee\big)$, and broadcast TrTx to the Abelian network.

The transaction content TrTxCon implies that this transaction consumes I coins and generates J new coins, and the transaction fee is *fee*. As to the consumed coins, TrTxCon implies that for each $i \in [1, I]$, a coin in ring $\mathsf{txoRing}_i$ is consumed and the consumed coin's serial number is sn_i.

The underlying cryptographic scheme ensures that, (1) for a TXO $\mathsf{txo} := (\mathsf{dpk}, \mathsf{cmt}, \mathsf{vc})$, only the owner with the corresponding master secret key can generate the corresponding serial number sn and (2) each TXO (with unique dpk) has a unique serial number. This implies that the public cannot identify the actual

consumed TXO from the TXO ring, and the coin-owners cannot double-spend their coins.

Due to the hiding of the consumed coins, a user's wallet may run the Serial-Number Generation subroutine of the underlying cryptographic scheme to generate the serial numbers for his coins in the wallet, so that the wallet can exactly know which coins have been spent and which are still spendable. This is to deal with the situation that a user runs two wallets with the same master key pair on different devices.

Note that in a transfer transaction, each txoRing is a set of pointers referencing to a set of existing TXOs, rather than a set of TXOs as described in Sec. 4.3. More specifically, each txoRing contains 3 hash values for the 3 blocks in the corresponding block group and there are at most $\mathsf{RingSize}_{sys}$ (TxHash, index) pairs.

4.4.8 The Validity Rule of Transfer Transaction

As described, a user can create a transfer transaction and broadcast the transaction to Abelian network. However, only after a transaction is included in some block and the block is appended into a blockchain, the transaction would be accepted as a part of the cryptocurrency's ledger. While we have described how a miner selects transfer transactions from his local memory pool and mines new block in Sec. 4.4.4, below we describe how a miner checks the validity of a received transfer transaction and put it into its local memory pool.

Before describing the validity rule of transfer transaction, we need to first describe how a miner maintains Unspent TXO Ring set, which is similar to the UTXO set in bitcoin, storing the (possible) spendable TXOs. In particular, the UTXORing Set S_{utxor} is defined such that each element in the set will be a (TXO Ring, SerialNumber Set) pair $(\mathsf{txoRing}, \mathsf{snSet})$, and will be maintained with the rule below.

Rule 6: The UTXORing set is maintained as follows:

1. ***Starting from the genesis block, the UTXORing Set*** S_{utxor} ***is empty.***

2. ***When a new block with height*** h ***is appended to the blockchain,***

   (a) ***for each transfer transaction (content) in this new block, say*** $\mathsf{TrTxCon} = \big(\{\mathsf{txoRing}_i, \mathsf{sn}_i\}_{i=1}^{I}, \{\mathsf{txo}_j\}_{j=1}^{J}, fee\big)$***,***

      i. ***for each*** $\mathsf{txoRing}_i$***, find the corresponding pair in*** S_{utxor}***, say*** $(\mathsf{txoRing}, \mathsf{snSet})$***, then add*** sn_i ***into*** snSet [26]***. If*** $|\mathsf{snSet}| = |\mathsf{txoRing}|$***, remove this pair from*** S_{utxor}***, since the equation implies that all*** TXO***s in*** txoRing ***have been spent.***

   (b) ***If*** $h\%3 = 2$***,***

      - ***take the previous two blocks and this new block to form a block group, and generate a set of new TXO Rings as described in Rule 4 in Sec. 4.4.6;***
      - ***for each newly generated TXO Ring,*** txoRing***, add*** $(\mathsf{txoRing}, \emptyset)$ ***into*** S_{utxor}***, where the empty SerialNumber set implies that none of the TXOs in*** txoRing ***has been spent.***

We now describe the validity rule of transfer transactions.

Rule 7: From the view of a miner, a transfer transaction, say $\mathsf{TrTx} = (\mathsf{TrTxCon}, \mathsf{abpf})$ ***with*** $\mathsf{TrTxCon} = \big(\{\mathsf{txoRing}_i, \mathsf{sn}_i\}_{i=1}^{I}, \{\mathsf{txo}_j\}_{j=1}^{J}, fee\big)$***, is valid with respect to a blockchain, say*** L***, if and only if***

- abpf ***is valid, which implies that (1) the owners of the consumed coins (specified by*** $\{\mathsf{txoRing}_i, \mathsf{sn}_i\}_{i=1}^{I}$***) have authorized and authenticated this transaction, and (2) the total value of the consumed***

[26] As this new block is being appended to the current blockchain, it is assumed to be valid with respect to the current blockchain, and all transfer transactions in this new block are also valid, namely, as shown later, there is a pair $(\mathsf{txoRing}, \mathsf{snSet}) \in S_{utxor}$ such that $\mathsf{txoRing} = \mathsf{txoRing}_i$ AND $|\mathsf{snSet}| < |\mathsf{txoRing}|$ AND $\mathsf{sn}_i \notin \mathsf{snSet}$, where $|\cdot|$ denotes the size of the set.

coins equals to the total value of the generated coins and transaction fee.

- ***For any*** $j_1 \neq j_2 \in [1, J]$***, it holds that*** $\mathsf{txo}_{j_1}.\mathsf{dpk} \neq \mathsf{txo}_{j_2}.\mathsf{dpk}$***, i.e., each generated TXO has a different derived public key.***

- ***For any*** $i_1 \neq i_2 \in [1, I]$***, it holds that*** $\mathsf{sn}_{i_1} \neq \mathsf{sn}_{i_2}$***, i.e., each consumed TXO has a different serial number. Note that repeated serial numbers imply double-spending.***

- ***For each*** $i \in [1, I]$***, there is a pair*** $(\mathsf{txoRing}, \mathsf{snSet}) \in S_{utxor}$ ***such that*** $\mathsf{txoRing}_i = \mathsf{txoRing}$ ***and*** $\mathsf{sn}_i \notin \mathsf{snSet}$***, and if*** $\mathsf{txoRing}_i$ ***is a coinbase-TXO-ring, it should be mature. Note that*** $\mathsf{txoRing}_i \notin S_{utxor}$ ***implies that*** $\mathsf{txoRing}_i$ ***does not exist with respect to*** L ***or it has been removed because all its coins have been spent. By*** $\mathsf{sn}_i \notin \mathsf{txoRing}$***, it implies that the corresponding coins has been spent by some existing transaction in*** L***.***

In Abelian, for honestly generated transactions, the derived public key of each TXO will be globally unique. Ideally, the miner should check this point to detect maliciously generated TXOs. However, it will be very inefficient to check the global uniqueness of derived public keys, especially when the blockchain becomes large. From the view of a miner, it might only care that the transaction is well authorized and authenticated and the balance holds, here the miner only checks whether the derived public keys are unique inside individual transactions, and leave more checks to the user, say the recipient of each TXO.

Hence, when a user detects that there is a new coin/TXO belonging to him, he needs to check whether this new coin's derived public key has been in his wallet. If the derived public key is a repeated one, the user should refuse to acknowledge the reception of the coin and then refuse to deliver goods/service to the payer. In particular, if the two coins belong to the same ring, the recipient could spend at

most of them, and if the two coins belong to different rings, the recipient could spend both of them, but his privacy will be weakened. As each user can easily detect the coins which are intended to him but with repeated derived key and then refuse to acknowledge the reception of the coin, no one will attempt to launch such an attack. On the other side, if an attacker attempts to generate two TXOs with the same derived public key but intended to two different users (i.e. master public keys), the underlying cryptographic scheme will ensures that at most one of the TXOs will be accepted as a valid payment by the target users.

Similarly, a miner checks whether repeated serial number appears in individual transactions and in individual rings, guaranteeing that there is no double-spending.

5 Tokenomics

The total supply of Abelian is 225.18 million ABELs (exact amount: 225,179,981 ABELs), or $2^{51} - 1$ Neutrinos (exact amount: 2,251,799,813,685,247 Neutrinos), where Neutrino is the smallest cryptocurrency unit in Abelian and 1 ABEL = 10^7 Neutrinos.

For every 256 seconds (about 4.27 minutes), one block will be generated and the mining difficulty will be reviewed and adjusted for every 4,000 blocks. In each *era*, there will be 400,000 blocks generated. In other words, each era will last for about 3.25 years. In the first era, the block reward is 256 ABELs. Halving will happen after each era, namely, the number of ABELs to be rewarded per block will be halved in the next era. In the final era, there will be 0.5 ABEL rewarded per block.

At the Genesis Block, there will be 20.58 million ABELs pre-mined (exact amount: 205,799,813,685,247 Neutrinos), and that count towards 9.14% of the total ABEL supply in Abelian. These ABELs will be exclusively used for supporting the decentralization of the network, and for promoting as well as building up a diversified and active Abelian community.

5.1 Token Release Schedule

- **Era 1**: 256 ABELs per block, (20.48+102.4) million ABELs will be mined in the first 3.25 years (55%)
- **Era 2**: 128 ABELs per block, (20.48+153.6) million ABELs will be mined in the first 6.5 years (77%)
- **Era 3**: 64 ABELs per block, (20.48+179.2) million ABELs will be mined in the first 9.75 years (89%)
- **Era 4**: 32 ABELs per block, (20.48+192) million ABELs will be mined in the first 13 years (94%)

- **Era 5**: 16 ABELs per block
- **Era 6**: 8 ABELs per block
- **Era 7**: 4 ABELs per block
- **Era 8**: 2 ABELs per block
- **Era 9**: 1 ABEL per block
- **Era 10**: 0.5 ABEL per block (final era)

It will take 32.5 years to complete all the ABEL mining, and the total supply is calculated as

$$\begin{aligned} & 20.58\ \texttt{million} + (256 + 128 + \ldots + 1 + 0.5) \times 400{,}000 \\ = {} & 20.58\ \texttt{million} + (2^{10} - 1) \times 200{,}000 \\ = {} & 225.18\ \texttt{million} \end{aligned}$$

Below is a time-related summary:

- One block is released for every 256 seconds on average.
- The mining difficulty is adjusted for every 11.8 days (approx).
- The block reward is halved for every 3.25 years (approx).
- All supply by mining will be released in 32.5 years.
- At the end of the first era, there will be 122,979,981.3685247 ABELs (including all the pre-issued at the genesis block and mined in the first era) available.

5.2 Indicative Roadmap

Abelian is an open community project which aims at making all the cryptographic algorithms peer reviewed academically and all the source code open for collaborative development by the community and for the community. The future development of Abelian will completely be steered by the community and we believe this will be the most transparent and sustainable way to promote the adoption of post-quantum cryptographic techniques into the Blockchain industry.

Below we outline some tentative areas that our community may consider to develop in the near future. As can be considered as version 1.0, Abelian will start with SHA-256 based cryptographic hash functions with the mining difficulty being reviewed and adjusted for every 4,000 blocks, and set the block size to 8MB. The block creation frequency will be 256 seconds and the initial block reward will be 256 ABELs. The block generation and consensus mechanisms are similar to that of Bitcoin. The theoretical throughput will be at 10 TPS (transactions per second). After the full launch of Abelian version 1.0, we may consider upgrading the security to Abelian version 2.0 by replacing the cryptographic hash functions to SHA-512 based with a slight tradeoff on the throughput that it will be down to 8 TPS. Right after having a stabilized network achieved, we may consider doubling the block size to 16MB for achieving 16 TPS theoretical throughput. We do strive to make the upgrades from one version to the next gracefully, and can upgrade individual nodes independently, despite some hard forking may be needed during upgrades.

We consider this as just the beginning of the post-quantum Blockchain initiative. On Abelian specifically, as of this writing, we are merely achieving the very first step of our dream only. We continue devising provably secure and practical cryptographic methods for achieving privacy with accountability, and devoting ourselves to achieve much higher efficiency, and even inter-operability with other quantum-resistant Blockchain systems.

As the ultimate goal of Abelian is to build an open community, we continue

inviting great researchers, developers, investors, all crypto stakeholders, who are passionate about post-quantum security or privacy coins, to participate and contribute, and we solicit all supporters' help on building up the Abelian ecosystem together.

In this open community, we welcome contributions in all form. To name a few, we are hoping that talented teams will be able to help build the following technologies for Abelian:

- build much faster payment or token transfer mechanisms;
- create side-chains or layer 2 post-quantum technologies to enable smart contracts and improve scalability as well as decentralization;
- improve the consensus protocol efficiency and effectiveness including the introduction of proof-of-stake related mechanisms;
- propose more efficient or secure cryptographic implementations for boosting the full privacy with provable security or shortening signature and key sizes; and many more.

6 Conclusion

With the dream of creating a quantum-resistant cryptocurrency platform and safeguarding the anonymity merit of cryptocurrency as first envisioned in the seminal paper of Bitcoin, we set the advancement of a cryptocurrency provisioning quantum-resistant and full privacy with optional accountability as the mission of Abelian. In this manuscript, we proposed our technical approaches, and also rigorously defined the cryptographic primitives and their security models for capturing practical attacking scenarios. We believe that these will lay out a strong foundation for us to build up the Abelian Blockchain platform in a provably secure fashion.

We defined three levels of privacy spanning from the level as of Bitcoin (i.e. Basic Privacy) to the Full Privacy level with the coin flows over addresses remain unlinkability and untraceability, and at the same, ensure that transaction amount is hidden. We also defined the level of Full Privacy with Accountability for accommodating applications where regulatory bodies or organizations may require the basic privacy level while keeping the full privacy level to all other participants.

We believe that setting up a strong development and application community is utmost important towards the sustainability of quantum-resistant Blockchain initiative. Based on the rigorous definitions and models we formulated, we welcome contributions to further develop Abelian and even adopt technologies from Abelian to their own applications. Abelian will always be open source, and the future of Abelian will be steered by its community and supporters. We would like to give the cryptocurrency market a new choice and a new platform, which empowers other innovators to build their next disruptive technologies and businesses using the Abelian quantum-resistant Blockchain platform and its forthcoming ecosystem.

References

[1] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108. ACM, 1996.

[2] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[4] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[5] M. H. Au, S. S. M. Chow, W. Susilo, and P. P. Tsang. Short linkable ring signatures revisited. In A. S. Atzeni and A. Lioy, editors, *Public Key Infrastructure, Third European PKI Workshop: Theory and Practice, EuroPKI 2006, Turin, Italy, June 19-20, 2006, Proceedings*, volume 4043 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2006.

[6] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 21–32. ACM, 1991.

[7] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert. More efficient commitments from structured lattice assumptions. *IACR Cryptology ePrint Archive*, 2016:997, 2016. To appear at SCN 2018.

[8] C. Baum, H. Lin, and S. Oechsner. Towards practical lattice-based one-time linkable ring signatures. *IACR Cryptology ePrint Archive*, 2018:107, 2018.

[9] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014.

[10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[11] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive arguments for a von neumann architecture. *IACR Cryptology ePrint Archive*, 2013:879, 2013.

[12] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2006.

[13] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. *J. Cryptology*, 22(1):114–138, 2009.

[14] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back

again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[15] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography*, pages 315–333. Springer, 2013.

[16] Bitcoinwiki. `https://en.bitcoin.it/wiki/Wei_Dai`. Accessed 27 September 2017.

[17] D. Boneh, S. Eskandarian, and B. Fisch. Post-quantum group signatures from symmetric primitives. *IACR Cryptology ePrint Archive*, 2018:261, 2018.

[18] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018. ACM, 2016.

[19] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - kyber: a cca-secure module-lattice-based KEM. *IACR Cryptology ePrint Archive*, 2017:634, 2017.

[20] C. Boschini, J. Camenisch, and G. Neven. Floppy-sized group signatures from lattices. *IACR Cryptology ePrint Archive*, 2018:453, 2018. Accepted to ACNS 2018.

[21] N. Chandran, J. Groth, and A. Sahai. Ring signatures of sub-linear size without random oracles. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 2007.

[22] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017.

[23] M. Chase, C. Ganesh, and P. Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2016.

[24] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.*, pages 199–203. Plenum Press, New York, 1982.

[25] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[26] W. Dai. b-money. `http://www.weidai.com/bmoney.txt`. Accessed 25 September 2017.

[27] R. del Pino, V. Lyubashevsky, G. Neven, and G. Seiler. Practical quantum-safe voting from lattices. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Com-*

puter and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 1565–1581. ACM, 2017.

[28] D. Derler, S. Ramacher, and D. Slamanig. Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 419–440. Springer, 2018.

[29] Y. Dodis. Lectures for introduction to cryptography.

[30] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 609–626. Springer, 2004.

[31] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[32] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, 1996.

[33] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 308–317. IEEE Computer Society, 1990.

[34] H. Finney. Rpow – reusable proofs of work, August 2004. `http://nakamotoinstitute.org/finney/rpow/index.html`. Accessed 28 September 2017.

[35] E. Fujisaki. Sub-linear size traceable ring signatures without random oracles. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 393–415. Springer, 2011.

[36] E. Fujisaki and K. Suzuki. Traceable ring signature. In T. Okamoto and X. Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2007.

[37] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[38] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In C. Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206. ACM, 2008.

[39] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108. ACM, 2011.

[40] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In T. Holz and S. Savage, editors, *25th USENIX Security*

Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pages 1069–1083. USENIX Association, 2016.

[41] J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 253–280. Springer, 2015.

[42] M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 102–111, 1993.

[43] J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. *IACR Cryptology ePrint Archive*, 2018:475, 2018.

[44] A. Kawachi, K. Tanaka, and K. Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2008.

[45] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.

[46] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.

[47] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 373–403, 2016.

[48] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Zero-knowledge arguments for matrix-vector relations and lattice-based group encryption. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 101–131, 2016.

[49] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Adaptive oblivious transfer with access control from lattice assumptions. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 533–563. Springer, 2017.

[50] B. Libert, S. Ling, K. Nguyen, and H. Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 1–31. Springer, 2016.

[51] B. Libert, S. Ling, K. Nguyen, and H. Wang. Lattice-based zero-knowledge arguments for integer relations. In *CRYPTO 2018*, 2018. Accepted to CRYPTO 2018.

[52] S. Ling, K. Nguyen, D. Stehlé, and H. Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, volume 7778 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2013.

[53] S. Ling, K. Nguyen, and H. Wang. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In J. Katz, editor, *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, volume 9020 of *Lecture Notes in Computer Science*, pages 427–449. Springer, 2015.

[54] S. Ling, K. Nguyen, H. Wang, and Y. Xu. Constant-size group signatures from lattices. In M. Abdalla and R. Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 58–88. Springer, 2018.

[55] J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, volume 3108 of *Lecture Notes in Computer Science*, pages 325–335. Springer, 2004.

[56] J. K. Liu and D. S. Wong. Linkable ring signatures: Security models and new schemes. In O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, editors, *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part II*, volume 3481 of *Lecture Notes in Computer Science*, pages 614–623. Springer, 2005.

[57] V. Lyubashevsky. Lattice-based identification schemes secure under active attacks. In R. Cramer, editor, *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2008.

[58] V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, 2009.

[59] V. Lyubashevsky. Lattice signatures without trapdoors. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, 2012.

[60] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*,

volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2006.

[61] V. Lyubashevsky and G. Neven. One-shot verifiable encryption from lattices. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 293–323, 2017.

[62] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[63] T. C. May. The crypto anarchist manifesto. `https://www.activism.net/cypherpunk/crypto-anarchy.html`. Accessed 25 September 2017.

[64] S. Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[65] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.

[66] NIST. Post-quantum cryptography standardization. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`. Accessed 21 March 2018.

[67] S. Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. `https://eprint.iacr.org/2015/1098`.

[68] S. Noether and A. Mackenzie. Ring confidential transactions. Ledger, vol. 1, pp. 1-18, 2016.

[69] M. Ober, S. Katzenbeisser, and K. Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future Internet*, 5(2):237–250, 2013.

[70] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[71] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166. Springer, 2006.

[72] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[73] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. *CoRR*, abs/1107.4524, 2011.

[74] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.

[75] D. Ron and A. Shamir. Quantitative analysis of the full bitcoin transaction graph. In A. Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 6–24. Springer, 2013.

[76] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[77] Satoshi Nakamoto Institute. `http://nakamotoinstitute.org/finney`. Accessed 27 September 2017.

[78] H. Shacham and B. Waters. Efficient ring signatures without random oracles. In T. Okamoto and X. Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2007.

[79] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[80] J. Stern. A new identification scheme based on syndrome decoding. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993.

[81] N. Szabo. Bit gold, 29 December 2005. `http://nakamotoinstitute.org/bit-gold/`. Accessed 25 September 2017.

[82] W. A. A. Torres, R. Steinfeld, A. Sakzad, J. K. Liu, V. Kuchta, N. Bhattacharjee, M. H. Au, and J. Cheng. Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1.0). *IACR Cryptology ePrint Archive*, 2018:379, 2018. Accepted to ACISP 2018.

[83] P. P. Tsang and V. K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In R. H. Deng, F. Bao, H. Pang, and J. Zhou, editors, *Information Security Practice and Experience, First International Conference, ISPEC 2005, Singapore, April 11-14, 2005, Proceedings*, volume 3439 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2005.

[84] N. van Saberhagen. Cryptonote v 2.0, 2013. `https://cryptonote.org/whitepaper.pdf`.

[85] R. Yang, M. H. Au, J. Lai, Q. Xu, and Z. Yu. Lattice-based techniques for accountable anonymity: Composition of abstract stern's protocols and weak PRF with efficient protocols from LWR. *IACR Cryptology ePrint Archive*, 2017:781, 2017.

A Cryptographic Primitives

In this part, we give a brief review of cryptographic tools that Abelian Coin uses, including Hash, Signature scheme, Linkable Ring Signature scheme, Commitment scheme, Zero Knowledge Proof, etc.

A.1 Hash

A cryptographic hash function (with output length l) is a probabilistic polynomial-time (PPT) algorithm $H : \{0,1\}^* \to \{0,1\}^l$ that is collusion resistant, i.e. no efficient (i.e. PPT) adversary can find $x, x' \in \{0,1\}^*$ such that $x \neq x'$ and $H(x) = H(x')$ except with negligible probability.

A.2 Digit Signature

Definition 7 (Signature) *A (digital)* **Signature** *scheme consists of three PPT algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *such that:*

- $\mathsf{Gen}(1^n) \to (pk, sk)$. *The key-generation algorithm* Gen *takes as input a security parameter* 1^n *and outputs a pair of keys* (pk, sk). *These are called the public key and the private key, respectively.*
- $\mathsf{Sign}(sk, M) \to \sigma$. *The signing algorithm* Sign *takes as input a private key* sk *and a message* M *from some message space (that may depend on* pk*). It outputs a signature* σ.
- $\mathsf{Vrfy}(pk, M, \sigma) \to 1/0$. *The deterministic verification algorithm* Vrfy *takes as input a public key* pk*, a message* M*, and a signature* σ*. It outputs a bit* b*, with* $b = 1$ *meaning valid and* $b = 0$ *meaning invalid.*

Correctness. It is required that except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that $\mathsf{Vrfy}(pk, M, \mathsf{Sign}(sk, M)) = 1$ for every (legal) message M.

Security. For a fixed public key pk generated by a signer S, a *forgery* is a message M along with a valid signature σ, where M was not previously signed by S. Security of a Signature scheme means that an adversary should be unable to output a forgery even if it obtains signatures on many other messages of its choice. Formally, the security of Signature scheme is defined by the following Def. 8.

Definition 8 *Given a Signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk)*.*
2. *Adversary* $\mathcal{A}$ *is given* pk *and access to an oracle* $\mathsf{Sign}(sk, \cdot)$*.*
3. *Adversary* $\mathcal{A}$ *outputs a (message, signature) pair* (M, σ)*. Let* Q *denote the set of all queries that* $\mathcal{A}$ *asked its oracle.* $\mathcal{A}$ *succeeds if and only if (1)* $\mathsf{Vrfy}(pk, M, \sigma) = 1$ *and (2)* $M \notin Q$*.*

A Signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* **existentially unforgeable under an adaptive chosen-message attack***, or just* **secure***, for all PPT adversaries* $\mathcal{A}$*, the success probability of* $\mathcal{A}$ *in the above game is negligible.*

Roughly speaking, a secure Signature scheme can ensure that (1) only the signer (who has a private key) can generate a valid signature and (2) anyone (since the corresponding public key is published) can verify if a signature with respect to a message is valid. Thus, a Signature scheme can be used to authenticate a message. In particular, when a sender S, the owner of a key pair (pk, sk), wants to authenticate a message M, it computes a signature $\sigma \leftarrow \mathsf{Sign}(sk, M)$ and sends (M, σ). Upon receipt of (M, σ), a receiver who knows the public key pk can verify the authenticity of m by checking whether $\mathsf{Vrfy}(pk, M, \sigma) = 1$. This establishes both that S sent M, and also that M was not modified in transit. Note that this implies a very useful feature of signature – 'non-repudiation', namely, if a (message, signature) pair (M, σ) is verified to be valid by a public key pk, the owner of pk cannot deny that he sent the message M.

With these features, say authenticity and non-repudiation, Signature becomes one of the foundations for cryptocurrency. In cryptocurrency, coins are "owned" by some public key pk, and when the "real owner" of the coins wants to spend the coins through a transaction, he needs to sign a message (i.e. the transaction) using the corresponding secret key sk. When a transaction and the corresponding signature are verified to be valid, it implies that the transaction is issued by the input coins' real owner and the owner will not be able to deny this transaction in the future.

A.3 Ring Signature

Ring Signatures, introduce by by Rivest et al. [74], enable a user to sign a message so that a 'ring' of possible signers (which the user is a member) is identified, without revealing exactly which member of that ring actually generated the signature. For ring signature, users may be unaware of each other at the time they generate their public keys and rings may be formed completely "on-the-fly" and in an ad-hoc manner, and users are given fine-grained control over the level of anonymity associated with any particular signature (via selection of an appropriate ring).

The formal definitions below are due to Bender et al. [12, 13], which, to the best of our knowledge, proposes the strongest security model.

Definition 9 (Ring Signature) *A* ring signature scheme *is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *that respectively, generate keys for a user, sign a message, and verify the signature of a message. Formally:*

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{PK}, \mathsf{SK})$. *The algorithm takes as input a security parameter* λ*, and outputs a public key* PK *and secret key* SK*.*

- $\mathsf{Sign}(M, R, s, \mathsf{SK}) \to \sigma$. *The algorithm takes as input a message* M*, a ring* $R = (\mathsf{PK}_1, \ldots, \mathsf{PK}_n)$*, an index* $s \in \{1, \ldots, n\}$*, and a secret key* SK*, and outputs a signature* σ *on the message* M *with respect to the ring* R*. We*

assume the following: (1) $(R[s], \mathsf{SK})$ *is a valid key-pair output by* Gen*; (2)* $|R| \geq 2$ *(since a ring signature is not intended to serve as a standard signature scheme); and (3) each public key on the ring is distinct. Note that the first condition simply models ring signature useage (where a signer "knows" its index* s *in the ring).*

- $\mathsf{Vrfy}(M, R, \sigma) \to 1/0$. *The algorithm takes as input a message* M*, a ring of public keys* R*, and a purported signature* σ *on the message* M *with respect to the ring* R*, and outputs a single bit indicating validity or invalidity.*

Correctness. For any λ, any $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n}$ output by $\mathsf{Gen}(1^\lambda)$, and $s \in \{1, \ldots, n\}$, and any message M, we have $\mathsf{Vrfy}(M, R, \mathsf{Sign}(M, R, s, \mathsf{SK}_s)) = 1$, where $R = (\mathsf{PK}_1, \ldots, \mathsf{PK}_n)$.

The security definitions for ring signature includes anonymity and unforgeability. Below we review the security definitions in [12, 13], which give the models from weak to strong, capturing the adversaries' ability.

A.3.1 Anonymity of Ring Signature

Definition 10 (Basic anonymity) *Given a ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, a polynomial* $n(\cdot)$*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. *Key pairs* $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{Gen}(1^\lambda)$*, and the set of public keys* $S := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot, \cdot, \cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$*, where it is required that* $R \subseteq S$ *and* $\mathsf{PK}_s \in R$*.*

3. $\mathcal{A}$ *outputs a message* M*, distinct indices* i_0, i_1*, and a ring* $R \subseteq S$ *for which* $\mathsf{PK}_{i_0}, \mathsf{PK}_{i_1} \in R$*. A random bit* b *is chosen, and* $\mathcal{A}$ *is given the signature* $\sigma \leftarrow \mathsf{Sign}(M, R, i_b, \mathsf{SK}_{i_b})$*.*

4. $\mathcal{A}$ *outputs a bit* b'*, and succeeds if* $b' = b$*.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ achieves basic anonymity if, for any PPT $\mathcal{A}$ and any polynomial $n(\cdot)$, the success probability of $\mathcal{A}$ in the above game is negligibly close to 1/2.

Remark. The above definition of basic anonymity leaves open the possibility of the following attack: (1) an adversary generates public keys in some arbitrary manner (which may possibly depend on the public keys of the honest users), and then (2) a legitimate signer generates a signature with respect to a ring containing some of these adversarially-generated public keys.

Definition 11 (Anonymity w.r.t. adversary-chosen keys) *Given a ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, a polynomial* $n(\cdot)$*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. *Key pairs* $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{Gen}(1^\lambda)$*, and the set of public keys* $S := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot, \cdot, \cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$*, where it is required that* $\mathsf{PK}_s \in R \cap S$*.* ***(Note that, in contrast to Definition 10, it no longer requires*** $R \subseteq S$***).***

3. $\mathcal{A}$ *outputs a message* M*, distinct indices* i_0, i_1*, and a ring* R *for which* $\mathsf{PK}_{i_0}, \mathsf{PK}_{i_1} \in R \cap S$*.* ***(Again, it no longer requires*** $R \subseteq S$***).*** *A random bit* b *is chosen, and* $\mathcal{A}$ *is given the signature* $\sigma \leftarrow \mathsf{Sign}(M, R, i_b, \mathsf{SK}_{i_b})$*.*

4. $\mathcal{A}$ *outputs a bit* b'*, and succeeds if* $b' = b$*.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ achieves anonymity w.r.t. adversary-chosen keys if, for any PPT $\mathcal{A}$ and any polynomial $n(\cdot)$, the success probability of $\mathcal{A}$ in the above game is negligibly close to 1/2.

Remark. (1) Note that the above definition of anonymity w.r.t. adversary-chosen keys is different from the basic anonymity in Definition 10 lies only in that it no longer requires $R \subseteq S$. Thus, it can defend against the attack mentioned in above remark. (2) On the other side, the above definition only guarantees anonymity of

a particular signature as long as there are at least two honest users in the ring. In some sense this is inherent, since if an honest signer U chooses a ring in which all other public keys (i.e., except for the public key of U) were created by an adversary, then that adversary "knows that U must be the signer (since the adversary did not generate the signature itself). As this case is inherent, a weaker requirement one might consider when the signer U is the only honest user in a ring is that the other members of the ring should be unable to prove to a third party that U generated the signature (this is called an attribution attack).

Definition 12 (Anonymity against attribution attack/full key exposure) *Given a ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, a polynomial* $n(\cdot)$*, and a PPT adversary* $\mathcal{A}$*, consider the following game:*

1. *For* $i = 1$ *to* $n(\lambda)$*, generate* $(\mathsf{PK}_i, \mathsf{SK}_i) \leftarrow \mathsf{Gen}(1^\lambda, \omega_i)$ *for randomly chosen* ω_i*. The set of public keys* $S := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$*.* ***(Note that it makes explict the random coins used to generate keys.)***

2. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot, \cdot, \cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$*, where it is required that* $\mathsf{PK}_s \in R \cap S$*.*

3. $\mathcal{A}$ *is given access to an oracle* $\mathsf{Corrupt}(\cdot)$ *that, on input* i*, returns* ω_i*.*

4. $\mathcal{A}$ *outputs a message* M*, distinct indices* i_0, i_1*, and a ring* R *for which* $\mathsf{PK}_{i_0}, \mathsf{PK}_{i_1} \in R \cap S$*. A random bit* b *is chosen, and* $\mathcal{A}$ *is given the signature* $\sigma \leftarrow \mathsf{Sign}(M, R, i_b, \mathsf{SK}_{i_b})$*.*

5. $\mathcal{A}$ *outputs a bit* b'*, and succeeds if* $b' = b$ *and* $|\{i_0, i_1\} \cap C| \leq 1$*, where* C *is the set of queries to the corruption oracle.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ achieves anonymity against attribution attacks if, for any PPT $\mathcal{A}$ and any polynomial $n(\cdot)$, the success probability of $\mathcal{A}$ in the above game is at most $1/2 + negl(\lambda)$. If we allow $|\{i_0, i_1\} \cap C| = 2$, then we say $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ achieves anonymity against full key exposure.

Remark. To the best of our knowledge, this is the strongest anonymity definition for ring signature.

A.3.2 Unforgeability of Ring Signature

Definition 13 (Unforgeability against fixed-ring attacks) *A ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is unforgeable against fixed-ring attacks if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the following game is neligible:*

1. *Key pairs* $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{Gen}(1^\lambda)$*, and the set of public keys* $R := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to a signing oracle* $OSign(\cdot, \cdot)$*, where* $OSign(s, M)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$*.*

3. $\mathcal{A}$ *outputs* (M^*, σ^*) *and succeeds if* $\mathsf{Vrfy}(\mathsf{M}^*, \mathsf{R}, \sigma^*) = 1$ *and also* $\mathcal{A}$ *never made a query of the form* $OSign(*, M^*)$*.*

Remark. Note that not only is $\mathcal{A}$ restricted to making signing queries with respect to the full ring R, but its forgery is required to verify with respect to R as well. The following definition is stronger and more natural.

Definition 14 (Unforgeability against chosen-subring attacks) *A ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is unforgeable against chosen-subring attacks if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the following game is negligible:*

1. *Key pairs* $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{Gen}(1^\lambda)$*, and the set of public keys* $S := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to a signing oracle* $OSign(\cdot, \cdot, \cdot)$*, where* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$*, where it is required that* $R \subseteq S$ *and* $\mathsf{PK}_s \in R$*.*

3. $\mathcal{A}$ *outputs* (M^*, R^*, σ^*) *and succeeds if* $R^* \subseteq S$, $\mathsf{Vrfy}(\mathsf{M}^*, \mathsf{R}, \sigma^*) = 1$, *and* $\mathcal{A}$ *never queried* $(*, M^*, R^*)$ *to its signing oracle.*

Remark. The above definition still leaves open the possibility of an attack whereby honest users are "tricked" into generating signatures using rings containing adversarially-generated public keys. The following definition takes this into account as well as, for completeness, an adversary who adaptively corrupts honest participants and obtains their secret keys.

Definition 15 (Unforgeability w.r.t. insider corruption)) *A ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is unforgeable w.r.t. insider corruption if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$, *the probability that* $\mathcal{A}$ *succeeds in the following game is neligible:*

1. *Key pairs* $\{(\mathsf{PK}_i, \mathsf{SK}_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{Gen}(1^\lambda)$, *and the set of public keys* $S := \{\mathsf{PK}_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$.

2. $\mathcal{A}$ *is given access to a signing oracle* $OSign(\cdot, \cdot, \cdot)$, *where* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, s, \mathsf{SK}_s)$, *where it is required that* $\mathsf{PK}_s \in R \cap S$.

3. $\mathcal{A}$ *is also given access to a corrupt oracle* $\mathsf{Corrupt}(\cdot)$, *where* $\mathsf{Corrupt}(i)$ *outputs* SK_i .

4. $\mathcal{A}$ *outputs* (M^*, R^*, σ^*) *and succeeds if* $\mathsf{Vrfy}(\mathsf{M}^*, \mathsf{R}, \sigma^*) = 1$, $\mathcal{A}$ *never queried* $(*, M^*, R^*)$, *and* $R^* \subseteq S \setminus C$, *where* C *is the set of corrupted users.*

A.3.3 Related Work of Ring Signature

Rivest et al. [74] introduced the concept of Ring signatures, and Bender et al. [12, 13] provided rigorous security definitions for ring signatures and generic constructions based on trapdoor permutations. Shacham and Waters [78] gave a ring signature scheme which is proved to be anonymous against full key exposure and unforgeable with respect to insider corruption in the standard model. The scheme

in [78] is based on paring assumptions and use common reference strings which require a trusted setup phase. On efficiency, the scheme in [78] is more efficient than that of [12, 13], but the signature size is still linear in the number of ring numbers. Chandran et al. [21] proposed a sub-linear size (linear in the square root of the ring size) ring signature scheme in the common reference string model (implying trusted setup is required) under pairing-based assumptions, which is proved anonymous against full key exposure and unforgeable with respect to insider corruption in the standard model. On signature size, it is worth mentioning that Dodis et al. [30] suggested a scheme with constant signature size independent of ring members, but it relies on random oracle and trusted setup.

Groth and Kohlweiss [41] proposed a ring signature scheme derived from a Sigma-protocol they proposed. The ring signature scheme in [41] does not need trusted setup and achieves logarithmic signature size, but its security is proved in the random oracle model.

Libert et al. [50] proposed a ring signature scheme derived from lattice-based accumulator protocol, and thus this scheme is post-quantum secure, while it still relies on random oracle. The scheme achieves logarithmic signature size, and is very efficient in the lattice-based cryptography, with PK size of $4.9MB$, SK size $3.25KB$, and signature size $61.5MB$.

As Ring Signature can hide the identity of the signer in a set of possible signers, a variant of ring signature, called Traceable Ring Signature [36], was used by CryptoNote [84] to have the payer's public key of a transaction hidden in a group of public keys all of which contain the same amount of coins, so that no one can tell which user actually sent the coins.

A.4 Linkable Ring Signature

While Ring Signature provides strong anonymity and is applicable to the scenarios of leaking a secret safely, Liu et al. [55] extended the concept to Linkable Ring Signature, where an additional property – linkability is added. Linkable Ring

Signature is proposed for the scenarios of e-vote, where each signer is expected to sign only once, but it is obvious that Linkable Ring Signature is also a potential tool for achieving user privacy in cryptocurrency, as it simultaneously (1) has the functionalities of Signature, (2) can hide the signer's public key in a set of possible public keys, and (4) can prevent double-spending as signing twice for the same public key (i.e. spending the same coins twice) will be detected. Actually, while the initial Monero and the original CryptoNote [84] protocol are derived from the Traceable Ring signatures of [36], the recently improved Monero protocol, Ring Confidential Transactions (RingCT) [67, 68], which attempts to hide the identities of the payer/payee and the transaction amount, is derived from the Linkable Ring Signature constructions by Liu et al.[55].

The Linkable Ring Signature definitions below is extended from that of Au et al. [5].

Definition 16 *A* **Linkable Ring Signature** *scheme is a tuple* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Link})$ *of four poly-time algorithms:*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. *The algorithm takes as input a security parameter* λ *and outputs a public/secret key pair* $(\mathsf{pk}_i, \mathsf{sk}_i)$. *When we say that a public key corresponds to a secret key or vice versa, we mean that the secret/public key pair is an output of* KeyGen.

- $\mathsf{Sign}(M, R, sk) \to \sigma$. *The algorithm takes as input a message* M, *a ring* $R = (pk_1, \ldots, pk_n)$, *and a secret key* sk *whose corresponding public key is contained in* R, *and outputs a signature* σ *on the message* M *with respect to the ring* R.

- $\mathsf{Vrfy}(M, R, \sigma) \to 1/0$. *The algorithm takes as input a message* M, *a ring of public keys* R, *and a purported signature* σ *on the message* M *with respect to the ring* R, *and outputs a single bit indicating validity or invalidity.*

- $\mathsf{Link}(M_0, R_0, \sigma_0, M_1, R_1, \sigma_1) \rightarrow 1/0$. *The algorithm takes as input two valid signatures* (M_0, R_0, σ_0), (M_1, R_1, σ_1), *and outputs a single bit indicating* linked *or* unlinked.

Correctness. Linkable Ring Signature schemes must satisfy:

1. **Verification Correctness**. Signatures signed according to specification are accepted during verification, with overwhelming probability; and
2. **Linking Correctness**. Two signatures signed according to specification are linked with overwhelming probability if the two signatures share a common signer. On the other hand, two signatures signed according to specification are unlinked with overwhelming probability if the two signatures do not share a common signer.

Security. The security definitions for Linkable Ring Signature includes linkable-anonymity, unforgeability, linkability, and non-slanderability. Below we review the security definitions.

A.4.1 Link-anonymity of Linkable Ring Signature

Definition 17 (link-anonymity w.r.t. insider corruption) *A Linkable Ring Signature scheme is linkable-anonymous w.r.t. insider corruption if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$, *the probability that* $\mathcal{A}$ *succeeds in the following game is negligibly close to* $1/2$*:*

1. **Setup**. *Key pairs* $\{(pk_i, sk_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{KeyGen}(1^\lambda)$, *and the set of public keys* $S := \{pk_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$.
2. **Probing Phase 1**. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot, \cdot, \cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, sk_s)$, *where it is required that* $\boxed{R \subseteq S}$ *and* $\underline{pk_s \in R \cap S}$. $\mathcal{A}$ *is also given access to a corruption oracle* $CO(\cdot)$ *where* $CO(\cdot)$ *outputs* sk_i *and it is required that* $1 \leq i \leq n(\lambda)$.

3. **Challenge Phase**. *$\mathcal{A}$ outputs a message M, distinct indices i_0, i_1, and a ring $\boxed{R \subseteq S}$ for which $\underline{pk_{i_0}, pk_{i_1} \in R \cap S}$. It is required that none of $OSign(i_0, \cdot, \cdot)$, $OSign(i_1, \cdot, \cdot)$, $CO(i_0)$, $CO(i_1)$ is queried. A random bit b is chosen, and $\mathcal{A}$ is given the signature $\sigma \leftarrow \mathsf{Sign}(M, R, sk_{i_b})$.*

4. **Probing Phase 2**. *Same as the* **Probing Phase 1**, *but with the restriction that none of $OSign(i_0, \cdot, \cdot)$, $OSign(i_1, \cdot, \cdot)$, $CO(i_0)$, $CO(i_1)$ is queried.*

5. **Output Phase**. *$\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.*

Note that for $R \subseteq S$, the underlined parts $\underline{pk_s \in R \cap S}$ and $\underline{pk_{i_0}, pk_{i_1} \in R \cap S}$ are actually $\underline{pk_s \in R}$ and $\underline{pk_{i_0}, pk_{i_1} \in R}$, respectively. This writing is for the link-anonymity w.r.t. adversarially-chosen keys definition below.

Definition 18 (link-anonymity w.r.t. adversarially-chosen keys) *A Linkable Ring Signature scheme is link-anonymous w.r.t. adversarially-chosen keys if for any PPT adversary $\mathcal{A}$ and for any polynomial $n(\cdot)$, the probability that $\mathcal{A}$ succeeds in the above game, $\boxed{\text{without the requirement of } R \subseteq S \text{ for } OSign(\cdot, \cdot, R) \text{ in}}$ $\boxed{\textbf{Probing Phase 1 and 2} \text{ and without restricting } R \subseteq S \text{ in } \textbf{Challenge Phase},}$ is negligibly close to $1/2$.*

Remark: *Note that link-anonymity w.r.t. adversarially-chosen keys is stronger than link-anonymity w.r.t. insider corruption, and a link-anonymous w.r.t. adversarially-chosen keys Linkable Ring Signature is more desirable in practice, as it is obvious that an attacker in practice can easily launch chosen-keys attacks.*

A.4.2 Unforgeability of Linkable Ring Signature

Definition 19 (Unforgeability w.r.t insider corruption) *A Linkable Ring Signature scheme is unforgeable w.r.t. insider corruption if for any PPT adversary $\mathcal{A}$ and for any polynomial $n(\cdot)$, the probability that $\mathcal{A}$ succeeds in the following game is negligible.*

1. **Setup**. *Key pairs* $\{(pk_i, sk_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{KeyGen}(1^\lambda)$*, and the set of public keys* $S := \{pk_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$.

2. **Probing Phase**. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot,\cdot,\cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, sk_s)$*, where it is required that* $\boxed{R \subseteq S}$ *and* $\underline{pk_s \in R \cap S}$. $\mathcal{A}$ *is also given access to a corruption oracle* $CO(\cdot)$ *where* $CO(\cdot)$ *outputs* sk_i *and it is required that* $1 \leq i \leq n(\lambda)$.

3. **Output Phase**. $\mathcal{A}$ *outputs* (M^*, R^*, σ^*) *and succeeds if* $\mathsf{Vrfy}(\mathsf{M}^*, \mathsf{R}^*, \sigma^*) = 1$, $\mathcal{A}$ *never queried* $(*, M^*, R^*)$ *to its signing oracle, and* $R^* \subseteq S \setminus C$*, where* C *is the set of corrupted users.*

Note that for $R \subseteq S$, the underlined part $\underline{pk_s \in R \cap S}$ is actually $\underline{pk_s \in R}$. This writing is for the unforgeability w.r.t. adversarially-chosen keys definition below.

Definition 20 (Unforgeability w.r.t adversarially-chosen keys) *A Linkable Ring Signature scheme is unforgeable w.r.t. adversarially-chosen keys if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the above game,* $\boxed{\text{without the requirement of } R \subseteq S \text{ for } OSign(\cdot,\cdot,R) \text{ in } \textbf{Probing Phase}}$*, is negligible.*

Remark: *Note that unforgeability w.r.t. adversarially-chosen keys is stronger than unforgeability w.r.t. insider corruption, and a unforgeable w.r.t. adversarially-chosen keys Linkable Ring Signature is more desirable in practice, as it is obvious that an attacker in practice can easily launch chosen-keys attacks.*

A.4.3 Linkability of Linkable Ring Signature

Definition 21 (Linkability w.r.t. insider corruption) *A Linkable Ring Signature scheme is linkable w.r.t. insider corruption if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the following game is negligible.*

1. **Setup**. *Key pairs* $\{(pk_i, sk_i)\}_{i=1}^{n(\lambda)}$ *are generated using* $\mathsf{KeyGen}(1^\lambda)$*, and the set of public keys* $S := \{pk_i\}_{i=1}^{n(\lambda)}$ *is given to* $\mathcal{A}$.

2. **Probing Phase**. $\mathcal{A}$ *is given access to an oracle* $OSign(\cdot,\cdot,\cdot)$ *such that* $OSign(s, M, R)$ *returns* $\mathsf{Sign}(M, R, sk_s)$*, where it is required that* $\boxed{R \subseteq S}$ *and* $\underline{pk_s \in R \cap S}$. $\mathcal{A}$ *is also given access to a corruption oracle* $CO(\cdot)$ *where* $CO(\cdot)$ *outputs* SK_i *and it is required that* $1 \leq i \leq n(\lambda)$.

3. **Output Phase**. $\mathcal{A}$ *outputs* $(M_i^*, R_i^*, \sigma_i^*)$ $(i = 1, 2)$*, and succeeds if it holds that* $\mathsf{Vrfy}(\mathsf{M}_\mathsf{i}^*, \mathsf{R}_\mathsf{i}^*, \sigma_\mathsf{i}^*) = 1$, $\boxed{R_i^* \subseteq S}$ *for* $i = 1, 2$, $Link(\sigma_1^*, \sigma_2^*) = 0$, *and* $\underline{|(R_1^* \cup R_2^*) \cap C| + |(R_1^* \cup R_2^*) \setminus S| \leq 1}$*, where* C *is the set of corrupted users.*

Note that: (1) in **Probing Phase**, for $R \subseteq S$, the underlined part $\underline{pk_s \in R \cap S}$ is actually $\underline{pk_s \in R}$; and (2) in **Output Phase**, for $R_i^* \subseteq S$ $(i = 1, 2)$, the underlined part $\underline{|(R_1^* \cup R_2^*) \cap C| + |(R_1^* \cup R_2^*) \setminus S| \leq 1}$ is actually $\underline{|(R_1^* \cup R_2^*) \cap C| \leq 1}$. This writing is for the linkability w.r.t. adversarially-chosen keys definition below.

Definition 22 (Linkability w.r.t. adversarially-chosen keys) *A linkable ring signature scheme is linkable w.r.t. adversarially-chosen keys if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the above game,* $\boxed{\text{without requiring } R \subseteq S \text{ for } OSign(\cdot,\cdot,\cdot) \text{ in } \textbf{Probing Phase} \text{ and without restricting } R_i^* \subseteq S \text{ in } \textbf{Output Phase}}$*, is negligible.*

Remark: *Note that linkability w.r.t. adversarially-chosen keys is stronger than linkability w.r.t. insider corruption, and a linkable w.r.t. adversarially-chosen keys Linkable Ring Signature is more desirable in practice, as it is obvious that an attacker in practice can easily launch chosen-keys attacks.*

A.4.4 Non-slanderability of Linkable Ring Signature

Definition 23 (Non-slanderability w.r.t. insider corruption) *A Linkable Ring Signature scheme is non-slanderable w.r.t. insider corruption if for any PPT ad-*

versary $\mathcal{A}$ and for any polynomial $n(\cdot)$, the probability that $\mathcal{A}$ succeeds in the following game is negligible.

1. **Setup**. *Key pairs $\{(pk_i, sk_i)\}_{i=1}^{n(\lambda)}$ are generated using $\mathsf{KeyGen}(1^\lambda)$, and the set of public keys $S := \{pk_i\}_{i=1}^{n(\lambda)}$ is given to $\mathcal{A}$.*

2. **Probing Phase**. *$\mathcal{A}$ is given access to an oracle $OSign(\cdot,\cdot,\cdot)$ such that $OSign(s, M, R)$ returns $\mathsf{Sign}(M, R, sk_s)$, where it is required that $\boxed{R \subseteq S}$ and $\underline{pk_s \in R \cap S}$. $\mathcal{A}$ is also given access to a corruption oracle $CO(\cdot)$ where $CO(\cdot)$ outputs sk_i and it is required that $1 \leq i \leq n(\lambda)$.*

3. **Output Phase**. *$\mathcal{A}$ outputs $(\hat{\sigma}, M^*, R^*, \sigma^*)$, and succeeds if it holds that $\hat{\sigma}$ is the output of $OSign(\hat{s}, \hat{M}, \hat{R})$ for some $\boxed{\hat{R} \subseteq S}$ and $\hat{s}$ such that $\underline{pk_{\hat{s}} \in \hat{R} \cap S}$, $\boxed{R^* \subseteq S}$, $\mathsf{Vrfy}(\mathsf{M}^*, \mathsf{R}^*, \sigma^*) = 1$, $Link(\hat{\sigma}, \sigma^*) = 1$, and $\mathcal{A}$ never queries $\hat{s}$ to $CO(\cdot)$.*

Note that: (1) in **Probing Phase**, for $R \subseteq S$, the underlined part $\underline{pk_s \in R \cap S}$ is actually $\underline{pk_s \in R}$; and (2) in **Output Phase**, for $\hat{R} \subseteq S$, the underlined part $\underline{pk_{\hat{s}} \in \hat{R} \cap S}$ is actually $\underline{pk_{\hat{s}} \in \hat{R}}$. This writing is for the non-slanderability w.r.t. adversarially-chosen keys definition below.

Definition 24 (Non-slanderability w.r.t. adversarially-chosen keys) *A Linkable Ring Signature scheme is non-slanderable w.r.t. adversarially-chosen keys if for any PPT adversary $\mathcal{A}$ and for any polynomial $n(\cdot)$, the probability that $\mathcal{A}$ succeeds in the above game, $\boxed{\text{without requiring } R \subseteq S \text{ for } OSign(\cdot,\cdot,\cdot) \text{ in } \textbf{Probing Phase}}$ $\boxed{\text{and without restricting } \hat{R} \subseteq S, R_i^* \subseteq S \text{ in } \textbf{Output Phase}}$, is negligible.*

Remark: *Note that non-slanderability w.r.t. adversarially-chosen keys is stronger than non-slanderability w.r.t. insider corruption, and a non-slanderable w.r.t. adversarially-chosen keys Linkable Ring Signature is more desirable in practice, as it is obvious that an attacker in practice can easily launch chosen-keys attacks.*

A.4.5 Related Work of Linkable Ring Signature

Linkable Ring Signature was first introduced by Liu et al. [55]. Then, Liu and Wong [56] enhanced the security model, and Tsang and Wei [83] enhanced the security and proposed a short linkable ring signature scheme whose signature size is independent of the number of ring members. Noted that [55, 56, 83] did not consider the insider attack, Au et al. [5] refined the security definitions which we review above. In particular, [55] did not consider linkable anonymity against insider corruption, [56] and [83] are linkable anonymous against insider corruption, and [5] is linkable anonymous w.r.t. adversarially-chosen keys; [55] and [56] are unforgeable against chosen-subring attack, [83] is unforgeable w.r.t. insider corruption, and [5] is unforgeable w.r.t. adversarially-chosen keys; [55] and [56] are linkable against chosen-subring attack, [83] is linkable w.r.t. insider corruption, and [5] is linkable w.r.t. adversarially-chosen keys; [55] and [56] can be proven non-slanderable w.r.t. adversarially-chosen keys [27], [83] is non-slanderable w.r.t. insider corruptions, and [5] is non-slanderable w.r.t. adversarially-chosen keys. All the schemes in [55, 56, 83, 5] rely on random oracle model. The signature size in [55, 56] is linear in the ring size, while the signature size in [83, 5] is independent of the ring size. However, while the security of the schemes in [55, 56] is based on on generally accepted DDH problem and Discrete Logarithm problem, the security of [83, 5] is based on a very strong new assumption such that, given two distinct RSA moduli $n_1 = p_1q_1$ and $n_2 = p_2q_2$, adversary cannot distinguish $g^{p_1+q_1}$ from $g^{p_2+q_2}$, where g is in QR(N) for RSA modules N. This implies that [83, 5] require a trusted setup phase. *In summary, the security of [55, 56, 83] are weak due to their weak security model, while [5] has a stronger security model* [28]*, but [5] relies on a very strong new assumption and requires a trusted setup phase.*

Another related concept that is worth mentioning is Traceable Ring Signature

[27] None of the models in [55, 56, 83] considers non-slanderable w.r.t. adversarially-chosen keys

[28] Note that the security model may be further enhanced to support unforgeability against adversarially-chosen key attacks

[36], which was proposed by Fujisaki and SuzuKi. Traceable Ring Signature is similar to Linkable Ring Signature, but with the functionality that, if a signer signs two different messages on the same ring, the identity will be revealed. While the traceable signature scheme in [36] has signature size linear in the ring size and relies on random oracle model, Fujisaki [35] proposed a traceable ring signature which has sub-linear signature size (linear in the square root of the ring size) and is secure in the standard model.

While the latest Monero bases its protocol on the Linkable Ring Signature scheme by Liu et al. [55], the scheme does not consider insider attacks. Instead, they only considered the chosen-subring attack. More specifically, the link-anonymity, unforgeability, linkability, and non-slanderability definitions with respect to chosen-subring are similar to that of insider corruption, except that the adversaries are not given access to the corruption oracle $CO(\cdot)$. In other words, link-anonymity, unforgeability, linkability, and non-slanderability definitions with respect to chosen-subring is much weaker than that of insider corruption and that of adversarially-chosen keys. In the setting of cryptocurrencies, any (potentially malicious) user could launch adversarially-chosen keys attacks, once his public key(s) are included in some ring. *Abelian Coin protocols will be based on the Linkable Ring Signature schemes resisting adversarially-chosen keys attacks, which are more secure and more practical in the setting of cryptocurrencies.*

Liu et al. [55]'s scheme is the initial linkable ring signature, which was analyzed only in *weak* security models, and that the derived protocols may inherit the weakness.

A.5 Commitment

While Ring Signature can help hide the identities of the payers and payees of transaction, Commitment is a very useful cryptographic tool which can help hide the transaction amounts.

In cryptography, a Commitment scheme allows a committer to commit to a

certain value while keeping it hidden from the public (confidentiality) and without being able to change its mind later on (bindingness). Below is a brief view of non-interactive Commitment definitions [29].

Definition 25 (Commitment Scheme) *A (non-interactive) Commitment Scheme (for a message space $\mathcal{M}$) is a tuple* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *of three poly-time algorithms:*

1. $\mathsf{Setup}(1^k) \to \mathsf{CK}$ *generates the public commitment key* CK.
2. *for any* $m \in \mathcal{M}$, $(c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$ *is the (commitment, opening) pair for* m. $c = c(m)$ *serves as the commitment value, and* $d = d(m)$ *as the opening value. We often omit mentioning the public commitment key* CK *when it is clear from the context.*
3. $\mathsf{Open}_{\mathsf{CK}}(c, d) \to \tilde{m} \in \mathcal{M} \cup \perp$, *where* $\perp$ *is returned if* c *is not a valid commitment to any message. We often omit mentioning the public commitment key* CK *when it is clear from the context.*

Correctness. *Commitment Scheme must satisfy : for any* $m \in \mathcal{M}$,

$$\mathsf{Open}_{\mathsf{CK}}(\mathsf{Commit}_{\mathsf{CK}}(m)) = m.$$

Here is how a Commitment Scheme is used. When a committer $\mathcal{C}$ wants to commit a value m to a receiver $\mathcal{R}$ (using the commitment key CK which we don't explicitly mention below), a two-phase protocol is run as below.

- **Commit Phase.** The committer $\mathcal{C}$ runs $(c, d) \leftarrow \mathsf{Commit}(m)$, and sends c to the receiver $\mathcal{R}$.
- **Reveal Phase.** The committer $\mathcal{C}$ sends d and m to the receiver $\mathcal{R}$. Then $\mathcal{R}$ runs $\tilde{m} \leftarrow \mathsf{Open}(c, d)$. $\mathcal{R}$ accepts if $\tilde{m} \neq \perp$ and $\tilde{m} = m$, otherwise $\mathcal{R}$ rejects. By correctness, if both parties are honest, $\mathcal{R}$ accepts.

Security. A Commitment Scheme should have the security properties: (1) the commitment c gives the receiver no information about the value m, and (2) the opening d cannot open c in two different ways (i.e. with a value different from the committed value). The properties are called *hiding* and *binding*.

1. **Hiding**. It is computationally hard for any adversary $\mathcal{A}$ to generate two messages $m_0, m_1 \in \mathcal{M}$ such that $\mathcal{A}$ can distinguish between their commitment c_0, c_1.

2. **Binding**. It is computationally hard for any adversary $\mathcal{A}$ to come up with a triple (c, d, d'), referred to as a *collision*, such that (c, d) and (c, d') are valid commitments for m and m' with $m \neq m'$.

Due to the properties of hiding, binding and non-interactivity, (non-interactive) Commitment Schemes have been used by Monero, ZeroCoin, and ZeroCash, to achieve transaction amounts hiding.

For a transaction in cryptocurrency, it is required that **the total number of output coins is equal to that of the input coins**. When the transaction amounts are hidden using Commitment Schemes, to check this condition, we need the Commitment Schemes to be *additively Homomorphic*.

Additively Homomorphic Commitment Scheme. Let the message space is $\mathcal{M} = \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ for some p, a Commitment Scheme is Additively Homomorphic, if it holds that: for any message $m_1, m_2 \in \mathbb{Z}_p$, let $(c_1, d_1) \leftarrow \mathsf{Commit}(m_1)$, $(c_2, d_2) \leftarrow \mathsf{Commit}(m_2)$, $\mathsf{Open}(c_1 \otimes c_2, d_1 \odot d_2) = m_1 + m_2$ holds, where $\otimes$ and $\odot$ are binary operations defined over the commitments and openings respectively, and $+$ is module addition over $\mathbb{Z}_p$.

A.6 Zero-Knowledge Proof

In a cryptocurrency, when the transaction amounts are hidden, we need a cryptographic range proof to show that the amount in a legal range. At the same time,

the proof should leak no information about the amount value, as we are trying to hide the amount. This requires the rang proof is a zero-knowledge proof.

Roughly speaking, zero-knowledge proof is a cryptographic tool, which is a way for somebody to prove a (mathematical) statement without revealing any other information that leads to that statement being true. For example, a prover may want to prove a statement such as "I know x such that $H(x)$ belongs to the set $\{\dots\}$". He could do this by revealing x. But a zero-knowledge proof allows him to do this in such a way that the other person (verifier) is no wiser about the value of x after seeing the proof than they were before.

Below we briefly review the formal definitions for zero-knowledge proof.

Definition 26 (Non-interactive Zero-Knowledge, NIZK [33]) *$\Pi = (l, P, V, S = (S_1, S_2))$ is a single-theorem NIZK proof system for the language $L \in NP$ with witness relation R, if l is a polynomial, and P, V, S_1, S_2 are all probabilistic polynomial-time machines such that there exists a negligible function α such that for all k:*

- **Completeness***: For all $x \in L$ of length k and all w such that $R(x, w) =$ True, for all strings σ of length $l(k)$, we have that $V(x, P(x, w, \sigma), \sigma) = \mathsf{True}$.*
- **Soundness***: For all unbounded (resp. polynomial-time) adversaries $\mathcal{A}$, if $\sigma \in \{0,1\}^{l(k)}$ is chosen randomly, then the probability that $\mathcal{A}(\sigma)$ will output (x, p) such that $x \notin L$ but $V(x, p, \sigma) = \mathsf{True}$ is less than $\alpha(k)$.*
- **Single-Theorem Zero Knowledge***: For all non-uniform PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have that*

$$|\Pr[\mathsf{Expt}_{\mathcal{A}}(k) = 1] - \Pr[\mathsf{Expt}^{S}_{\mathcal{A}}(k)]| \leq \alpha(k),$$

where the experiments $\mathsf{Expt}_{\mathcal{A}}(k)$ and $\mathsf{Expt}^{S}_{\mathcal{A}}(k)$ are defined as follows:

$\mathsf{Expt}_{\mathcal{A}}(k)$:

$$\sum \leftarrow \{0,1\}^{l(k)}$$
$$(x, w, s) \leftarrow \mathcal{A}_1(\sum)$$
$$p \leftarrow P(x, w, \sum)$$
$$return\ \mathcal{A}_2(p, s)$$

$\mathsf{Expt}^{S}_{\mathcal{A}}(k)$:

$$(\sum, \tau) \leftarrow S_1(1^k)$$
$$(x, w, s) \leftarrow \mathcal{A}_1(\sum)$$
$$p \leftarrow S_2(x, \sum, \tau)$$
$$return\ \mathcal{A}_2(p, s)$$

To define a notion of NIZK where any polynomial number of proofs can be simulated, we change the Zero-knowledge condition as follows:

Definition 27 (unbounded NIZK [33]) *$\Pi = (l, P, V, S = (S_1, S_2))$ is an unbounded NIZK proof system for the language $L \in NP$, if Π is single-theorem NIZK proof system for L and furthermore: there exists a negligible function α such that for all k:*
*(***Unbounded Zero Knowledge***): For all non-uniform PPT adversaries A, we have that $|\Pr[\mathsf{Expt}(k) = 1] - \Pr[\mathsf{Expt}^{S}_{A}(k) = 1]| \leq \alpha(k)$, where the experiments $\mathsf{Expt}_{\mathcal{A}}(k)$ and $\mathsf{Expt}^{S}_{A}(k)$ are as defined as follows:*

$\mathsf{Expt}_{\mathcal{A}}(k)$:

$$\sum \leftarrow \{0,1\}^{l(k)}$$
$$return\ \mathcal{A}^{P(\cdot,\cdot,\sum)}(\sum)$$

$\mathsf{Expt}^{S}_{\mathcal{A}}(k)$:

$$(\sum, \tau) \leftarrow S_1(1^k)$$
$$return\ \mathcal{A}^{S'(\cdot,\cdot,\sum,\tau)}(\sum)$$

where $S'(x, w, \sum, \tau) \stackrel{def}{=} S_2(x, \sum, \tau)$.

A.7 zk-SNARK

For a field $\mathbb{F}$, an $\mathbb{F}$-*arithmetic circuit* takes elements in $\mathbb{F}$ as inputs, and its gates output elements in $\mathbb{F}$. In particular, the circuit has an input $x \in \mathbb{F}^n$ and an auxiliary input $a \in \mathbb{F}^h$ which is called a witness. A gate with inputs $y_1, \cdots, y_m \in \mathbb{F}$

is *bilinear* if its output is $(\vec{a}\cdot(1, y_1, \cdots, y_m))(\vec{b}\cdot(1, y_1, \cdots, y_m))$ for some $\vec{a}, \vec{b} \in \mathbb{F}^{m+1}$. Here we consider the circuits that only have bilinear gates.

An *arithmetic circuit satisfiability problem* of an $\mathbb{F}$-arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ is captured by the relation $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, a) = 0^l\}$, and its language is $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists a \in \mathbb{F}^h \text{ s.t. } C(x, a) = 0^l\}$.

Given a field $\mathbb{F}$, a *publicly-verifiable preprocessing zk-SNARK* [29] for $\mathbb{F}$-arithmetic circuit satisfiability is a triple of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{KeyGen}(1^\lambda, C) \to (\mathsf{pk}, \mathsf{vk})$. On input security parameter λ and an $\mathbb{F}$-arithmetic circuit C, the *key generator* KeyGen probabilistically samples a *proving key* pk and a *verification key* vk. Both keys are published as public parameters and can be used any number of times to prove/verify memberships in $\mathcal{L}_C$.

- $\mathsf{Prove}(\mathsf{pk}, x, a) \to \pi$. On input a proving key pk and any $(x, a) \in \mathcal{R}_C$, the *prover* Prove outputs a non-interactive proof π for the statement $x \in \mathcal{L}_C$.

- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to b$. On input a verification key vk, an input x, and a proof π, the *verifier* Verify outputs $b = 1$ if he is convinced that $x \in \mathcal{L}_C$.

A zk-SNARK satisfies the following properties:

- **Completeness**. For every security parameter λ, $\mathbb{F}$-arithmetic circuit C, and any $(x, a) \in \mathcal{R}_C$, the honest prover can convince the verifier. Namely, $b = 1$ with probability $1 - \mathrm{negl}(\lambda)$ in the following experiment: $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C)$; $\pi \leftarrow \mathsf{Prove}(\mathsf{pk}, x, a)$; $b \leftarrow \mathsf{Verify}(\mathsf{vk}, x, \pi)$. Informally, completeness ensures that a zk-SNARK functions correctly.

[29] SNARGs represents 'succinct non-interactive arguments'. If a SNARG satisfies a certain natural proof-of-knowledge property, it is called a SNARG of knowledge (SNARK). If it also satisfies a certain natural zero-knowledge property, it is called a zero-knowledge SNARK (zk-SNARK). The 'processing' implies the proofs are in the preprocessing model which relies on an expensive but reusable key generation.

- **Succinctness**. An honestly-generated proof π has $O_\lambda(1)$ bits and $\mathsf{Verify}(\mathsf{vk}, x, \pi)$ runs in time $O_\lambda(|x|)$. Informally, a zk-SNARK generates short proofs efficiently.

- **Proof of knowledge (soundness)**. For every poly(λ)-size adversary $\mathcal{A}$, there is a poly(λ)-size extractor $\mathcal{E}$ such that $\mathsf{Verify}(\mathsf{vk}, x, \pi) = 1$ and $(x, a) \notin \mathcal{R}_C$ with probability negl(λ) in the following experiment: $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, C)$; $(x, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk})$; $a \leftarrow \mathcal{E}(\mathsf{pk}, \mathsf{vk})$. Informally, if the verifier accepts a proof output by a bounded prover, then the prover "knows" a witness for the given instance.

- **Perfect zero knowledge**. An honestly-generated proof is perfect zero-knowledge. Namely, there is a polynomial-time simulator Sim such that for all stateful distinguishers $\mathcal{D}$ the following two probabilities are equal:

$$\Pr\left[\begin{array}{r|l} (x,a) \in \mathcal{R}_C & (\mathsf{pk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}(C) \\ \mathcal{D}(\pi) = 1 & (x,a) \leftarrow \mathcal{D}(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow \mathsf{Prove}(\mathsf{pk},x,a) \end{array}\right]$$

and

$$\Pr\left[\begin{array}{r|l} (x,a) \in \mathcal{R}_C & (\mathsf{pk},\mathsf{vk},\mathsf{trap}) \leftarrow \mathsf{Sim}(C) \\ \mathcal{D}(\pi) = 1 & (x,a) \leftarrow \mathcal{D}(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow \mathsf{Sim}(\mathsf{pk},x,\mathsf{trap}) \end{array}\right]$$

Informally, the proof π reveals no information about witness a.

A.7.1 Related Work of zk-SNARK

The study of zk-SNARK originates from the proof of the PCP theorem [6, 32, 4, 3], which states that NP statements have Probabilistically Checkable Proofs (PCPs) that can be verified in time polylogarithmic in the size of a classic proof. Based on collision-resistant hashes (CRHs) and PCPs, Kilian [45] showed a four-message interactive argument for NP which is succinct. A challenge which is of both theoretical and practical interest is the construction of non-interactive succinct argu-

ments. As a first step in this direction, Micali [64] provided a one-message succinct non-interactive argument for NP in the random oracle model. Gentry and Wichs [39] gave a formal definition of succinct non-interactive arguments (SNARG), and showed that SNARG can be proven secure only via non-standard assumptions.

Extending the above works, Bitansky et al. [14] improved Micali's construction by removing the random oracles, and replaced them with Extractable Collision-Resistant Hash functions (ECRHs), and obtained SNARGs of knowledge (SNARKs). Then, Bitansky et al. [15] proposed a weaker version of SNARK, called *preprocessing* SNARK, where the verifier is allowed to conduct an expensive offline phase that is independent of the statement to be proven later. In [15], Bitansky et al. gave a general technique for constructing zk-SNARKs. First, they define a linear PCP where the honest proof oracle is a linear function (over an underlying field) and soundness is required to hold only for linear proof oracles. Then, they show a transformation (also based on knowledge-of-exponent assumptions) from any linear PCP with a low-degree verifier to a SNARK; also, if the linear PCP is honest-verifier zero-knowledge (HVZK), the resulting SNARK is zero knowledge.

Gennaro et al. [37] introduced a new characterization of the NP complexity class, called Quadratic Span Programs (QSPs), which is a natural extension of span programs defined by Karchmer and Wigderson [42]. Using QSP and its adaption Quadratic Arithmetic Programs (QAPs), Gennaro et al. [37] constructed succinct arguments of NP-statements that are quick to construct and verify. QSPs and QAPs are expressive enough to allow for much simpler and more efficient cryptographic checking. In particular, they invested significant effort in obtaining an efficient reduction from circuit satisfiability to QSPs and QAPs.

Recent studies on SNARKs mostly based their work on that of Bitansky et al. [15] and Gennaro et al. [37]. Two [10, 70] of these work provided implementations. In particular,

- Parno et al. [70] presented two main contributions:

  - A zk-SNARK with essentially optimal asymptotics, for arithmetic circuit satisfiability. The construction is based on the QAPs [37], and an implementation for the construction is given.
  - A compiler that maps C programs with fixed memory accesses and bounded control flow (e.g. array accesses and loop iteration bounds are compile-time constants) into corresponding arithmetic circuits.

- Ben-Sasson et al. [10] presented three main contributions:
  - A zk-SNARK with essentially optimal asymptotics for arithmetic circuit satisfiability, and with a corresponding implementation. The construction is also based on QAPs, but follows the linear interactive proof approach of [15].
  - A simple RISC architecture, TinyRAM, along with a circuit generator for generating arithmetic circuits that verify correct execution of TinyRAM programs.
  - A compiler that, given a C program, produces a corresponding TinyRAM program.

Then, Ben-Sasson et al. [11] combined the works of [10] and [70] and made further optimizations. The main contributions include

- A new circuit generator that incorporates the following three main improvements.
  - The circuit generator supports programs on a new, more expressive architecture: vnTinyRAM. This new architecture follows the von Neumann paradigm: program and data are stored in the same read-write address space. vnTinyRAM can thus efficiently support many programming styles, including the C compiler of [10].

  - The circuit generator is universal: when given input bounds ℓ, n, T, it produces a circuit that can verify the execution of any program with $\leq \ell$ instructions, on any input of size $\leq n$, for $\leq T$ steps. In contrast, prior circuits generators hardcodes the program in the circuit.
  - The circuit generator efficiently handles large programs.

- A high-performance implementation of a zk-SNARK for arithmetic circuit satisfiability. This implementaion is an improvement upon the protocol of Parno et al. [70].

This is used in the construction of Zerocash [76]. Remark that the construction is based on pairing, and is not post-quantum secure.