

UNIVERSIDAD DE MURCIA

TRABAJO FIN DE MASTER

Privacidad en entornos de Blockchain

Autor:

José Luis
CÁNOVAS SÁNCHEZ

Tutores:

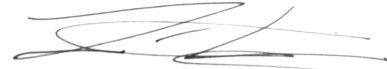
Antonio
SKARMETA GÓMEZ
Jorge
BERNAL BERNABÉ

July 9, 2018

Declaration of Authorship

I, José Luis Cánovas Sánchez, declare that this Master Thesis, titled PRIVACIDAD EN ENTORNOS DE BLOCKCHAIN and the work presented in it are my own, in the sense that I put the utmost effort in citing all sources.

Murcia, June 2018



José Luis Cánovas Sánchez

UNIVERSIDAD DE MURCIA

Abstract

Facultad de Informática
Department or School Name

Master en Nuevas Tecnologías en Informática

Privacidad en entornos de Blockchain

by José Luis Cánovas Sánchez

This paper presents a unified compendium of the foundations of the blockchain technology, with focus on the privacy issues related to it. The fundamentals on blockchain prepares the reader to understand how the many components work together to provide trust in a distributed untrusted network. A survey on current blockchain solutions presents the current state of the technology, evolved from the first blockchain, the Bitcoin. The fundamentals on cryptography focus on the mathematical foundations of the privacy solutions proposed for blockchain technologies. Finally, applications of blockchain with these privacy preserving tools are proposed.

Contents

Declaration of Authorship	ii
Abstract	iii
Introducción	1
1 Introduction	3
2 Fundamentals on Blockchain	5
2.1 Definitions	5
2.2 Transaction	8
2.3 Block	11
2.4 Consensus	14
2.4.1 Byzantine General's Problem	14
2.4.2 Proof-of-Work	15
2.4.3 Other Bizantine-fault tolerant algorithms	16
2.5 Privacy	17
3 Survey on current Blockchain technologies	18
3.1 Privacy preserving research proposals in blockchain	18
3.2 Self-sovereign identities in blockchain	21
4 Privacy preserving cryptography in blockchain	24
4.1 Zero-Knowledge Proofs	24
4.1.1 Complexity theory: P and NP Problems	24
4.1.2 Interactive Proof Systems	25
4.1.3 Zero-Knowledge Proof Systems	25
4.1.4 Commitment schemes	28
4.1.5 Anonymous Credential Systems	30
Camenisch-Lysyanskaya signature	30
Selective disclosure	31
Accumulator	32
4.1.6 Zero-Knowledge Proofs in Blockchain	32
Zerocoin	33
Sovrin	33
4.2 zkSNARK	34
4.2.1 Homomorphic encryption	36
4.2.2 Quadratic Span Programs	37
4.2.3 Evaluate a polynomial succinctly and zero-knowledge	38
4.2.4 A SNARK of the QSP Problem	40
4.2.5 Zero-Knowledge for the SNARK	42
4.2.6 Encoding a problem as a polynomial	42
4.2.7 Using zkSNARKs for shielded transactions	45

4.2.8	The applicability problems of zkSNARKs	46
4.2.9	STARKs	46
4.3	Ring Signatures	47
4.4	Comparison of technologies	49
5	Applications	51
5.1	Internet of Things	51
5.2	e-Government scenarios	54
6	Conclusions and future work	57
A	Survey Paper Draft	59
	Bibliography	82

List of Figures

2.1	Token-based transactions.	9
2.2	Bitcoin transaction example [35].	10
2.3	Bitcoin Public Key Hash generation[24].	11
2.4	ScriptSig structure [35]	12
2.5	Signing Message Template [35]	12
2.6	Block Header [67]	13
2.7	Merkle Tree [52]	14
2.8	Proof-of-Work over a block.	16
4.1	The cave story. Images by Adrián Sánchez Martínez.	27
4.2	Schnorr's protocol $ZPK\{(\alpha) : y = g^\alpha\}$. The Prover knows (g, q, y, α) such that $g^\alpha = y$. The Verifier knows (g, q, y)	28
4.3	Interactive CL-signature.	31
4.4	Sovrin interactions: issuance, presentation and trust relationship.	34
4.5	Elliptic curves addition.	36
4.6	Arithmetic circuit for $(c_1 \cdot c_2) \cdot (c_1 + c_3)$	43
4.7	Rivest, Shamir, Tauman ring signature scheme.	47
5.1	Envisaged blockchain-aware IoT ecosystem.	54
5.2	e-Government and education system overview	56

Introducción

En los últimos años, la tecnología blockchain ha ganado más atención por sus múltiples posibilidades y soluciones al problema del *trust* o confianza en escenarios descentralizados.

La tecnología blockchain fue descrita por primera vez en la creación de Bitcoin, por Satoshi Nakamoto [52], un alias para una persona, o grupo de personas, aún desconocidas. El objetivo era crear una moneda descentralizada, independiente de bancos y gobiernos. El problema principal a resolver era la confianza sin una autoridad central. Bitcoin fue el primero en resolver el problema de control del doble-gasto en moneda digital usando una red *peer-to-peer* (P2P). Un blockchain existe en una red de nodos, donde ningún nodo es de confianza. La confianza proviene de las bases criptográficas que componen el blockchain. Cualquier nodo puede verificar cualquier dato nuevo recibido en la red blockchain, y confiar en que es correcto, no por el nodo de origen, sino por los datos recibidos en sí mismos. Hay muchos otros sub-problemas relacionados con la moneda descentralizada, por ejemplo, cómo se representa a un usuario en la cadena, cómo se transfiere dinero, asegurar que la operación se origina del verdadero dueño, evitar el doble gasto de la misma moneda, acordar el orden global en que ocurren los eventos, etc. Todos estos problemas se resuelven con la tecnología blockchain.

Sin embargo, la tecnología blockchain no aplica solo a crypto-monedas, aunque dados sus orígenes, la aplicación más común es la de crear monedas alternativas. Modificaciones al diseño original permiten muchas otras aplicaciones donde se necesite confianza. Confiar en la tecnología, en una red de nodos en los que no confiamos.

Por ejemplo, un consorcio de empresas independientes puede rastrear las condiciones de los productos de cada compañía, y confiar, gracias a la inmutabilidad de blockchain, que ninguna compañía falsificará datos ya añadidos a la cadena. Por ejemplo, un barco pesquero puede poner a cada pieza una etiqueta RFID, de modo que a partir de ese momento se pueda detectar cuándo entra y sale de las cámaras frigoríficas, de los camiones de transporte, etc. Durante ese proceso, los datos de temperatura se pueden registrar en el blockchain también. Así, cuando una compañía recibe el producto de su proveedor, puede asegurar la calidad de las condiciones en que se guardó, y como esa compañía también registrará los datos, sus clientes también confiarán en que el producto está en óptimas condiciones. La tecnología blockchain permite este tipo de confianza sin una autoridad central.

Pero Bitcoin, y por tanto la tecnología blockchain, nunca se diseñó con la privacidad en mente. Muchas de las soluciones que aporta Bitcoin a la descentralización de la moneda dependen de que todos los nodos tengan la misma información. Esto implica que cualquier transacción en Bitcoin es pública, en el sentido de que cualquier nodo en la red puede ver toda su información (excepto claves privadas), correlando el origen y destino de la moneda, siguiendo todo su histórico. Aún más, blockchain es un registro permanente de datos, redundante en todos los nodos de la red.

Hablando sobre la privacidad, debemos mencionar las existentes regulaciones,

como la *General Data Protection Regulation* (GDPR), que da a los ciudadanos el derecho de rectificar sus datos, borrarlos o el derecho al olvido. En este sentido, la existencia de regulaciones puede entrar en conflicto con la tecnología blockchain, pues la cadena debe ser inmutable, persistente y distribuida. Por ejemplo, si un usuario registra sus datos personales en una cadena, debe ser consciente de que puede que nunca sea capaz de ejercer sus derechos establecidos por la GDPR. Por tanto, se precisa de especial cuidado para evitar guardar datos personales en el blockchain, incluso si están cifrados.

Nuevos proyectos de blockchain están aplicando soluciones para preservar la privacidad, intentando mantener la propiedad de *confianza* que caracteriza al blockchain, mientras permiten anonimidad y privacidad de los datos de los usuarios. Muchas de estas tecnologías tienen aplicaciones limitadas debido al alto coste computacional, o la necesidad de una inicialización donde se confíe en ciertos nodos, lo que viola el principio de que ningún nodo de la red es de confianza.

Dada la novedad del tema, la información en este área es dispersa, y los principios matemáticos se referencian a los artículos originales, enfocados a expertos en cada área, haciendo difícil que los nuevos investigadores entiendan cómo se aplican al blockchain. El propósito de este trabajo es dar un documento de referencia para las soluciones de privacidad en blockchain, enfocándose en sus fundamentos matemáticos, realizando una comparación entre ellos y estudiando su viabilidad para hacer frente a nuevos escenarios que puedan aprovecharse de la tecnología blockchain, como Internet de las Cosas o escenarios de administración electrónica de gobiernos.

Este documento primero presenta las bases del blockchain, en el capítulo 2, explicando los componentes clave que son comunes a todos los diseños de blockchain. El capítulo 3 muestra un estudio de los blockchain alternativos propuestos en la investigación sobre la privacidad. El capítulo 4 ofrece un análisis en profundidad de las distintas soluciones de privacidad aplicadas en blockchain, con los principios matemáticos de cada alternativa, con el objetivo de que el lector sea capaz de juzgar las ventajas e inconvenientes, aplicabilidad y grados de privacidad que cada solución puede aportar. El capítulo 5 presenta propuestas de futuras aplicaciones de estas tecnologías. Finalmente, el capítulo 6 resume las conclusiones de este estudio.

Chapter 1

Introduction

In recent years, the blockchain technology has gained more attraction for its many possibilities and solutions to trust in decentralized scenarios.

The blockchain technology was first described in the creation of Bitcoin, by Satoshi Nakamoto [52], an alias for a still unknown person or group of people. The objective was to create a decentralized currency, independent of banks or governments. The main problem to solve was the **trust** without a central authority. Bitcoin was the first to solve the double-spending problem for digital currency using a peer-to-peer network. A blockchain exists within the network nodes that comprise it, and no network node is trusted. The trust comes from the cryptography foundations of the blockchain data. Any node can verify any new blockchain data received from the network, and trust that it is correct, not because of the origin node, but because of the data itself. There are many sub-problems related to a decentralized cryptocurrency, such as how a user is represented, how money is transferred, secure that the operation originates from the real owner, avoid double spending of the same coin, agree on the order of events, etc. All these problems were solved with blockchain.

Nonetheless, the blockchain technology does not only apply to cryptocurrencies, although given its origin, the most common application is for alternative currencies. Modifications to the original design allow for many other applications, where the key characteristic is the trust. Trust the technology in a network of untrusted nodes.

For example, a consortium of independent companies, can track each product conditions in each company, and trust, thanks to the immutability of blockchain, that no company can forge provenance data already in the chain. For example, a fishing boat could tag each piece with a cheap RFID tag. From there on, the freezers, transport trucks, etc. can register when the product came in, the temperature conditions, when it was delivered to the next company, etc. The consortium gains trust in the providers and the clients are able to verify the good quality of the product. The blockchain technology allows this kind of trust without central authorities.

But Bitcoin, and therefore the blockchain technology, was never designed with privacy in mind. Many of the solutions that Bitcoin applies to the decentralized currency rely on all nodes having all the same information. This implies that any transaction in Bitcoin is public, in the sense that any other node in the network can see all its information (except private keys), correlating the source of the coin and follow its new history. Furthermore, blockchain is a permanent registry of data, redundantly stored in every node of the network.

Talking about users and privacy, we should mention the existence of regulations such as General Data Protection Regulation (GDPR), which empowers citizens with the rights to have their data rectified, erased or forgotten. In this sense, the existing regulations may come into conflict with Blockchain technology in the way of when you keep something in the chain it is supposed to be immutable, persistent and unmodifiable. For example, if a user registers his personal data in a blockchain

solution, he must be aware that maybe he will not be able to exercise all his rights collected by the GPDR. Therefore, special care needs to be taken to avoid pushing personal data in the blockchain, even if encrypted.

New blockchain projects are applying privacy preserving solutions to blockchain, trying to keep the trust property, while allowing anonymity and privacy of the user's information. Many of these technologies have limited applicability due to high computational requirements, or the need of trusted setups that violate the principle of not trusting the nodes of the network.

Given the novelty of the subject, the information on this area is scattered, and the mathematical principles are dispersed in their original papers, aimed for experts in each subject, making it difficult to understand for many new-coming researchers how they are applied to blockchain. The purpose of this document is to provide a single reference document for these privacy preserving solutions in blockchain, giving the mathematical foundations, performing a comparison of them, and studying their suitability to cope with prominent scenarios that can exploit blockchain features, such as Internet of Things, and e-government/public administration scenarios.

This document first presents the fundamentals of blockchain in Chapter 2, giving an insight of the key components that are common to all blockchains. Then, Chapter 3 describes a survey of many blockchain alternatives with proposals for the privacy issues. Chapter 4 offers a deep analysis of the privacy preserving solutions applied to blockchain, with the mathematical grounds of each alternative, with the objective of being able to judge the advantages and disadvantages, applicability, and degree of privacy acquired with each solution. Chapter 5 briefly presents proposals of applications of these technologies. Finally, Chapter 6 sums up the conclusions from this study.

Chapter 2

Fundamentals on Blockchain

This chapter focuses on the basic definitions and functionality of blockchains. For each of the main components there exist different solutions and specific blockchain deployments that make use of them.

2.1 Definitions

Blockchain shifts trust from a classical centralized approach to a fully decentralized network of nodes. It is based on a synchronized Distributed Ledger Technology (DLT), which acts as a decentralized database, keeping the information replicated and shared among multiples nodes spread in remote locations.

The concept of blockchain was first introduced by Satoshi Nakamoto in [52] as the technical foundations of a new peer-to-peer version of electronic cash. The inspiration for a *cryptocurrency* comes from Wei Dai in 1998, and is the base of the Bitcoin work, to solve the problems related to a new currency. The Bitcoin paper does not actually mention the word *blockchain* in it, but it describes all its parts, as transactions, proof-of-work, the network, incentives, etc.

The research that followed Bitcoin, more interested in the technical foundations than the cryptocurrency, proposed many variations to how a blockchain may work, but helped to distinguish what makes a blockchain. From those common concepts, here we present some definitions of *blockchain*:

Gattschi et al. [26] briefly defined the blockchain as “a public ledger distributed over a network that records transactions (messages sent from one network node to another) executed among network participants. Each transaction is verified by network nodes according to a majority consensus mechanism before being added to the blockchain. Recorded information cannot be changed or erased and the history of each transaction can be re-created at any time.”

Crosby et al. [20] defined the blockchain as a “distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system. Once entered, information can never be erased. The blockchain contains a certain and verifiable record of every single transaction ever made.”

The Decode project presents a more technical description of a blockchain [1]:

The building block of blockchain is the hash pointer. A hash pointer is simply some information (typically called transaction) along with a hash of the information. The hash serves to identify the information, and also allows verification of its integrity. A blockchain is a linked list of hash pointers, where usual pointers

have been replaced with hash pointers. The first block in the chain points to a special block called the genesis block. Each block contains a hash of the previous block and information specific to the current block. A key result of iterative hashing is that a block implicitly verifies integrity of the entire blockchain before it. Thus given the current head of the blockchain, any party can independently verify the entire blockchain by generating hashes from the beginning of the chain up until the end. The final hash should match that in the head, otherwise the blockchain has been tampered with.

All definitions agree on this collection of concepts that shape what a blockchain is:

- *Transaction*: A single record in the ledger that can specify a piece of information or an operation over previous transactions, e.g. to send the funds from a previous transaction to another public address.
- *Block*: A group of transactions, establishing a chronological order between them. A block also includes a hash pointer to the previous block of the blockchain.
- *Hash pointer*: Given a transaction or a block their hash is included in the blockchain, which gives integrity and a means to point to the transaction or block, like a memory pointer to a variable. When a new block of transactions is added to the chain, the hash pointer of the previous block is appended. To compute the hash pointer of the new block, the hash pointer of the previous one must be included in the calculation, increasing the integrity of all the previous blocks.
- *Genesis block*: The first block in a blockchain, establishing a starting point for the linked list of hash pointers.
- *Chain*: The linked list of hash pointers from the genesis block to the last block. The chain determines the chronological order of all transactions, as well as integrity of the entire blockchain before it. One can verify the integrity of the chain by computing the hashes from the genesis block to the last one, if any hash pointer to the previous block differs from the one computed, the chain has been altered.
- *Merkle Tree* [48]: A binary tree where the leaves are the hash pointers of the transactions in a block, and each parent node is the hash of the two children nodes. The root of the Merkle tree is a hash that gives integrity to all transactions in a block, including their order within it. The complete block's hash pointer is the hash of the Merkle tree root, with the hash pointer of the previous block and any consensus information that makes the node valid. When verifying integrity of the chain, the Merkle tree allows to compute the valid hash of the block, without having the entire transaction information on disk. In the original Bitcoin paper, the Merkle tree is used to reclaim disk space from old spent transactions (which in theory are not used again), but the Merkle tree also allows to give proof of existence for a transaction in the blockchain without including in the proof the rest of transactions in the block.
- *Network*: Referring to the *blockchain network*, it is the set of nodes that interact between them in a peer-to-peer (P2P) fashion, exchanging the blockchain data, adding transactions, validating them, and agreeing on what new blocks are added to the head of the chain.

- *Consensus*: The algorithm run by the nodes of the network to agree on the state of the blockchain. The set of all transactions, in order, defines this state. When a new transaction is added, the state changes. Because of possible delays transmitting the new transaction to all nodes, the order transactions arrive to a node may differ with respect to other node. Because there is not a central authority node to decide which transaction arrived before, the consensus algorithm is run by the network and can be verified by any node, shifting the trust from a traditional central authority to the verification cryptographic methods.
 - *Proof of Work*: The first Consensus algorithm, applied in Bitcoin. The right to cast a vote is equivalent to the ammount of CPU work used.
 - *Proof of Stake*: Alternative to Proof of Work with the aim to reduce its great cost. The right to cast a vote is equivalent to the ammount of tokens put on stake.
 - *Practical byzantine fault tolerance (PBFT)*: Consensus protocol that replicates itself multiple times to avoid faulty nodes. Each nodes transmit its vote to a designated primary node, which is different in each replica. Used mostly in private blockchains.
- *Fork*: Depending on the consensus algorithm, the network may accept two blocks at the same height of the chain. This creates a fork of the chain. Part of the network may continue to add blocks using one of the forks, while the other part uses the other fork. Because the hash pointers of each block will differ, the chains are incompatible, the network must agree on which fork to use. In Bitcoin, the longest fork is the valid one. This solves the problem of malicious nodes creating a fork before spending some funds, to regain them. During the consensus algorithm, a new shorter fork is rejected.
- *Script*: Piece of code embedded in a transaction, based on a limited programming language, that establishes the conditions to validate a transaction. E.g., in Bitcoin, the script allows to differ a payment to a given date, or until more signatures from other nodes are present in the chain.
- *Smart Contract*: Evolution of the script language, usually Turing complete, but deterministic, that allows to shift from a static transaction to an execution of code. A transaction that is result of executing a smart contract can be verified by any other node by executing the same smart contract with the same inputs. E.g., the blockchain Ethereum defines the Solidity language, and the Ethereum Virtual Machine (EVM) where the compiled bytecode is executed.
- *Mining*: The process where a blockchain node validates new transactions (running the script or smart contract) and executes the consensus algorithm. The name originates from the Proof of Work consensus algorithm, giving the high CPU consumption it requires to run.

Because blockchain technology was created to support the Bitcoin cryptocurrency, the widest use of blockchain is to create alternative cryptocurrencies. The key issue blockchain solves is known as the *double spending* problem. Without a central entity, a network of untrusted peers must agree on valid transactions (consensus), controlling that no malicious peer spends twice the same funds. Bitcoin solves it by making all transactions public on the ledger, so any node can keep track of the spent transactions.

From the point of view of **blockchain types**, there are three different architectures:

- *Public Permissionless Blockchain*, where anyone can read, write, participate in consensus and where transactions are transparent but with participants with some anonymity or pseudo-anonymity.
- *Private Permissioned Blockchain*, where the participating nodes must be granted access to the network, via an invitation or *permission*, in order to perform operations over the distributed ledger or participate in consensus. The access control mechanism could vary: existing participants could decide future entrants; a regulatory authority could issue licenses for participation; or a consortium could make the decisions instead [32].
- *Public Permissioned Blockchain*, a concept introduced by the Sovrin Foundation [65], identifies those blockchain instances that are open for all to use, that is, read or change the state of the ledger, but the network of nodes performing consensus is permissioned.

In the following sections we will describe each component of the blockchain.

2.2 Transaction

A transaction is any piece of information meaningful to the chain. A transaction can store a sensor's data for immutable log management, a public certificate [25], a program code, or more usually, it defines an operation over one or more previous transactions.

Because blockchain originates from Bitcoin, most of the blockchain alternatives follow the same structure, although they may apply any arbitrary change based on their own design decisions. Therefore, there doesn't exist a single definition of a transaction.

Nonetheless, this section explains how Bitcoin's transactions work, showcasing how they operate over previous transactions and what makes a transaction valid, the two key points that are present in any other blockchain's transaction.

In transactions that represent ownership of a token, such as cryptocurrency, each coin would be represented as a chain of digital signatures, stored in one transaction each. The digital signature chain function is, on the one hand, establish the current ownership of the coin, and on the other hand, track the history of the coin to avoid double spending by one of the previous owners, who after spending it once, should not be able to sign it again.

To transfer the coin to a new owner, a new transaction is created, with a signature of the previous transaction's hash and the public key of the payee. The previous transaction refers to the last transaction of the coin being transferred, not the last transaction in the global blockchain. Figure 2.1 depicts the token's ownership chain of digital signatures.

Therefore, when a new transaction arrives, to assure that the token has not already been spent by the signer, and to avoid the need of a centralized trusted party, Bitcoin's solution was to make all transactions public. With consensus, as explained in the following sections, all nodes agree on a single history of the order in which transactions were received. A node would keep track of the unspent transaction outputs (UTXO) with a faster database, created using the blockchain's public transactions information.

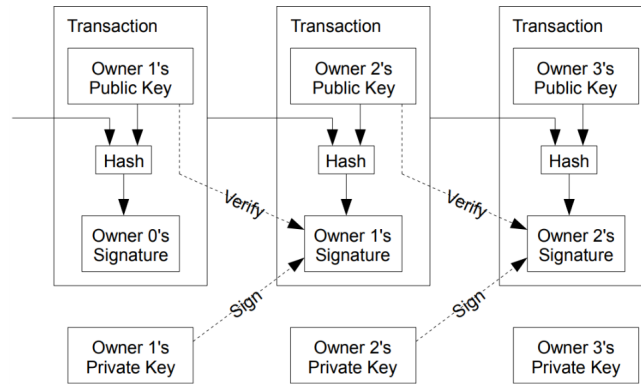


FIGURE 2.1: Token-based transactions.

But before a transaction is considered for consensus, it must be validated. The validation rules applied by the nodes depend on the particular blockchain. The nodes participating of a blockchain agree to those rules, as otherwise, the majority in consensus will reject their transactions. Bitcoin made the validation somewhat dynamic by creating a limited Script language, stack memory based, which is embedded in each transaction and defines the conditions to spend the tokens.

A new transaction will refer as an input one or more UTXO, and produce one or more of them. The new UTXO created will incorporate a locking script. The locking script imposes a condition that must be met before the UTXO can be spent. If this UTXO is subsequently used as the input in a future transaction, the input segment of this future transaction must provide the unlocking script that will make the UTXO spendable, and consequently the transaction itself a valid transaction [35]. Therefore, each transaction has two Script segments, one to unlock the inputs, and one to lock the outputs. These two Scripts aren't related between them, but with those of the previous and next transactions, respectively.

The most common type of Script is the P2PKH (Pay to Public Key Hash), which refers to locking a new UTXO to be spendable only if: the transaction that consumes it can provide a Script that proves possession of the private key related to the public key hash stored in the locking Script.

Figure 2.2 shows the structure of a Bitcoin transaction, highlighting the input segment, which is the information about the previous transaction of the coin, i.e., the transaction to spend, and the output segment, which is the information of the new owner of the coin.

As shown in 2.3, Bitcoin uses Elliptic Curve Cryptography to generate private and public keys, SHA256 and RIPE160 to shorten the public key, and Base58 encoding to publish it inside a transaction. The value of the shortened hash is the Public Key Hash a P2PKH refers to.

The input segment includes the hash pointer to the previous transaction, where the locking script is defined. Then, the ScriptSig is the unlocking script, stored as the bytecode of the Script language. This Script consists of two operations, the first one pushes into the stack memory the ECC signature of the current transaction (except for the ScriptSig field, as we are generating it now) in DER format, and the second operation also pushes to the stack the public key of the previous signature. Figure 2.4 showcases this structure, and Figure 2.5 depicts the message signed, which is the current transaction, but in the place of the ScriptSig field, the ScriptPubKey field is replicated.

The output segment consists of another Script bytecode. In the case of a P2PKH,

Version	01 00 00 00
Number of Inputs	01
Previous Tx Hash (reversed)	41 6e 9b 45 55 18 0a aa 0c 41 70 67 a4 66 07 bc 58 c9 6f 01 31 b2 f4 1f 7d 0f b6 65 ea b0 3a 7e
Previous Output Index	00 00 00 00
Script Length	6a
ScriptSig (Unlocking script)	47 30 44 02 20 1c 3b e7 1e 17 94 62 1c be 3a 7a de c1 af 25 f8 18 f2 38 f5 79 6d 47 15 21 37 eb a7 10 f2 17 4a 02 20 4f 8f e6 67 b6 96 e3 00 12 ef 4e 56 ac 96 af b8 30 bd df fe e3 b1 5d 2e 47 40 66 ab 3a a3 9b ad 01 21 03 bf 35 0d 28 21 37 51 58 a6 08 b5 1e 3e 89 8e 50 7f e4 7f 2d 2e 8c 77 4d e4 a9 a7 ed ec f7 4e da
Sequence	ff ff ff ff
Number of Outputs	01
Value	20 4e 00 00 00 00 00 00
Script Length	19
ScriptPubKey (Locking Script)	76 a9 14 e8 1d 74 2e 2c 3c 7a cd 4c 29 de 09 0f c2 c4 d4 12 0b 2b f8 88 ac 00 00 00 00
Locktime	00 00 00 00

Magenta: Input Segment

Yellow: Output Segment

FIGURE 2.2: Bitcoin transaction example [35]

this code describes that the rule to spend the coin is to be in possession of the private key related to an specific public key hash. The operations consist on duplicating the current top of the stack, perform the SHA256 and RIPE160 on that value, push to the stack the public key hash of the coin recipient, compare the two top values of the stack, and finally, the most important one, to verify the signature of the current transaction with the values in the stack. This Script includes only one value, the recipient public key hash, and the set of instructions that allow to verify the owner. Specifically, the comparison step checks that the creator of the transaction knows the original public key, from where the hash is created; and the verify signature step checks that the creator of the transaction knows the private key, aside from integrity of the transaction values. These two steps combined authenticate the owner of the coin and authorize to spend it.

Now we know how a new transaction is created: refer the previous transactions as input, the public key hashes as output addresses, sign the transaction to proof ownership of the secret key.

The next step that all nodes will perform when receiving the new transaction, is to **validate** it. Validation consists on executing first the ScriptPubKey code (unlocking code), which in the P2PKH case is to store the signature and public key to the stack, and secondly, leaving the stack memory as is, execute the ScriptSig code (locking code), which, as described above, will check the signature and public key currently in the stack memory. Only if the execution is correct, a True value will be left in the stack, and the node will assume the transaction is valid, in any other case, the transaction is rejected. The order is important, as the node will only check the stack memory. The unlocking code goes first, preparing the stack with any needed proof of being the owner of the coin, and then the locking code is executed, giving the previous owner the right to decide if the proof is valid.

Improving the idea of a programming language to define the validation of a transaction, other blockchains define a Turing complete programming language. The most prominent is Ethereum, which gives the specification for the EVM, Ethereum

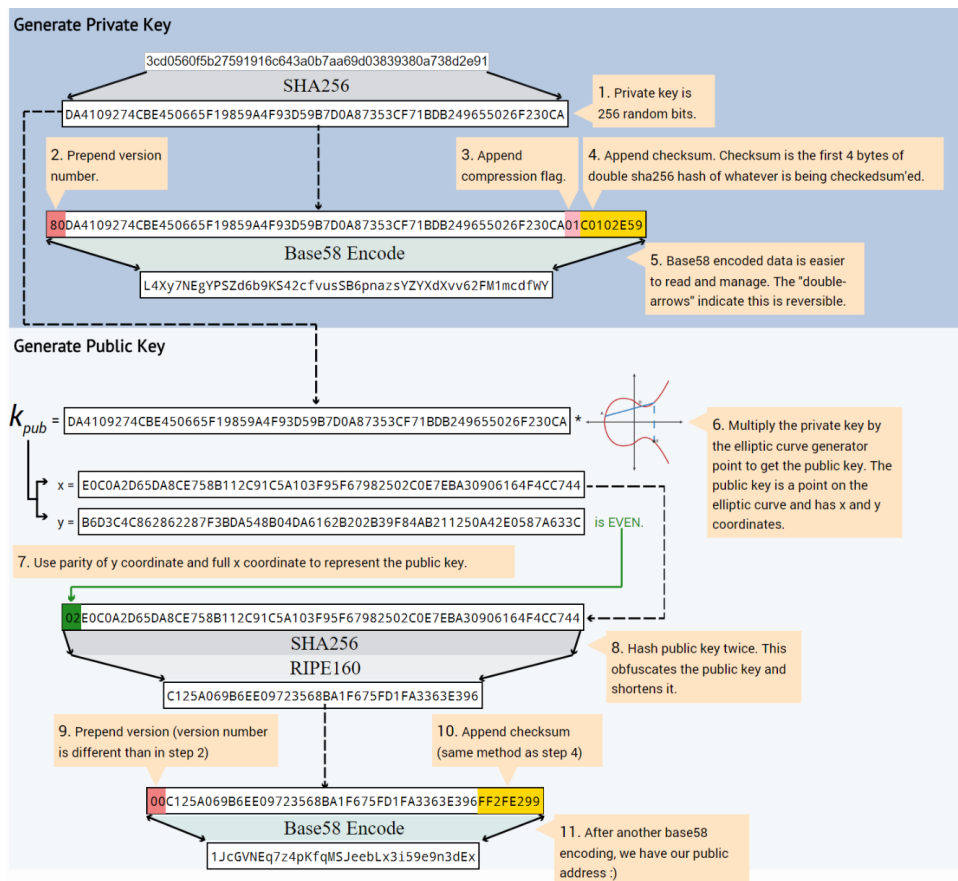


FIGURE 2.3: Bitcoin Public Key Hash generation[24].

Virtual Machine, where every node executes the so called Smart Contracts. Simplifying, Ethereum describes two types of assets, one is a token, like Bitcoin, called gas, and the other one is the Smart Contract, bytecode for the EVM. To run a Smart Contract, a new transaction transfers gas ownership to the Smart Contract. The amount of gas depends on the amount of operations in the code. Now, not only a client with public and private keys can own tokens, but also the Smart Contracts, which can also send it to a new public address, if it is programmed to do so, like an opening Script in Bitcoin.

2.3 Block

The block is the keystone to the blockchain technology, alongside hash pointers. A block is a data structure that groups multiple transactions in it, establishing an order between them. The block is a necessary construction to improve the blockchain performance. The main problem about creating new transactions is to agree on the order of creation. Given network delays, topology, etc., not every node will receive new transactions at the same time or order. The next section will explain how with Consensus, the network agrees on a unique order for all transactions. For now, the most notable characteristic to know is that consensus is expensive. Instead of executing consensus on every single transaction, it is applied to a group of new transactions, a block, so the consensus algorithm agrees on the order of small chunks of new transactions, multiplying the protocol efficiency.

PUSHDATA Opcode		47
Signature (DER encoded)	Header	30
	Sig Length	44
	Integer	02
	R Length	20
	R	1c 3b e7 1e 17 94 62 1c be 3a 7a de c1 af 25 f8 18 f2 38 f5 79 6d 47 15 21 37 eb a7 10 f2 17 4a
	Integer	02
	S Length	20
	S	4f 8f e6 67 b6 96 e3 00 12 ef 4e 56 ac 96 af b8 30 bd df fe e3 b1 5d 2e 47 40 66 ab 3a a3 9b ad
SigHash Code		01
PUSHDATA Opcode		21
Public Key (compressed with 03 prefix)		03 bf 35 0d 28 21 37 51 58 a6 08 b5 1e 3e 89 8e 50 7f e4 7f 2d 2e 8c 77 4d e4 a9 a7 ed ec f7 4e da

FIGURE 2.4: ScriptSig structure [35]

Version	01 00 00 00
Number of Inputs	01
Previous Tx Hash (reversed)	41 6e 9b 45 55 18 0a aa 0c 41 70 67 a4 66 07 bc 58 c9 6f 01 31 b2 f4 1f 7d 0f b6 65 ea b0 3a 7e
Previous Output Index	00 00 00 00
Script Length	19
ScriptPubKey of Previous Output (placeholder for final scriptSig)	76 a9 14 99 b1 eb cf c1 1a 13 df 51 61 ab a8 16 04 60 fe 16 01 d5 41 88 ac
Sequence	ff ff ff ff
Number of Outputs	01
Value	20 4e 00 00 00 00 00 00
Script Length	19
ScriptPubKey (Locking Script)	76 a9 14 e8 1d 74 2e 2c 3c 7a cd 4c 29 de 09 0f c2 c4 d4 12 0b 2b f8 88 ac
Locktime	00 00 00 00
SigHash Code	01 00 00 00

FIGURE 2.5: Signing Message Template [35]

The first block in the chain is called the Genesis Block, and bootstraps any verification process, as the trust in this special block being the right one is what defines the trust in the integrity of the rest of the chain.

Again, depending on each particular blockchain re-design, a block structure may vary, but the majority are based on the original Bitcoin block. The block is made-up of the block header and the transactions in order. The previous section showcased the byte structure of a transaction. Figure 2.6 showcases the byte structure of the block header.

The block header structure has the following fields:

- *Magic number* (4 bytes): a constant value of 0xD9B4BEF9 that identifies the Bitcoin blockchain. Other value may identify a developing chain, with a custom genesis block, or a parallel instantiation of the blockchain.
- *Block size* (4 bytes): size in bytes of the entire block. Bitcoin is currently limited to 1MB in size. Other blockchains, like Bitcoin Cash, expanded it to 32MB, allowing to increase not only the amount of transactions in a block, but also the Script operations admitted inside the transactions, for example.

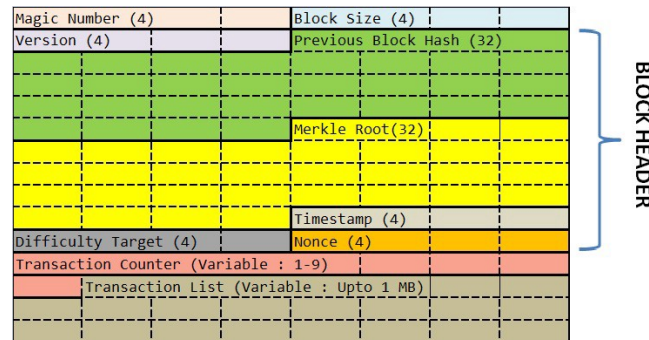


FIGURE 2.6: Block Header [67]

- *Version* (4 bytes): the protocol version used to generate the transaction. When the blockchain protocol is modified, all nodes update their code and tag the new transactions with the new version.
- *Previous Block Hash Pointer* (32 bytes): this is the hash pointer of the previous block in the chain.
- *Merkle Tree Root* (32 bytes): as explained below, this is a hash value of the transactions inside the block.
- *Timestamp* (4 bytes), *Difficulty Target* (4 bytes), *Nonce* (4 bytes): these three values relate to the Consensus algorithm, explained in the next section.

After the header, the *Transaction Counter* and the *Transaction List* are the transactions themselves, occupying up to the 1MB limit of Bitcoin's blocks.

To generate the **hash pointer** of the block, instead of appending all transactions with the header for the hashing function, i.e. using the entire block, Bitcoin[52] decided to use Merkle Trees, and the hash pointer is generated only from the block header. Although, the original objective of Merkle Trees was to reclaim free disk space from spent transactions, without losing the capability of verifying the chain integrity.

A Merkle Tree is usually a binary tree where the leaves are the hashes of the transactions in the block, in order. The parents of each pair of hashes, i.e. tree nodes, is the hash of the two children. The root of the Merkle Tree is a hash that gives integrity to all the block's transactions in order.

To verify that a transaction belongs to a block, a node only needs the transaction and part of the Merkle Tree. It performs the hash of the given transaction, and $\log_2(n)$ more hashes to reach the root hash, using the neighbour tree nodes in each step. Figure 2.7 shows this process, and how it helps to keep integrity of the chain after removing the rest of the transactions from disk.

A derived application of the Merkle Trees in blockchain allows for mobile devices to verify a transaction without the need to download from a peer the entire blockchain. The mobile device, with a wallet app, may ask for the blocks headers, the transaction it is interested in, and the partial Merkle Tree of the transaction's block. This way, the wallet can verify the list of hash pointers of the chain, and verify that the transaction really exists within a block in the chain. The device stores the minimum amount of data, and has proof of a single transaction to exist in the chain. This type of verification is weaker than verifying the blockchain from the genesis block, and relies on connecting to honest nodes.

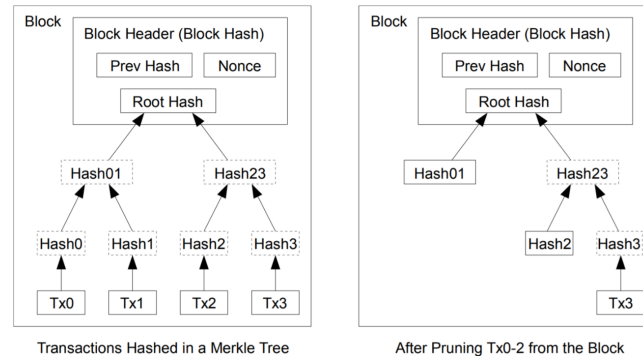


FIGURE 2.7: Merkle Tree [52]

2.4 Consensus

In order for the blockchain network to agree on the order of transactions, grouped in blocks, the consensus protocol is executed.

From the Bitcoin paper [52]: *To accomplish this without a trusted party, transactions must be publicly announced, and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.*

This is the necessity for a consensus algorithm, to agree that the majority, in some way, are following the same rules or order of the chain.

First, let's introduce the Byzantine's General Problem, which is the problem consensus must solve:

2.4.1 Byzantine General's Problem

The Byzantine Generals' Problem is an agreement problem described by Leslie Lamport, Robert Shostak and Marshall Pease in 1982 [39]. The set is a group of seven generals, each one with its own troops of the Byzantine army, surrounding a city. The options are to attack or retreat. The generals know that if they attack the city, unless the seven of them attack at the same time, they will lose the battle. If some retreat, but others attack, the city's defenses will defeat the attackers, harming the Byzantine's military forces.

The solution is for the seven generals to send messengers with their vote to the other generals. But if there are treacherous generals among them, they could selectively send different votes to the generals. For example, three generals vote to attack, three to retreat, and the traitor has the decisive vote. If the traitor sends message to attack to the first three, and to retreat to the others, the generals will not attack at the same time and lose the battle. Some other complications refer to possible messengers getting lost, or forging false votes.

This problem is mapped onto computer networks in the sense that each node is a general, the network messages are the messengers, and the optimal condition is a coordinated action of the network.

Byzantine Fault Tolerance can be achieved if the loyal generals have a majority agreement on their strategy.

Usually, to describe a Byzantine fault tolerant protocol, it specifies a function of how many nodes it needs to handle at most f faulty nodes. For example, the Redundant Byzantine Fault Tolerance (RBFT) protocol used in Hyperledger Indy needs $3f + 1$ nodes to handle f faulty nodes [4].

2.4.2 Proof-of-Work

The first consensus algorithm, described in the Bitcoin paper [52]. The original objective was to implement a distributed timestamp server on a peer-to-peer basis. This translates into giving an order to the transactions, defining the same state of the blockchain for all peers in the network.

The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash [52].

In other words, assuming the random oracle model, in the sense that in practice we cannot predict the value of a hash function, we search a random value whose hash is below a certain *difficulty* value. Verifying that hash involves only one application of the function, but finding the random value is statistically exponential in time. The name Proof-of-Work makes reference to the CPU work the machine does in order to find the random value with valid a hash. This task is commonly known as *mining*, and the blockchain node that performs the PoW is called a *miner*.

To link a PoW with a new block, the hash function takes as input not only the random value, called *nonce*, but also the block header, which includes the hash pointer of the previous block in the chain and the Merkle tree root, therefore, linking all transactions of the new block to the PoW. This Proof-of-Work hash becomes the block's hash pointer, to be used by the next block to point to the previous one in the chain.

The *difficulty* (i.e. the number of bits starting with 0 in the hash to be generated) is recalculated every 2016 blocks, making the mining process (finding the nonce that can generate such a hash) harder or easier. The objective is to keep as a constant the mean of time it costs to add a new block to the chain. In Bitcoin this mean time is 10 minutes per new block.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote [52].

The longest chain of blocks with valid PoW nonces represents the effort invested in that chain. Therefore, when a fork appears in the chain, it is resolved when one of the forks grows longer than the other.

After several new blocks are added to the chain, a given block is protected by the hash pointers of its successors and the difficulty of the PoW. If an attacker were to change a past block to modify any spent transaction, it would have to redo all the PoW of the modified block, and the following ones, as the hash pointers would change. The attacker would have to compete with the computational power of the rest of the network to create PoW faster than them, to create the longest chain. If the difficulty value is chosen in order for the PoW to be exponential in time, the probability of an attacker catching up with the rest of the network also drops exponentially.

The so called *51% attack* refers to a PoW attack where an attacker would statistically be successful if it owns more than half the resources of the network, therefore, being able to catch up with the computational resources of the rest and creating a longer chain.

In the particular case of Bitcoin, but also mimicked by other cryptocurrencies, the incentive is key to the honest operation of the chain. The mining process involves a great amount of computational work. Any node may be interested in mining a fork



FIGURE 2.8: Proof-of-Work over a block.

chain that benefits itself, and if many nodes do it, the honest nodes computational power would decrease to the point that an attacker may actually succeed. Bitcoin incentives honest behaviour by rewarding a successful miner with new tokens.

2.4.3 Other Bizantine-fault tolerant algorithms

Aside from the Proof-of-Work Consensus protocol, there exist other proposals for consensus. Depending on the type of blockchain (public, private, permissioned or permissionless), and the requirements for *democracy* or speed needed, there will be a different consensus protocol to apply.

The public blockchain, uses primarily Proof-of-Work, although the second most popular option is known as Proof of Stake (PoS), where the idea is that a mining node puts some amount of coin at stake, the more coin, the more chances to mine the next block and retrieve the coin at stake, but if a node is detected to be cheating, the network will take the bet coin, dissuading malicious nodes. Other popular consensus protocols follow the same principles as PoW or PoS, the miner must give something to gain rights to vote. For example, there are Proof of Elapsed Time (PoET), Proof of Existence (PoE), Delegated Proof of Stake (DPoS), Proof of Activity (hybrid of proof of work and proof of stake), Proof of Importance, Proof of Storage, etc. [61].

For scalability, permissioned blockchains like Hyperledger Fabric distinguish two types of nodes, validating nodes and non-validating nodes. The first set are the authorized nodes to perform the consensus protocol, the second set includes proxy nodes that communicate clients with validating nodes, and although they cannot execute the consensus protocol, they can still validate transactions to help detect malicious behaviour. The validating nodes execute the Practical Byzantine Fault Tolerance protocol (PBFT) [18], which provides high-performance Byzantine state machine replication, processing thousands of requests per second with sub-millisecond increases in latency. PBFT needs $3f + 1$ nodes to tolerate f faulty or malicious ones. The general idea is that all verification nodes execute multiple instances of the same state machine, but in each instance, only one node is known as *primary* node. Given an instance, when a client sends a request to the primary node, it broadcasts it to the rest of nodes, which respond validating the primary's order. When $f + 1$ replies accept, the protocol has succeeded. The parallel instances are used as a safety-net in case that the *principal* one fails, meaning that there may be a malfunction of that consensus replica execution or that primary node is a malicious node. Several other Byzantine fault tolerance protocols are based on PBFT.

2.5 Privacy

When Bitcoin defined the foundations of blockchain, it addressed privacy as anonymity of the public keys. In the blockchain, to solve the double spending problem, all nodes must have access to all transactions, following their history. In comparison, a trusted third party as a bank hides this information to everyone except the sender, receiver and itself. But the bank requests a real identity being provided, and the cryptographic public keys in blockchain don't. One creates a new address without attaching any Personally Identifiable Information (PII) to it, and it would be an anonymous address.

Nonetheless, this method does not avoid many other correlations. One example is the IP address of the machine creating the transaction, which can be logged by other P2P nodes. Another case is when using multiple inputs in a single transaction, that shows that all input addresses were linked to the same owner.

To sum up, the main privacy issues of Bitcoin are:

- Multi-entry transactions: this kind of transaction require having different addresses belonging to the same user in the same transaction, making them linkable [31]. There should be a different address for every received transaction of digital assets.
- Transactions with change: it allows to trace the user when he employs the same address to get the change. There should be different address to get the change of the transaction.
- Curious/Malicious Mixing services: outsourced and centralized services can be adopted by users to improve their privacy by mixing transactions. However, it can become a privacy issue as the service might know both the input and output addresses pairs.
- Non erasable Data: privacy is also about the right to erasure data, but the blockchain is immutable. Therefore, personal data should not be stored in blockchain. In this regard, [38] presents a block matrix data structure for Integrity Protection with Erasure Capability. It allows continuous addition of hash-linked records, enabling at the same time deletion of records.

Chapter 4 focuses in the study of the privacy-preserving cryptographic methods to solve the problems that Bitcoin didn't address. But to decide what are the most relevant cryptographic methods, Chapter 3 analyzes a set of blockchain solutions aimed for privacy preservance.

Chapter 3

Survey on current Blockchain technologies

This chapter presents the different blockchain technologies evolved from Bitcoin, with focus on their relation to privacy. This analysis will help to identify the key cryptographic solutions to privacy.

3.1 Privacy preserving research proposals in blockchain

Recalling from Chapter 2, in Bitcoin a transaction between a source payer address A_{Ps} and target payee A_{Pt} can be defined as $T(A_{Ps} \rightarrow A_{Pt}) = (\tau, B, A_{Pt}, \text{Sig}_{sk_{A_{Ps}}}(\tau, B, A_{Pt}))$. Where the payer A_{Ps} signs with its secret key $sk_{A_{Ps}}$ the amount B of Bitcoins, including a reference to the most recent previous transaction τ in which A_{Ps} acquired the amount B of Bitcoins. Afterwards, when the bitcoin transaction T is confirmed, A_{Pt} can use this transaction as reference to spend the acquired Bitcoins. The Bitcoins transactions set up a public record that can be verified by anyone, by checking the signatures of the chain. The Bitcoins addresses are pseudonyms that are mapped to the ECDSA public/private key pair, and users have hundreds of them kept in their wallets. When the payment amount exceeds the value of each of the available Bitcoins in one address it performs multi-input transaction, aggregating different input addresses from the same user and linking different addresses each other.

When it comes to privacy in blockchain-based cryptocurrencies two main properties can be identified. *Anonymity*, that is, hiding the identities of payer and payee in a transaction and, *Confidentiality*, i.e. hiding the amount of coins transferred. However, as it has been analyzed in several works [3] [37] [56], Bitcoin lacks of confidentiality and it is subject to diverse privacy issues, users are not actually anonymous in their transactions, only pseudonymity is ensured. The usage of different pseudonyms achieves certain degree of unlinkability, but it not sufficient to become anonymous, especially when different user's addresses need to be combined in one transaction to sum up the transferred amount, as different transaction can be linked to each other.

Moreover, several thefts have occurred in Bitcoin with fraudulent mix services, in 2015 Darknet Marketplace, more than 10 million were stolen in Bitcoins, thereby it is important to devise anonymity-enhancing solutions to prevent, by design, coins theft. In this sense, recently a plethora of solutions have been proposed to deal with privacy concerns in Bitcoin.

Transaction mixing services or tumblers such as Bitcoin Fog [51] can leverage privacy by splinting transactions into smaller ones, obfuscating and scheduling them. Mixing services [50] [68] provide a mixing address to receive coins from different users and send them randomly to a fresh address for each user. The main concern

is that the mixing service might know both the input and output addresses pairs, so anonymity is not preserved against the mixer service. Besides, the delay to reclaim coins must be big as it requires to have enough amount of coins to be mixed. Moreover, the mixing service might theft the coins in case it does not deliver the transaction to the appropriate agreed output address. To deal with this, different proposal like Mixcoin [12] provides accountability prior mixing, giving to the payee a signed warranty which will enable him to prove whether the mixing has been forged.

Blindcoin [69] modifies Mixcoin protocol to ensure that the input/output address mapping is hidden with respect to the mixing server. It uses a blind signature scheme as well as an append-only public log.

Similarly, Tumblebit [30] is a coin mixing protocol that prevents users to trust in the tumbler for privacy or security, to perform the transactions anonymously, even with respect to the tumbler service. In the majority of mixer services including Tumblebit and Mixcoin, transactions must transfer the same amount of funds, to avoid linkability of input and output accounts.

Maxwell [45] introduced the Confidential Transactions (CT) approach, defining a transaction format that ensures payment value privacy in the blockchain. The input and output amounts in a transaction are hidden in Pedersen commitments [54]. The transaction includes a Zero-Knowledge Proof (ZKP) that proofs that all the outputs are positive, and the sum of the committed inputs is greater than the sum of the committed outputs.

Stealth Addresses (SA) [66] allows payees to publish a single, fixed, address that payers can send funds efficiently and privately. It aims to avoid that payers and third-parties can learn what other payments have been made to the stealth address.

Coinjoin [44] supports multi-input multi-output (MIMO) of transactions, to hind the relation between payers and payees. Transferring funds from potentially concealed inputs to output address. However, CoinJoin can mix only small amounts of coins between the users who are currently online. Besides, it requires effort to find out appropriate mixing partners and it might be subject to DOS attacks by third parties and users if they refuse to provide a signature during the signing round of the protocol.

Dash [22] is a cryptocurrency that focuses on ensuring anonymity and privacy, it provides a mixing service interacted by user's wallet that mixes user inputs with the inputs of two other users.

Coinshuffle++ [59] is a mixing protocol employed to create CoinJoin transactions.

Xim [10] is a two-party mixing protocol that is compatible with Bitcoin. It includes a decentralized system for anonymously pooling mix partners based on advertisements placed in the blockchain. Reducing Sybil attacks, denial-of-service attacks, and timing-based inference attacks.

Valueshuffle [58] extends the mixing protocol CoinShuffle++, and combines Stealth Addresses and the Confidential Transactions to provide payer and payee anonymity as well as payment value privacy. It ensures that inputs and outputs of the CoinJoin transaction cannot be linked. Since it uses SA, it provides one-time addresses for receiving payments, thereby providing payee anonymity. Furthermore, as it is based on CoinJoin it inherits theft resistant capabilities, compatibly with pruning of spent outputs. Unlike Tumblebit, or traditional mixers, in Valueshuffle funds are directly sent in one single transaction to the receivers of the CoinJoin transaction.

Möbius [47] uses a Smart Contract for performing the mixing, instead of using the the mixing service or Tumbler, it accepts any kind of blockchain token and can use anonymity protocol such as ZCash or Monero. It reduces high coordination costs of decentralized solutions and standalone cryptocurrencies, enhancing availability.

ZeroCoin [49] is a cryptographic extension to Bitcoin that augments the protocol to allow fully anonymous currency transactions. It uses signature of knowledge on messages (ZKSoK) to sign the Bitcoin transaction hash instead of using ECDSA. ZeroCoin authenticates coins using ZKPs to demonstrate that coins are in a public list of valid coins maintained on the blockchain. One of the main issues of ZeroCoin is performance as it uses double-discrete-logarithm proofs of knowledge that takes around 450 ms to be verified (at the 128-bit security level). Besides, ZeroCoin does not hide the amount of transactions, and it does not support payments of exact values.

Zerocash [62] is an decentralized anonymous payment scheme that provides a anonymity-by-design solution leveraging zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). Zerocash outperforms ZeroCoin, reducing size of transactions and verification time, hides transactions amounts and allows transactions of any kind.

One of the drawbacks of Zerocash and ZeroCoin is that it is not possible to see the outputs that have been spent already, therefore, blockchain pruning (to reclaim free disk space) is not possible. Zerocash uses zkSNARKS, meaning that a trusted setup must be ensured. Besides, zkSNARKS might be subject to non-falsifiable cryptographic hardness assumptions [28].

BulletProofs [14] is a protocol based on non-interactive Zero-Knowledge Proof that employs short proofs and without the requirement of a trusted setup. It uses the Fiat-Shamir heuristic for making it non-interactive. This is built on the techniques of Bootle et al., which relies only on the discrete logarithm assumption, and the proofs are made. Multiple range proofs are aggregated in BulletProofs, e.g. for transactions with multiple outputs, into a single short proof. It is used for CT in Bitcoin, as the transactions can have two or more outputs. It also allows to aggregate multiple range proofs from different users into one single aggregated range proof.

Luin Li et al. [40] defined and evaluated a research novel proposal for privacy-preserving a blockchain incentive vehicular network via an efficient anonymous announcement aggregation protocol, called CreditCoin. It allows that different users can send announcements and generate signatures and in a potentially untrusted environment. CreditCoin features conditional privacy as it is able to trace anonymous announcements of malicious users' identities.

Hardjone et al. [29] proposed a privacy-preserving ChainAnchor system on permissioned blockchain that uses ZKPs sent by the user to proof their membership against a permissioned verifier. They proposed a mixed permissioned - permissionless blockchain in which an organization can overlay their own permissioned sub-group on top of the public permissionless blockchain, while maintaining the anonymity of its group-members.

Dunphy et al. [23] provides an interesting analysis and comparison of the main DLT-based Identity management schemes, concretely, UPort, ShoCard and Sovrin.

Non-interactive ZKPs have been proposed as a tool to enable complex privacy-preserving smart contracts [17]. However, performance is a problem as communications and computational power required in public blockchains with Smart Contracts are high.

Blockchain e-voting protocol [46] that does not require a trusted entity to protect the voter's privacy.

Baars et al. [5] studied the self-sovereign identity concept, and designed an architecture for a Decentralized Identity Management System (DIMS) using claim-based identity and blockchain technology.

Hawk [36] is a privacy-preserving decentralized smart contract system where contractual parties interact with the blockchain, using cryptographic primitives such as ZKPs.

CryptoNote [70], the destination of each CryptoNote output is a public key, derived from recipient's address and sender's random data.

In cryptonote unlike Bitcoin each destination key is different. It uses ring signatures, and concretely a custom one-time ring signature scheme. In CryptoNote, the sender performs a Diffie-Hellman in order to obtain a shared secret, that is derived from his data along with first part of the recipient's address. Afterwards, a one-time destination key is computed, derived from such a shared secret the other part of the address. For these two steps, it requires two EC-keys for the recipient, meaning that address in CryptoNote are larger than a Bitcoin wallet address. Similarly, receiver also carries out the Diffie-Hellman to get the secret key.

CryptoNote might have concerns dealing with the ring signatures depending on the large of anonymity set n , as it requires that each transaction contains a ring signature of size $O(n)$. Besides, storing ring signatures in public blockchain might become a problem. Unlike CryptoNote, CoinJoin [44] or ValueShuffle [58] facilitate pruning, which is a drawback in Cryptonote, as rings signatures make the pruning difficult. Another cryptocurrency based on traceable ring signatures was ShadowCash, but an implementation error resulted in the partial de-anonymization of 20% of the one-time keys used. Monero evolved the technology to use Ring Confidential Transactions, concealing both, amount and the source funds to anyone, but the recipient and sender.

Enigma [72], developed by MIT, allows secure data sharing, using multi-party computation (MPC) and homomorphic encryption. Enigma aims to allow developers to build *privacy by design* and decentralized applications, avoiding a trusted third party (TTP). Enigma cannot be defined as a cryptocurrency or a blockchain platform. Instead, Enigma connects to an existing blockchain, and performs computation offloading of the private and intensive computations on an off-chain network. Unlike in blockchain schemes, the incentives are based on fees instead of mining rewards, where nodes are compensated for providing computational resources. The nodes pay a security deposit, which deters malicious nodes.

Enigma uses secure multi-party computation (sMPC) technology, in which data queries are distributively computed, and data is divided across different nodes each one computing certain functions in a distributed way without leaking information to the other.

H2020 EU project "**My Health - My Data (MHMD)**" [53] defines a privacy-by-design blockchain solution by defining smart contracts that enforces dynamic consent mechanisms and peer-to-peer data transactions between public and private healthcare providers and patients.

3.2 Self-sovereign identities in blockchain

Centralized Identity management solutions, based on traditional Central Authorities, are subject to different problems and threats such as data breaches, identity theft and privacy concerns. The rise of federated Identity management models helped to mitigate partially those problems. This kind of server-centric systems enable users to adopt the same identity system across different domains. The user is redirected for authentication and user identity data retrieval to his home identity provider. Some federated IdM systems such as Stork go a step forward implementing a user-centric

approach, since users are put in the middle to take control of their personal data, asking their consent each time their user data is released in the federation from its home identity provider (data controller) to the Service provider (data processor).

Several technologies such as, OpenId, Oauth, SAML or Fido, allow implementing this user-centric approach, enabling users to share its identity across different services. Nonetheless, user-centric identity federations are still subject to privacy issues, as user data related to their identity is still hold in server side, and authentication is validated in the server (usually through a Knowledge-base and some other weak authentication e.g. passwords).

Unlike those traditional approaches, the self-sovereign identities [65] focuses on providing a privacy-respectful solution, enabling users with fully control and management of their personal identity data without needing a third-party centralized authority. Thus, citizens are not anymore data subjects, instead, they become the data controller of their own identity. This is, they can determine the purposes, and ways in which personal data is processed.

Blockchain enables sovereignty as users can be endowed with means to transfer digital assets to anyone privately, without rules in behind.

Christopher Allen described in the detail this path [2] to self-sovereign identities to achieve a full user autonomic, and detailed the ten Principles of Self-Sovereign Identity: Existence, Control, Access, Transparency, Persistence, Portability, Interoperability, Consent, Minimization and Protection.

Identity sovereign have been already realized in the past following a user-centric and privacy-preserving approach for mobiles using Anonymous credential systems, e.g. in the scope of Irma project [13], and even, for IoT scenarios, e.g. in our previous works [9] [60]. Unlike those proposals, nowadays, self-sovereign identities are being lead to a new stage, as they are being materialized through blockchain, which facilitates the governance of the SSI system, increases the performance to internet scale and enables the accessibility of identities to everyone. In this sense, latest solutions [43] [65] make use of distributed ledger technologies (DLT), along with user-centric and mobile-centric approaches, thereby allowing users to maintain secure their needed crypto-credentials linked to the blockchain, which acts as distributed and reliable identity verifier, providing provenance and verifiability.

uPort [43] uses 20-byte hexadecimal identifier to represent the user's uPortID, with the address of a Proxy smart contract deployed in Ethereum. Such a contract introduces a layer of indirection between the user's private key - maintained on their mobile device - and the application smart contract being accessed by users. The user app contacts an instantiation of a Controller Smart Contract (which holds the main access control logic such as authentication or authorization), which in turn, is the unique entity capable of interacting with the proxy. uPort supports certain degree of unlinkability, as users can create many unlinkable uPortIDs. Selective disclosure of attribute is allowed, encrypting an attribute with a symmetric encryption key, which are individually encrypted with the public key of the identity allowed to read the attestation attribute. uPort also supports identity recovery and rely on DID¹ standard and Verifiable Claims², both being standardized by the W3CC.

Sovrin [65], is an open-source decentralized identity network for permissioned blockchain. Sovrin is an utility identity deployed over Hyperledger-Indy³ that implements the Plenum Byzantine Consensus (BPT) algorithm. Sovrin supports DPKI (Decentralized Public Key Infrastructure), where every public key has its own public

¹<https://w3c-ccg.github.io/did-spec>

²<https://www.w3.org/2017/vc/charter.html>

³<https://www.hyperledger.org/projects/hyperledger-indy>

address in the ledger (DID, decentralized identifier) that enable universal verification of claims. Sovrin allows having different DID for every relationship the user has, with different keypairs, unlinkable each other. Like in uPort, a Sovrin client generates crypto keypairs and maintains the private key in the user's mobile. It supports identity recovery, and makes use of software Agents that can act on behalf of the user to facilitate interactions with third party service provider agents. Unlike uPort, Sovrin supports not only attestation and verifiable assertions, but also Anonymous Credentials with Zero-Knowledge Proofs to achieve fully unlinkability and comply with the minimal disclosure principle. Sovrin allows to demonstrate proofs offchain directly in a secure channel with the third party, without storing the attributes in the ledger. In this case, the blockchain is used to identify the trusted service endpoint to interact with. The web of trust supported by the Sovrin trust anchors provides verifiability of the target party being interacted.

Shocard [63] uses the ledger as a repository of certifications that maintains signatures-of-hashes-of each personal data attribute along with a code to avoid brute-force discovery. Shocard is blockchain agnostic, and uses private parallel sidechains to speed up the transactions in the ledger. It supports third parties to verify and then certify an individual's identity and credentials. Shocard provides an app that holds cryptokeys and entities can verify a user's claims of identity through certifications and signatures hold in the ledger. The ShoCardId can be bootstrapped from trusted breeder document e.g. ePassport, through an IDproofing stage to validate user's identity, checking biometrics in the ePassport chip. However, the enrollment with biometrics requires storing encrypted sensitive data in the Shocard server. This server might trace interactions between Relying parties and ShoCardIDs.

Civic [64] is a decentralised trusted Identity applications that provides identity proofing relying on existing eID documents like ePassports. The app stores private keys of the user's Civic ID used to record signed hashes of attestations in the blockchain. It supports multiple-factor authentication, user-consent, and minimal disclosure. However, anonymity and unlinkability mechanisms are not seem to be implemented in Civic yet. And the civic server can act as an intermediary authorization server between the user and service provider, which raises privacy concerns.

Different solutions such as the Civic platform [64], are starting to apply the user-centric and decentralized blockchain approach to provide real-time authentication through biometrics, where identity data is encrypted in the mobile app. These solutions, uses Merkle tree randomized hashes using nonces signed by the validators entities, and supporting selective disclosure of the identity information (certain portions of the Merkle tree hashes are revealed) in the blockchain, after user consent, enabling user control privacy and enhancing security, since the attestations and proofs cannot be tampered in the blockchain.

To conclude this chapter, we can identify that the most common solution to privacy is the heuristic of mixing to hinder linkability of data, but the solutions that don't rely on trust on third mixing parties make use of powerful cryptographic tools. Chapter 4 will focus on the fundamentals of this technologies that allow strong privacy assertions for the blockchains that use them.

Chapter 4

Privacy preserving cryptography in blockchain

This chapter presents the mathematical principles to enable privacy preserving blockchains. Specifically, this chapter will study Zero-Knowledge Proofs, zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARK) and Ring Signatures. To complete each technology explanation, some examples are given with regards to blockchain technology. Other technologies that rely on third parties to achieve privacy-preserving in the blockchain after bootstrapping are left as future work to study, although many of them will rely on the cryptographic tools explained in this chapter.

4.1 Zero-Knowledge Proofs

One of the first privacy preserving solutions applied to blockchain were Zero-Knowledge Proofs. A Zero-Knowledge Proof (ZKP) is a cryptographic protocol that allows a party, the Prover, to proof to another entity, the Verifier, that a given statement is true, without revealing any information except that the proof itself is correct.

The Zerocoin [49] cryptocurrency proposed a model to integrate with Bitcoin (and virtually any other cryptocurrency) adding **unlinkability** to the transactions, using accumulators, digital commitments and Zero-Knowledge Proofs over them.

Although cryptocurrencies are the more typical type of blockchain, other applications of the blockchain technology use Zero-Knowledge Proofs to create a decentralized Public Key Infrastructure (PKI) of Anonymous Credential Systems (ACS).

Before commenting on the use of ZKP over blockchain, the following sections presents their mathematical foundations. The main bibliographic reference used in this section is [34, Chapter 12].

4.1.1 Complexity theory: P and NP Problems

To understand what Zero-Knowledge Proofs can proof, we have to briefly review the class of problems P and NP.

When we talk about an algorithm, we say that it has a certain complexity depending on how long, respect to the size n of the problem, the algorithm will take to run. If the algorithm is of complexity $O(n^k)$, with k some constant, we say it is *polynomially bounded*. We usually say that a *polynomial time* algorithm is *good* or *efficient*, and an *exponential time* algorithm is *bad* or *inefficient*. Depending on the power k or the *hidden constants* of the complexity notation O , a polynomial time algorithm may be worst than an exponential time one for small sizes of n , but for now, we won't take that in count.

Algorithms are used to solve problems. In this case, we are interested in *decision* problems, i.e. a question whose answer can be *true* or *false*. One example is the decision problem of the composite integer: *Is an integer n the product of two or more integers greater than 1?* The answer can either be *false*, if n is prime, or *true*, if it is composite.

The set of decision problems solvable in polynomial time (there exists a polynomial time algorithm that solves it) is called the *class P*.

A super-set of **P** is the *class NP* of decision problems which a *true* answer can be *verified* in polynomial time, given some extra information called *witness*. For example, for the composite integer problem, if we are given the decomposition of n in primes, we can multiply them and check equality with n in polynomial time, therefore, it is an problem in the class **NP**.

The usefulness of the **P** and **NP** classes are that it is an open problem to decide if $\mathbf{P}=\mathbf{NP}$ or $\mathbf{P}\subset\mathbf{NP}$. If a problem in **NP** is unknown to be in **P**, it means there is not a known polynomial time algorithm that solves it, and it is considered *difficult* to solve.

Another subset of the class **NP** are the **NP-Complete** problems, or **NPC**. Informally, a problem belongs to **NPC** if it is a **NP** problem and any other problem in **NP** can be solved by transforming it, in polynomial time, to an instance of the **NPC** problem. In other words, a function that solves a **NP** problem can call in polynomial time a subroutine that solves the **NPC** problem, although the subroutine may not be of polynomial time. Cook's theorem [19] proofs that the satisfiability problem is **NPC**.

4.1.2 Interactive Proof Systems

An interactive proof system is an interactive protocol, polynomially bounded in the number of messages, that solves a decision problem Q , between two entities, called the *Prover*, a computationally unbounded machine, and the *Verifier*, a probabilistic polynomially bounded machine. At the end of the protocol, the Verifier either accepts that the instance of the problem Q is *true*, or rejects the proof and considers it to be *false*. The protocol has two properties:

- *Completeness*: For any instance q of the problem Q that is indeed *true*, the Verifier accepts q as *true* after the protocol.
- *Soundness*: For any instance q of the problem Q that is *false*, the Verifier rejects the proof with a probability greater than $\epsilon = 1 - n^{-c}$, for n the size of the instance q , $c > 0$ a constant.

In other words, if the Prover wants to proof as *true* an instance that is *true*, the protocol will always work, there aren't false positives; but if the Prover is a cheater and wants to proof as *true* an instance that is actually *false*, he may succeed and deceive the Verifier with a negligible probability.

4.1.3 Zero-Knowledge Proof Systems

A Zero-Knowledge Proof system is an interactive proof system with a third property, called *zero-knowledge*.

First, let's define a *Simulator*: it is a probabilistic polynomially bounded algorithm, that can generate valid transcriptions of an interactive proof, without interacting with the Prover. The transcripts are valid in the sense that any third party

reading the transcription could not differentiate between a real interactive transcription, between a Prover and Verifier, and the simulated one. We call *ensemble* the set of random variables that represent a transcription. A Simulator generates an ensemble, where each transcription generated will have a certain probability. Two Simulators are identical, if the two ensembles they generate have the same probabilities.

An interactive proof system (complete and sound) for a decision problem Q , is *Zero-Knowledge* if there exists a Simulator of transcriptions which its ensemble is identical to the ensemble generated by the Prover and Verifier, for any *true* instance q of Q .

The last property is also called *perfect Zero-Knowledge*. There exist more relaxed definitions of Zero-Knowledge depending on what assumption we modify. *Statistical Zero-Knowledge* means that the probability ensembles of the Simulator and the protocol are asymptotically identical. *Honest-Verifier Zero-Knowledge* affects to the information available to the Simulator, which supposes that the Verifier will follow the protocol and never cheat. *Honest-Prover Zero-Knowledge* applies the same principal to the Prover. And anticipating the next section, an *argument* is a proof where the Prover is considered to be polynomially bounded like the Verifier.

To sum up, the definition of Zero-Knowledge means that any party with access to a transcription can not differentiate between a real interaction and a simulated one, and therefore, the information that it can acquire from the transcription is the same that if the transcription where generated by the Simulator. Because the Simulator does not have access to the Prover, the only computationally unbounded machine, one cannot get any new information from a transcription that a Simulator could get. This is what means for a protocol to be *Zero-Knowledge*, that from the protocol itself, we get no other information, except that the transcription is valid.

Then, why does the Verifier trust the proof? Because most of the Zero-Knowledge Protocols are built with the following scheme:

1. Prover \rightarrow Verifier: Random *commitment*.
2. Verifier \rightarrow Prover: Random *challenge*.
3. Prover \rightarrow Verifier: Commitment opening, depending on the challenge and the instance q of the problem Q being proved. If q is *true*, the Prover always can open the commitment. If q is *false*, there is a probability $\frac{1}{\lambda}$ of the cheating Prover being able to open the commitment. If the opening is not valid, the Verifier rejects the entire proof.

Repeat the previous steps n times, until the Verifier accepts the proof with a soundness of $\frac{1}{\lambda^n}$.

In the second step, the Verifier is the one choosing the challenge randomly, and therefore trusts that the Prover could not guess its value. The first commitment message binds the Prover to answer the challenge honestly, or there would be some probability of the Verifier rejecting the proof, which increases with the more iterations are performed.

Manuel Blum [11] proved that with this scheme we can construct any Zero-Knowledge Proof. In 1990, Quisquater et al. [33] published a short story explaining the intuition behind Zero-Knowledge Proofs without this much mathematical formalism. Here we present an adapted version which will help to identify the three steps in the protocol and how the Verifier trusts the proof in the end:

Imagine a cave, where the path forks in two passages, and at the end of each one, they join again, with the shape of a ring. In the point the paths meet, there is a magic door, that only opens when someones pronounces the magic work.

*Peggy knows the secret word and wants to **prove** it to her friend, Victor, but without revealing it. Peggy and Victor meet at the entrance of the cave, then Victor awaits while Peggy goes inside the cave, taking one of the passages, that we will name A and B. Victor can't see which way Peggy went.*

When Peggy arrives at the door, Victor enters the cave, and when he arrives to the fork, stops and yells which path, A or B, he wants Peggy to come back, to verify she knows how to open the door.

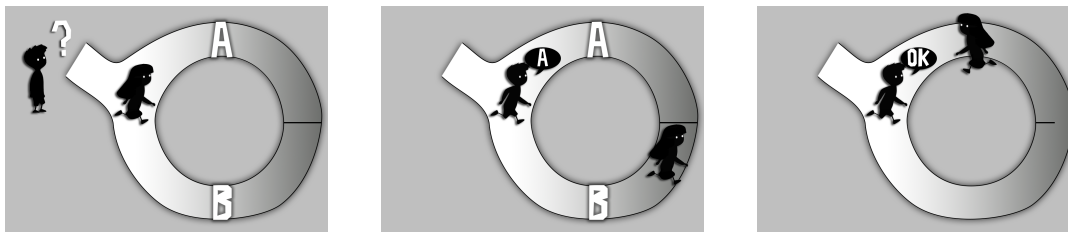


FIGURE 4.1: The cave story. Images by Adrián Sánchez Martínez.

If Peggy actually knows the secret, she always can take the requested path, opening the magic door if needed. But if Peggy doesn't know the magic word, she has a 50% chance of guessing correctly what passage Victor was going to ask. That means she had a chance to fool Victor.

Victor then asks to repeat the experiment. With 20 iterations, the chances of Peggy fooling Victor in all of them is only 2^{-20} , practically negligible.

Eve, curious about what Victor and Peggy were doing in the cave, eavesdrops Victor during the process from a bush. The problem is that Eve doesn't know if Peggy and Victor agreed on what paths to choose, because they wanted to prank her for being nosy. Only Victor is confident he is choosing the returning passage in a random way.

Later, Eve asks Victor about what happened in the cave. Victor tells Eve about the door and how he is convinced that Peggy knows the magic word to open it, but he can't prove it to Eve because he doesn't know the secret word to open the door.

In the story, Peggy *commits* by entering through one of the two paths, then Victor challenges with a random path, and only he trusts himself to choose it randomly, and finally Peggy responds to the challenge with the correct path.

The simile between the story and the protocol template is clear: a Prover chooses a problem *difficult* to solve directly by the computationally bounded Verifier, what usually means a problem in the class **NP**. Then, the computationally unbounded Prover can solve it, or a computationally bounded Prover knows a *witness* that helps him compute a commitment and open it on the Verifier's challenge.

Nowadays, many of the ZKP protocols in practice are based on the discrete logarithm problem, the same problem the RSA protocol is based on for asymmetric encryption. Because our machines aren't able to compute the discrete logarithm for big modules, the Prover knows the discrete logarithm (the *witness*). The most representative protocol is the Schnorr's identification scheme.

Using the notation introduced by Camenisch and Stadler [16], the discrete logarithm ZKP can be written as $ZPK\{(\alpha) : y = g^\alpha\}$, given a known group $G = \langle g \rangle$ of prime order q and public value $y \in G$. The notation means: “I know a secret value α such that g^α is y ”, i.e. the discrete logarithm of y , $\log_g y = \alpha$. It is called an *identification protocol* because it can be used as an authentication system, where an identity is assigned to the public value y , and the secret value is α , only known to the user.

During the first step of the protocol, the Prover chooses randomly a value r , computes the *commitment* $t := g^r$ and sends t to the Verifier. Then, the Verifier chooses another random exponent, the *challenge* c , and sends it to the Prover. Next, the Prover computes the *response* $s := r - c\alpha \bmod q$ and sends it to the Verifier. Finally, the Verifier checks whether or not $t \stackrel{?}{=} g^s y^c$ holds. The Verifier never receives the secret value α , nor is able to compute it, given t and s .

The protocol holds because $g^s y^c = g^{r-c\alpha} g^{\alpha c} = g^r = t$.

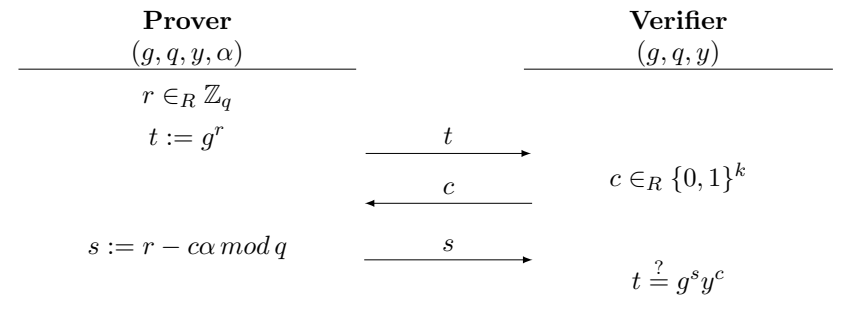


FIGURE 4.2: Schnorr’s protocol $ZPK\{(\alpha) : y = g^\alpha\}$. The Prover knows (g, q, y, α) such that $g^\alpha = y$. The Verifier knows (g, q, y) .

Nonetheless, the necessity of synchronizing the Prover and the Verifier in order to exchange messages in multiple iterations of the protocols is a nuisance. As a solution, the Fiat-Shamir heuristic lets us replace the Verifier’s challenge with a random oracle model, or in practice, a hash function \mathcal{H} .

For example, in the Schnorr protocol, the Prover computes the challenge as $c := \mathcal{H}(g \mid y \mid t)$. The Prover then sends c and s in a single message. Because the Verifier knows g and y , it can verify the proof by checking the equality $c \stackrel{?}{=} \mathcal{H}(g \mid y \mid g^s y^c)$.

4.1.4 Commitment schemes

From the example Zero-Knowledge Proofs shown, the protocols follow a common scheme, where the first message was always a commitment from the Prover, which binds it to be honest in order to be successful, even if the Prover is computationally unbounded. The second message is from the Verifier, a challenge to that commitment. Finally, the Prover *opens* the commitment as a response.

This section showcases the commitment schemes, a tool that allows to give any NPC problem a (computational) ZKP, as proven by Manuel Blum in *How to Prove a Theorem So No One Else Can Claim It* [11]. The basic bibliography for this section can be found in [21, 55].

These schemes allow to hide the structure of a *true* instance, where the Prover commits to it before knowing the challenge from the Verifier, so the Verifier cannot obtain any information from the commitment. In the Prover’s last message, the response, a little piece of the *true* instance’s structure is revealed, but from where

the Verifier can neither learn anything, only verify the correctness of the proof. Let's begin with the definition of a commitment scheme for hiding one bit:

- A commitment scheme for one bit is a pair of functions (f, v) of polynomial time. The function $f : \{0, 1\} \times Y \rightarrow \chi$ transforms bit $b \in \{0, 1\}$ with a random key $y \in Y$. The value $x = f(b, y)$ is called a *blob* or *witness* of b . The verification function $v : \chi \times Y \rightarrow \{0, 1, \bullet\}$ either opens the blob, revealing the bit b , or shows that the pair (blob= x , key= y) is not valid.

The pair (f, v) must satisfy the following conditions:

1. *Binding*: For all blob $x = f(b, y)$, the Prover is not capable of finding a value $y' \neq y$ such that the blob can be opened with a different value, i.e. $v(x, y) \neq v(x, y')$.
2. *Hiding*: The ensembles $\{f(0, Y)\}$ and $\{f(1, Y)\}$ are indistinguishable.

The bit commitment schemes can be classified in two types:

- *Unconditional binding*. A machine P with unlimited computational power (such as the Prover in ZKPs), cannot change the committed bit sent in the witness.
- *Unconditional hiding*. The ensembles $\{f(0, Y)\}$ and $\{f(1, Y)\}$ are identical. That is, no computationally unbounded machine can distinguish between the ensembles with higher probability that choosing randomly the value of the committed b .

If a commitment scheme is unconditionally binding, it cannot be unconditionally hiding, and vice versa.

Suppose we have the unconditional binding property, P cannot find y' such that x opens with a different bit. Then, V could distinguish to which b the witness x corresponds, with the simple algorithm of trying all values $y' \in Y$ (we are supposing P and V computationally unbounded), until finding $v(x, y') \neq \bullet$. Therefore, a bit scheme can never be unconditionally hiding and binding at the same time.

Nonetheless, a weaker version of the properties can be applied if we suppose P and V polynomially bounded. The properties are then called *Computational hiding* and *Computational binding*.

An example of a computational binding and unconditional hiding commitment scheme with the discrete logarithm:

- *Hiding*: P chooses randomly $y \in Y$ and hides bit b as $x = f(b, y) = s^b g^y \bmod p$.
- *Opening*: P sends V the value y . V checks bit b :

$$b = v(x, y) = \begin{cases} 0 & \text{if } x \equiv g^y \bmod p \\ 1 & \text{if } x \equiv sg^y \bmod p \\ \bullet & \text{in other case.} \end{cases}$$

The commitment schemes can be expanded for strings of bits, hiding the value and opening it with the same properties of binding and hiding. The hiding function is now defined as $f : \{0, 1\}^n \times Y \rightarrow \chi$, taking the string $s = (b_1, \dots, b_n) \in \{0, 1\}^n$ and the random key $y \in Y$. The opening function is $v : \chi \times Y \rightarrow \{0, 1, \dots, 2^{n-1}, \bullet\}$, that either opens the string s or shows that the pair (blob= x , key= y) is not valid.

The Pedersen commitment scheme is based on the discrete logarithm problem. It is computationally binding and unconditionally hiding. Be p the prime modulo,

$g \in \mathbb{Z}_p^*$ the generator and h a random value such that $\log_g h$ is unknown. The key set is $Y = \mathbb{Z}_{p-1}$ and the blob set $\chi = \mathbb{Z}_p^*$. The commitment of a string s is as follows:

- *Hiding*: P chooses randomly $y \in Y$ and hides s as $x = f(s, y) = g^s h^y \bmod p$.
- *Opening*: P sends V the pair (s', y') . V checks that:

$$s = v(x, y) = \begin{cases} s' & \text{if } x \equiv g^{s'} h^{y'} \bmod p \\ \bullet & \text{in other case.} \end{cases}$$

4.1.5 Anonymous Credential Systems

One of the main applications of Zero-Knowledge Proofs are in attribute-based credential systems. They consist on a list of name-value pairs, with a special signature that then allows the user to proof the ownership of a valid credential, without disclosing the attribute values.

There exist multiple anonymous credential systems based on ZKP, like Microsoft's U-Prove [], but the most complete and versatile protocol, in the author's opinion, is IBM's Identity Mixer, although some criticism remains in their revocation system, for example.

The secret to Identity Mixer's credentials is the Camenisch-Lysyanskaya signature:

Camenisch-Lysyanskaya signature

The CL-signature is an interactive protocol where an Issuer and a User sign together a list of messages (attribute values of the credential) with the Issuer's private key. The protocol allows the User to hide certain attributes from the Issuer, but at the same time, the Issuer can include them in the signature.

This happens thanks to ZKPs exchanged between the Issuer and User, proving correctness on their part of the signature computation, without revealing the attributes (zero-knowledge). A generalization of the Schnorr's protocol is used in this case. Instead of proving that $y = g^a$, the Prover shows that he knows x_1, \dots, x_l such that $y = \prod_{i=1}^l g_i^{x_i}$, for l different bases in the group.

The commitment is $t = \prod_{i=1}^l g_i^{r_i}$, for r_i random values. The challenge is like before, a single random integer of k bits $c \in_R \{0, 1\}^k$. The response consists of a list of l values $s_i = r_i - c \cdot x_i \bmod q$, for $i = 1, \dots, l$. The Verifier checks that $t = y^c \prod_{i=1}^l g_i^{s_i}$.

The protocol works because:

$$y^c \prod_{i=1}^l g_i^{s_i} = \left(\prod_{i=1}^l g_i^{x_i} \right)^c \prod_{i=1}^l g_i^{s_i} = \prod_{i=1}^l g_i^{cx_i + r_i - cx_i} = \prod_{i=1}^l g_i^{r_i} = t$$

The CL-signature needs some system parameters to be defined by the Issuer, some of them will be private values and the rest public. Let p' and q' be two primes, such that $p = 2p' + 1$ and $q = 2q' + 1$ are *secure primes*. The public system parameters are: the modulo $n = pq$, $Z, S, R \in QR_n$ (the multiplicative group of quadratic residues modulo n). The private key of the Issuer is (p, q) .

A CL-signature over a single message m is the tuple (A, e, v) such that $A \equiv \left(\frac{Z}{S^v R^m} \right)^{1/e} \bmod n$, where e, v are random, e prime and $\frac{1}{e} \cdot e \equiv 1 \bmod \varphi(n)$.

To verify a CL-signature it's enough to check if $Z \stackrel{?}{\equiv} A^e S^v R^m \bmod n$.

To sign a list of l different messages, as we observe that the base for the single message was the value R , we will now need $R_i \in QR_n$, for $i = 0, \dots, l$ different public values. The signature over the messages m_0, m_1, \dots, m_l is (A, e, v) such that

$$A \equiv \left(\frac{Z}{S^v \prod_{i=0}^l R_i^{m_i}} \right)^{1/e} \mod n$$

To verify the signature, check $Z \stackrel{?}{=} A^e S^v \prod_{i=0}^l R_i^{m_i} \mod n$.

One great property of this signature, applicable to privacy, is its randomization. The User can generate a new tuple (A', e, \hat{v}) different to (A, e, v) , that is still a valid signature from the Issuer, even without the private key (p, q) .

The User chooses a random integer r and calculates $A' = A \cdot S^{-r} \mod n$ and $\hat{v} = v + er$. This new signature holds for the same verification because:

$$A'^e S^{\hat{v}} \prod_{i=0}^l R_i^{m_i} \equiv A^e S^{-er} S^v S^{er} \prod_{i=0}^l R_i^{m_i} \equiv A^e S^v \prod_{i=0}^l R_i^{m_i} \equiv Z \mod n$$

The interactive protocol for the CL-signature is depicted in Figure 4.3. The User hides the first 3 attributes to the Issuer, but using a ZKP, the Issuer trusts that the part of the signature that the User computes is right. The attribute m_0 is usually used for the User's private key, therefore, the CL-signature should always be interactive, at least hiding m_0 . The Issuer continues with the signature, using its private key to compute the inverse of e in the exponent. Finally, the User computes v and verifies the Issuer's ZKP. Because the User didn't disclose v' , he is the only one who knows the complete tuple (A, e, v) , the Issuer can't register the tuple to later identify the User, if he doesn't randomize the signature.

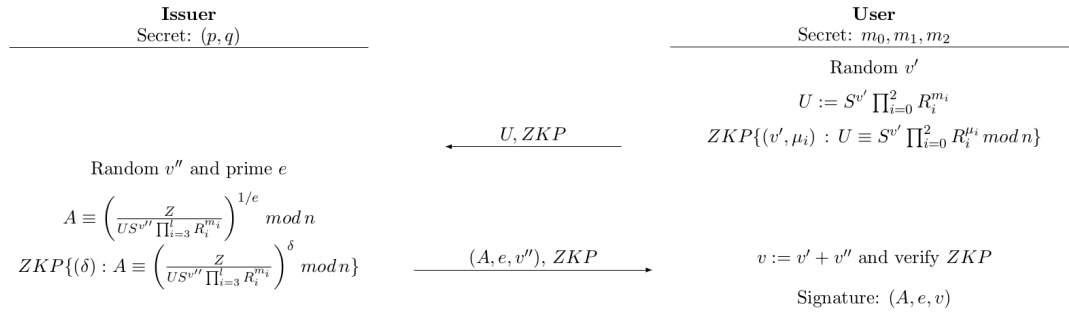


FIGURE 4.3: Interactive CL-signature.

Selective disclosure

One of the most important applications of an Idemix credential is the selective disclosure of attributes to a Service Provider, in order to authenticate, showing a valid signature of the Issuer, but disclosing the minimum information required.

Suppose the User has a CL-signature (A, e, v) over the attributes m_0 to m_l . He wants to hide m_0 , the secret key, m_1 and m_2 from the Verifier, but send the rest of the values.

The User would first randomize the signature: $(A' := AS^{-r}, e, \hat{v} := v + er)$.

Then, the User sends the attributes m_3, \dots, m_l and following proof:

$$\text{ZKP}\{(\epsilon, \hat{v}, \mu_0, \mu_1, \mu_2) : Z \prod_{i=3}^l R_i^{-m_i} \equiv A'^\epsilon S^{\hat{v}} \prod_{i=0}^2 R_i^{\mu_i} \bmod n\}.$$

The Service Provider, acting as a Verifier, can check the proof with the public values of the Issuer and the attributes disclosed. Thanks to the Zero-Knowledge property, the Verifier learns nothing of the hidden attributes.

Accumulator

One-way accumulators were first proposed by Benaloh and de Mare [8]. An accumulator allows to combine many elements into a public constant-size data structure, which allows parties to prove that one specific value is contained within in, without revealing which value of them all that form the accumulator. It is a form of ZKP of set membership.

Camenisch and Lysyanskaya [15], the same creators of the interactive blind CL-signature, developed an accumulator based on the Strong RSA assumption, which is in turn based on an accumulator of Baric and Pfitzmann [6] and Benaloh and de Mare [8].

The Camenisch and Lysyanskaya accumulator was aimed to solve the revocation of credentials while maintaining the privacy of the credential.

If a credential were assigned an ID number to add it to a CRL list like in traditional PKIs revocation systems, then, the user should always present such ID and then all presentations of the credential would be linkable.

The solution was to make public an accumulator, a very composite number, and during the issuance, a form of factor of the value is given to the user. When presenting a credential, the user can prove in Zero-Knowledge that his non-revocation attribute is part of the accumulator. The verifier only learns that the user has such an element from the accumulator, but she doesn't learn which one, therefore, there is no linkability.

When a credential is revoked, the accumulator is updated, removing that user's factor. The drawback here is that all remaining valid credentials must actively update their non-revocation attribute, consulting the new updated value of the accumulator. Nonetheless, this update does not require the interaction with the issuer, this is performed autonomously by the user. The revoked credential is no longer capable of updating its non-revocation attribute, and neither can generate a valid proof with the current published accumulator. This derives in another issue, that is that the verifier must also update the accumulator value periodically to avoid accepting proofs from revoked clients.

Another problem the accumulators introduce for revocation is the validity period of a proof. While the nonce of a proof is used for freshness, following some verifier's policy where the freshness is valid, for example, thirty minutes or thirty days, here the accumulator dictates the validity of a proof as long as no other credential is revoked, even our credential is still valid.

4.1.6 Zero-Knowledge Proofs in Blockchain

Depending on what layer of the blockchain technology Zero-Knowledge Proofs are applied, the resulting blockchain has different applications. Because blockchain technology was defined for Bitcoin, many of the current blockchains are *alternative cryptocurrencies*, and so, Zerocoin applied ZKPs to give privacy to transactions, which in Bitcoin are all public. Nonetheless, the general research interest in blockchain is in the potential of a trusted technology built on a trustless network. The Sovrin project defined the now called Hyperledger Indy blockchain, a decentralized

PKI of Anonymous Credential Systems, that allow to trust credentials without disclosing any private information, even when built over a blockchain network.

Zerocoin

Zerocoin [49] is another cryptocurrency based on the functionality of Bitcoin. Their solution is based on Commitments over the coins and accumulators.

For each new transaction created, a serial number S is associated, and when the transaction is sent to the network, instead of S , a commitment C of the value S is attached to it. Only with a secret r value the commitment can be opened.

When the owner wants to spend the coin, she first search in the chain for a sufficiently big set of commitments $\{C_1, \dots, C_n\}$. Then, creates a ZKP stating that she knows a secret value r such that one of the commitments in the previous set opens to the value S .

The rest of the nodes will first verify that the value S has not previously been used to spend a transaction, avoiding the double spend of the same transaction. Then, they will verify the ZKP.

A naive implementation of the ZKP would be a list of logical ORs stating S opens to the value C_1 OR S opens to the value C_2 Instead, Zerocoin uses a modification of the Camenisch and Lysyanskaya accumulator, allowing for an efficient ZKP of set membership.

$$\text{ZKP}\{(r, C) : C = v(S, r) \text{ and } C \in \text{accum}\{C_1, \dots, C_n\}\}$$

Sovrin

Sovrin defined the architecture and implemented what currently is the Hyperledger Indy blockchain, part of the Linux Foundation's Hyperledger project. The Sovrin Foundation continues developping the base code of Indy, although, they operate one instantiation of the ledger.

Hyperledger Indy is a public and permissioned blockchain, which means that anyone can read and request writing to it, but only authorized parties can modify it. The Sovrin Board of Trustees ensures that the organizations that operate blockchain nodes, the Stewards, agree and comply with the Sovrin Steward Agreement. A set of rules and policies are specified in a way that not only consensus over new transactions must be achieved, but also consensus on the members that can be Stewards. This way, the blockchain itself gains some trust from the decentralization of the permissioning of nodes.

The blockchain transactions record the information of Decentralized IDentifiers (DIDs) and public keys. Optionally, a credential (also called claim) can be uploaded, ciphered or not. For public institutions, like a University, they may have a public credential in the ledger, where they include contact information. For individual users, they may upload the ciphered credential, but it is not recommended, in case their keys are broken, the information in the ledger is already stored in multiple devices and cannot be ensured the deletion of the credential.

Therefore, the recommended use of Sovrin is for Issuer entities to upload their DIDs with a public credential, and for Users to save theirs on their local wallets, but not in the ledger.

Figure 4.4 shows the Issuance and Presentation interactions. These are all off-ledger interactions, just like a user presents a traditional credential to authenticate to a Service Provider, in this case, the User presents a Zero-Knowledge Proof, based

on the Identity Mixer protocol, of a valid claim, issued by an Issuer, identified with its DID. The Verifier can check that the DID and signing public keys are valid in the blockchain. The trust assigned to the User will depend on the trust relationship between the Verifier and the Issuer.

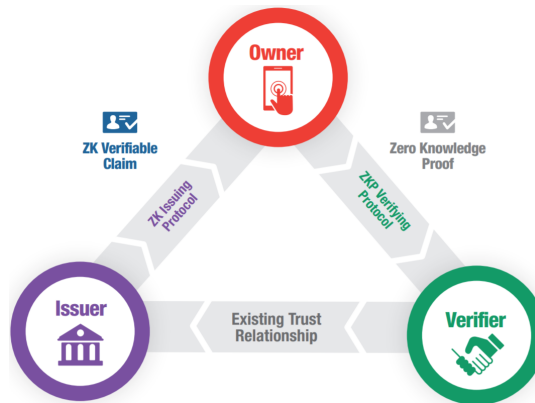


FIGURE 4.4: Sovrin interactions: issuance, presentation and trust relationship.

For example, a University may issue claims to the students. At the end of the course, the students are asked to fill a poll about the classes and professors. The poll asks for authentication before accessing the poll. The User then sends a ZKP of being a student of that class, with a domain pseudonym to ensure he or she only fills the poll once. The poll verifies that the public keys of the Issuer correspond to the public keys of the DID in the blockchain. If the Issuer and Proof are valid, the poll grants access to the student, without learning who he or she might be.

4.2 zkSNARK

Another great cryptographic tool used in blockchain are the zkSNARKs. The main bibliography for this chapter can be consulted here [27, 57, 71].

zkSNARKs are a cryptographic protocol that allow to verify the correctness of computations without executing them. In blockchain technologies, where the nodes do not trust each other and must always check the validity of a transaction, this allows to trust the execution of other nodes, without wasting computational power, nor learning private inputs.

To use a zkSNARK, first there is a process to transform a computation into an expression with polynomials, and then we can properly apply a zkSNARK.

The properties that define a zkSNARK are specified in its name: zero-knowledge Succinct Non-Interactive ARguments of Knowledge. Let's see what do they mean:

- They are *zero-knowledge*, meaning that a Simulator exists, so the Verifier learns nothing from the argument, except that it is valid.
- They are *Succinct*, in a way that the verification process and the proof size are polynomial in the size of the problem input.
- They are *Non-Interactive*, so the Prover and Verifier don't need to communicate synchronously, it's only needed a message from the Prover (called proof or argument), which the Verifier can validate off-line. This is also called the "public verifier" property, very useful in blockchain.

- They are *ARguments*, unlike the *proofs* from ZKPs. In a ZKP, a Prover can be considered computationally unbounded. In an argument, the Prover is considered limited in polynomial time. This means that an argument has only *computational soundness*, instead of *perfect soundness*. Although, the *completeness* property is the same as in Zero-Knowledge Proofs, meaning that an honest Prover can always perform a valid argument.
- They are *of Knowledge*, forcing the Prover to know a *witness* in order to being able to construct the argument.

For example, in a cryptocurrency transaction, the validation computation is a function $f(\omega_1, \omega_2, s, r, v, p_s, p_r) = 1 \Leftrightarrow \omega_1$ and ω_2 are the roots of the Merkle-trees (before and after the transaction), s is the sender and r the receiver account, and p_s and p_r are Merkle-tree proofs that show that the balance of s is at least v in the ω_1 state, and it hashes to ω_2 when said v amount is transferred to r .

A zkSNARK is created where only ω_1 and ω_2 are public values, and the tuple (s, r, v, p_s, p_r) is the witness string of knowledge from the Prover. Thanks to the zero-knowledge property, no node acting as Verifier will be able to know who sent how much money to whom, but allows to check validity of the transaction, avoiding the double spending problem.

The four main *ingredients* of a zkSNARK are [57]:

(A) Encoding as a polynomial problem

A given program is transformed to a quadratic equation of polynomials, i.e. to the form of the product $t(x)h(x) = w(x)v(x)$. The equality holds if the computation is correct. The objective is for the Prover to convince the Verifier that the equality holds.

(B) Succinctness by random sampling

Instead of performing polynomials multiplication and equality checking, using the Schwartz–Zippel Theorem we can reduce it to multiplication and equality on a single point s that the Verifier chooses and keeps secret. Therefore, the proof size and verification time are reduced to $t(s)h(s) = w(s)v(s)$.

(C) Homomorphic encryption

An encryption function E with homomorphic properties is used to hide secret values, but allowing to operate with them. The Prover will be able to compute $E(t(s))$, $E(h(s))$, $E(w(s))$ and $E(v(s))$ without the value s , but its encryption $E(s)$, and some other encrypted powers of s .

(D) Zero-Knowledge

The Prover adds statistical Zero-Knowledge by multiplying $E(t(s))$, $E(h(s))$, $E(w(s))$ and $E(v(s))$ with a random value, so the encrypted values are obfuscated but the Verifier can still check the equality.

That is the structure of zkSNARKs, but in the following subsections we will study how they are actually built.

4.2.1 Homomorphic encryption

Homomorphism comes from *homos*, meaning same, and *morphe*, meaning form or shape, and refers to the operation that preserves the structure between two algebraic structures, like two groups or rings. Therefore, a homomorphic encryption, also called homomorphic hiding or encoding, is a type of cipher that preserves the structure between the cleartext and the ciphertext spaces.

For example, the RSA encryption is *multiplicatively homomorphic*:

Given an RSA public key (e, n) and private key (d) , i.e. integers that validate $n = p \cdot q$ with p and q prime and $d \cdot e \equiv 1 \pmod{\phi(n)}$. The encryption of a message x is given by $E(x) \equiv x^e \pmod{n}$. The group multiplication is then a homomorphic property of the RSA encryption: $E(x)E(y) = x^e y^e \equiv (xy)^e \pmod{n} = E(xy)$.

As shown above, be $E()$ the RSA encryption scheme, $E(x) \cdot E(y) = E(x \cdot y)$ is the multiplicative homomorphism property, the product of the ciphered values equals the encryption of the product on the plaintext values.

Although there exist homomorphic encryption schemes like the Fan-Vercauteren (FV) scheme[], which is, to a certain degree, additively and multiplicatively homomorphic, as well as considered quantum-resistant, they are not practical yet because of their complexity.

Supposing that the discrete logarithm problem is hard, and knowing that exponents add up when the elements are multiplied, we can use the following function: $E(x) = g^x$, for g a generator¹ of a chosen group. It is additively homomorphic, as $E(x) \cdot E(y) = g^x \cdot g^y = g^{x+y} = E(x+y)$.

But in zkSNARKs we will need homomorphic addition and at least a limited form of multiplication. For that, we will combine the previous additively homomorphic hiding with Elliptic Curve pairings, which will give us arbitrary additions, and one multiplication. Let's briefly introduce elliptic curves and their pairings.

Given the field of p elements \mathbb{F}_p for a prime p , and the Weierstrass equation $Y^2 = X^3 + u \cdot X + v$, for u and v in the field, the pairs $(x, y) \in \mathbb{F}_p^2$ of coordinates in \mathbb{F}_p , plus a special point \mathcal{O} called the infinity, are the set of points of an elliptic curve. Intuitively, Figure 4.5 shows how the addition of points in elliptic curves work. Given the two points of the curve, P and Q , to add, *draw* the line between them and the addition of all the points of the curve that intersect the line must be the infinity point, which acts as the identity element, or zero in the additive notation. We don't need to enter in any more detail, only understand that we have an additive group.

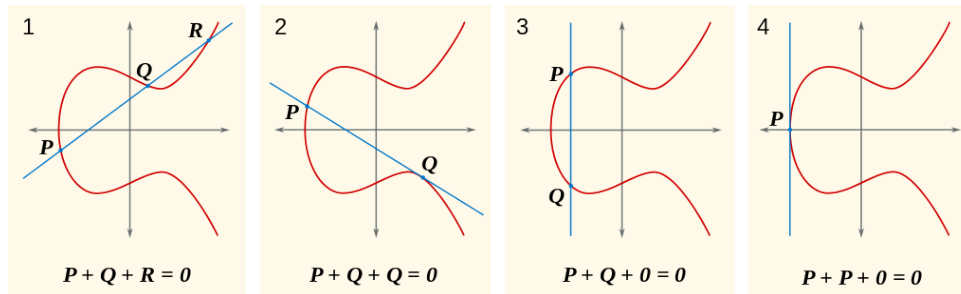


FIGURE 4.5: Elliptic curves addition.

Let's call the group of the elliptic curve with coordinates in \mathbb{F}_p as G_1 . Assume that there are r different elements in G_1 , for r a prime different from p . This is many

¹A generator g is a group element such that, given the group order n , the list g^0, g^1, \dots, g^{n-1} contains all the elements in the group exactly once.

times the case, and it means that any element $g_1 \in G_1$ different from \mathcal{O} is a generator of G_1 .

The smallest integer k such that r divides $p^k - 1$ is called the *embedding degree* of the curve. It is conjectured that when k is not too small, then the discrete logarithm problem in G_1 is very hard. This is equivalent to the discrete logarithm in the integers modulo a prime being hard, but in this case, because the group is additive, the problem is to find α given the values g_1 (the generator) and $\alpha \cdot g_1$.

For simplification in the rest of the zkSNARK explanation, we will consider that the elliptic curve and the pairing are chosen together such that be g a generator of the group and e the pairing that satisfies $e(g^x, g^y) = e(g, g)^{xy}$ for all points x and y .

As we have shown, the exponentiation of g is additively homomorphic, and we can express the identity as $e(E(x), E(y)) = e(E(1), E(1))^{xy}$. This gives us arbitrary additions, by multiplying the first or the second parameter by $E(z)$:

$$e(E(x) \cdot E(z), E(1)) = e(g^x \cdot g^z, g) = e(g^{x+z}, g) = e(g, g)^{(x+z)}.$$

And also one multiplication, by the property of the pairing function:

$$e(E(x), E(y)) = e(g^x, g^y) = e(g, g)^{xy}.$$

The combination of the functions E and e gives us the additively and (limited) multiplicatively homomorphic hiding we needed for zkSNARKs.

4.2.2 Quadratic Span Programs

The Zero-Knowledge Proofs section introduced complexity theory, explaining the set of problems in **P**, **NP** and **NPC**, with the latest being special as any other problem in **NP** could be reduced to be solved by a problem in **NPC**. Aside from the satisfiability problem, another **NPC** problem is the *Quadratic Span Program* (QSP) problem, that was showed by Gennaro et. al. to be well suited for zkSNARKs [27].

Because the QSP is in **NPC**, any other problem in **NP** can be reduced to it, therefore we could reduce any computation to a QSP problem, for example, the code to validate a transaction. The advantage of this technique is that it is much easier now to create a zkSNARK for any QSP, and express our code as a QSP, rather than work with arbitrary code to find a zkSNARK for it.

A Quadratic Span Program consists on a set of polynomials, where the Prover needs to compute a linear combination of the polynomials, with the property that this product is a multiple of another given polynomial. Furthermore, the binary representation of the input of the problem will restrict the polynomials allowed.

A QSP over a field F , for **inputs** of length n , consists of:

- polynomials $v_0, \dots, v_m, w_0, \dots, w_m$ over the field F
- the target polynomial t over F
- an injective function $f : \{(i, j) \mid 1 \leq i \leq n, j \in \{0, 1\}\} \longrightarrow \{1, \dots, m\}$

The task is to find a **linear combination** of the first set of polynomials such that the product is a **multiple** of the target polynomial t . For a given binary input u of n bits, the function f **restricts** the possible **factors** in the linear combination, taking the pair (i, j) that is the bit position in u and the value of $u[i]$ the i th bit.

An input u is accepted or verified by the QSP if and only if there are tuples $a = (a_1, \dots, a_m)$ and $b = (b_1, \dots, b_m)$ from the field F such that:

- $a_k, b_k = 1$ if $k = f(i, u[i])$ for some i

- $a_k, b_k = 0$ if $k = f(i, 1 - u[i])$ for some i
- t divides $v \cdot w$, where $v = v_0 + a_1v_1 + \dots + a_mv_m$ and $w = w_0 + b_1w_1 + \dots + b_mw_m$ are the linear combinations.

For input sizes smaller than half the number of polynomials, i.e. $2n$ smaller than m , there is enough freedom to choose the factors a_i, b_i . It works well for zkSNARKs when inputs are generally small. Also notice that the factor tuples a and b are the **NP** witness of the problem. A verifier can check that the polynomial t divides $v \cdot w$ in polynomial time, and can compute $v \cdot w$ in polynomial time given the witness pair a, b .

The verification can be even more efficient if the prover facilitates a polynomial h such that $th = vw$, which turns the task into polynomial equality, instead of division. Furthermore, the verifier will check that $th - vw = 0$, i.e., the difference is the zero polynomial.

But the multiplication of polynomials is still a hard task, given that the degree of the polynomials is roughly 100 times the number of logical gates of the original circuit. Instead of computing the linear combinations v and w , or the product $t \cdot h$, the verifier can choose a secret point $s \in F$, and computes the numbers $t(s), h(s), v_k(s)$ and $w_k(s)$ for all k , and then the linear combinations $v(s)$ and $w(s)$. The verifier then checks the equality of elements in F , not polynomials, $t(s)h(s) = v(s)w(s)$. So instead of polynomials additions and multiplications, the verifier performs field additions and multiplications.

Of course, checking on only one random point s reduces the security, but the prover could only cheat if he finds a zero of the polynomial $th - vw$. Given that the prover does not know s , and that the number of zeroes of the polynomial is small (the degree of the polynomial) compared to the possible values of s ($|F|$), in practice this is statistically safe.

4.2.3 Evaluate a polynomial succinctly and zero-knowledge

Before going further with the creation of a zkSNARK for the QSP problem, let's first study how to do the basic step of evaluating the polynomials in a given point, like the secret point s , but with Zero-Knowledge and succinctly.

For this, we will use the encryption function E and the pairing function e described in a previous section. Remember that a group is chosen, an elliptic curve, of order n and a generator g . The encryption is simply $E(x) := g^x$. The pairing function satisfies $e(g^x, g^y) = e(g, g)^{xy}$ for all points x and y .

The verifier randomly selects a secret field element $s \in F$, and sends to the prover the values:

$$E(1) = g, E(s) = g^s, E(s^2) = g^{s^2}, \dots, E(s^d)$$

where d is the maximum degree of all the polynomials. The value s should be forgotten now. If an attacker recovers s , and some other future secret values, it could create false proofs by finding the roots of the polynomials. In some blockchains, this is called *toxic waste*.

Now, a prover can compute $E(f(s))$ for any given polynomial f , without the value s , thanks to the homomorphic properties of E . To give an example, suppose $f(x) = 4x^2 + 2x + 4$. The prover wants to compute $E(f(s)) = E(4s^2 + 2s + 4) =$

$E(s^2)^4 \cdot E(s)^2 \cdot E(1)^4$, which can be calculated using the published values from before, without knowing s .

But now the verifier is not capable to validate the proof, as he deleted s . To solve this problem, before deleting s , the verifier chooses again a secret $\alpha \in F$, and publishes the *shifted* values:

$$E(\alpha), E(\alpha s), E(\alpha s^2), \dots, E(\alpha s^d)$$

and then deletes both s and α , the *toxic waste*.

Now, the prover also computes the value $E(\alpha f(s))$. In the example this is $E(4\alpha s^2 + 2\alpha s + 4\alpha) = E(\alpha s^2)^4 \cdot E(\alpha s)^2 \cdot E(\alpha)^4$, also computable with the published values from above.

The prover sends both $A := E(f(s))$ and $B := E(\alpha f(s))$.

The verifier now can verify the correctness of the computation with the help of the pairing function. Remember that the pairing function gave the homomorphic multiplication property, while the encryption function E only gave the homomorphic addition, used in the two previous steps. The verifier checks that $e(A, g^\alpha) = e(B, g)$. Note that the verifier knows g^α although he deleted α because it is the published value $E(\alpha) = g^\alpha$. The verification works because:

$$e(A, g^\alpha) = e(g^{f(s)}, g^\alpha) = e(g, g)^{\alpha f(s)}$$

$$e(B, g) = e(g^{\alpha f(s)}, g) = e(g, g)^{\alpha f(s)}$$

To protect against a cheating prover finding values A', B' that pass the test, there is the *d-power knowledge of exponent assumption*. Like the assumption that the discrete logarithm is hard, this assumption states that is unknown whether a cheating prover can find A', B' , but it is believed to be hard.

Technically, this protocol doesn't allow to verify if the prover evaluated the polynomial $f(x) = 4x^2 + 2x + 4$, but that the prover evaluated *some* polynomial at the given point s . In the following section, the zkSNARK for the QSP will include other values to verify that the correct polynomials were used for the proof.

The importance of this construction is that a verifier does not evaluate the full polynomial to confirm the proof, it suffices with the pairing functions equality check. Therefore, we have the succinctness. Now comes the Zero-Knowledge property.

The prover picks a random δ and instead of A , he sends the commitment $A' = E(\delta + f(s))$ and $B' = E(\alpha(\delta + f(s)))$. The random value hides the value of the polynomial evaluation, but thanks to the homomorphic properties of E , the prover can indeed compute the values A' and B' , and the verifier can check with the same method the correctness.

The prover calculates: $A' = E(\delta + f(s)) = g^{\delta + f(s)} = g^\delta g^{f(s)} = E(\delta)E(f(s)) = E(\delta)A$, and then $B' = E(\alpha(\delta + f(s))) = g^{\alpha\delta + \alpha f(s)} = g^{\alpha\delta} g^{\alpha f(s)} = E(\alpha)^\delta E(\alpha f(s)) = E(\alpha)^\delta B$. Because he knows δ , A , B and $E(\alpha)$, the prover can compute A' and B' .

The verifier already checked the equalities $A = E(a)$ and $B = E(\alpha a)$ for any value a , therefore, in this case $a = \delta + f(s)$ and the verification holds for the values A' and B' above.

Summing up, the prover can now compute the encrypted evaluation of a polynomial given an encrypted secret point s , without leaking any other information to the verifier. The next section applies this construction to create the SNARK for the QSP problem.

4.2.4 A SNARK of the QSP Problem

Once we have reduced the original program to a polynomial representation for the QDP Problem, these polynomials are **fixed**, v_k and w_k for all k and the target polynomial t .

Then, a setup phase follows, which generates the *common reference string* (CRS). The verifier generates this string from the secret field elements s and α , encrypting the evaluation of the polynomials at these points. We will also see another values of the CRS that will make verification more efficient and Zero-Knowledge.

To summarize up, in the QSP problem we are given polynomials $v_0, \dots, v_m, w_0, \dots, w_m$ for the linear combination, the target polynomial t of degree at most d , and an input u of size m bits. The prover calculates $a_1, \dots, a_m, b_1, \dots, b_m$, which are restricted depending on the input u , and a polynomial h to ease the verification such that:

$$th = (v_0 + a_1v_1 + \dots + a_mv_m)(w_0 + b_1w_1 + \dots + b_mw_m)$$

The verifier chooses the secret numbers s and α and publishes as the CRS the values:

- $E(1), E(s), E(s^2), \dots, E(s^d)$
- $E(\alpha), E(\alpha s), E(\alpha s^2), \dots, E(\alpha s^d)$

Because in this case we don't have a single polynomial, but a set, the CRS also includes the following values:

- $E(t(s)), E(\alpha t(s))$
- $E(v_0(s)), \dots, E(v_m(s))$
- $E(\alpha v_0(s)), \dots, E(\alpha v_m(s))$
- $E(w_0(s)), \dots, E(w_m(s))$
- $E(\alpha w_0(s)), \dots, E(\alpha w_m(s))$

And to verify that the polynomials used by the prover are indeed the correct ones, the verifier chooses the secret numbers β_v, β_w, γ and publishes in the CRS:

- $E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma)$
- $E(\beta_vv_1(s)), \dots, E(\beta_vv_m(s))$
- $E(\beta_wv_1(s)), \dots, E(\beta_wv_m(s))$
- $E(\beta_vt(s)), E(\beta_wt(s))$

These values are the full common reference string (CRS). The verifier must delete the secret values $s, \alpha, \beta_v, \beta_w$ and γ .

The **prover** now finds the polynomial h and the values $a_1, \dots, a_m, b_1, \dots, b_m$. For this, the prover uses the same reduction used to transform the original program to polynomial form, supposing it is a witness-preserving reduction, i.e., the witness of the NP problem that represented the original program is transformed to a witness of the QSP problem. The objective of the prover is to convince the verifier that said witness exists, without revealing information about it. For that, instead of sending the witness itself, the prover computes the values described below.

Remember that the input u restricted the possible values of a_k and b_k using an injective function $f : \{(i, j) \mid 1 \leq i \leq n, j \in \{0, 1\}\} \rightarrow \{1, \dots, m\}$. When m is relatively large, there are numbers which do not appear in the image of f , therefore, there exist not restricted indices. Let I_{free} be the set of unrestricted indices for the input u , and be $v_{free} = \sum_{k \in I_{free}} a_k v_k(x)$ the partial part of the polynomial v with free indices. The prover sends to the verifier the proof:

- $V_{free} := E(v_{free}(s)), W := E(w(s)), H := E(h(s))$
- $V'_{free} := E(\alpha v_{free}(s)), W' := E(\alpha w(s)), H' := E(\alpha h(s))$
- $Y := E(\beta_v v_{free}(s) + \beta_w w(s))$

As the previous section showed, all this values are computable knowing the encrypted values in the CRS. The last item will be used to check that the correct polynomials were used.

The **verifier** first computes the missing part of the v polynomial. Because the non-free indexes values were restricted univocally by the input u , the verifier can compute:

$$E(v_{in}(s)) = E\left(\sum_k a_k v_k(s)\right)$$

Where k ranges over the indices *not* in I_{free} , and the value is computable using the CRS.

Finally, the verifier checks the following equalities using the pairing function for the homomorphic (single) multiplication property:

- $e(V'_{free}, g) = e(V_{free}, g^\alpha)$
- $e(W', E(1)) = e(W, E(\alpha))$
- $e(H', E(1)) = e(H, E(\alpha))$
- $e(E(\gamma), Y) = e(E(\beta_v \gamma), V_{free}) \cdot e(E(\beta_w \gamma), W)$
- $e(E(v_0(s)) \cdot E(v_{in}(s)) \cdot V_{free}, E(w_0(s)) \cdot W) = e(H, E(t(s)))$

If the prover supplied a *correct* proof, the equalities hold. The first three checks come from the previous section of evaluating polynomials in a secret point s , using the values in the CRS.

The fourth equality, $e(E(\gamma), Y) = e(E(\beta_v \gamma), V_{free}) \cdot e(E(\beta_w \gamma), W)$, verifies the use of the correct polynomials v and w . The prover can not compute the encrypted combination $Y := E(\beta_v v_{free}(s) + \beta_w w(s))$ without using the exact values $E(v_{free}(s))$ and $E(w(s))$. This is because the values β_v and β_w of the encrypted linear combination are not in the CRS, neither their encrypted values. In the CRS there are only the *mixed* values:

- $E(\gamma), E(\beta_v \gamma), E(\beta_w \gamma)$
- $E(\beta_v v_1(s)), \dots, E(\beta_v v_m(s))$
- $E(\beta_w w_1(s)), \dots, E(\beta_w w_m(s))$
- $E(\beta_v t(s)), E(\beta_w t(s))$

If the prover provides a correct proof, the equalities hold because:

- $e(E(\gamma), Y) = e(E(\gamma), E(\beta_v v_{free}(s) + \beta_w w(s))) = e(g, g)^{\gamma(\beta_v v_{free}(s) + \beta_w w(s))}$
- $e(E(\beta_v \gamma), V_{free}) \cdot e(E(\beta_w \gamma), W) = e(E(\beta_v \gamma), E(v_{free}(s))) \cdot e(E(\beta_w \gamma), E(w(s))) = e(g, g)^{(\beta_v \gamma) v_{free}(s)} \cdot e(g, g)^{(\beta_w \gamma) w(s)} = e(g, g)^{\gamma(\beta_v v_{free}(s) + \beta_w w(s))}$

The last item the verifier checks is the main condition of the QSP problem in the point s , that $t(s)h(s) = (v_0(s) + a_1 v_1(s) + \dots + a_m v_m(s))(w_0(s) + b_1 w_1(s) + \dots + b_m w_m(s))$. Remember that the product of encrypted values translates in addition for the homomorphic property of E .

4.2.5 Zero-Knowledge for the SNARK

To add the Zero-Knowledge property to the previous SNARK, the prover *shifts* the values with a random secret value, balancing the equalities by applying the shift on both sides.

Let δ_{free} and δ_w be random values chosen by the prover. In the SNARK of the previous section, replace the values:

- $v_{free}(s) \longrightarrow v_{free}(s) + \delta_{free} \cdot t(s)$
- $w(s) \longrightarrow w(s) + \delta_w \cdot t(s)$

This way, the values V_{free} and W , which were an encoding of the witness of the QSP problem, are now indistinguishable from randomness, making impossible to extract the witness.

For the verification to work, the prover needs to *correct* the value H such that

$$(v_0(s) + a_1 v_1(s) + \dots + a_m v_m(s))(w_0(s) + b_1 w_1(s) + \dots + b_m w_m(s)) = h(s)t(s)$$

which can be expressed as

$$(v_0(s) + v_{in}(s) + v_{free}(s))(w_0(s) + w(s)) = h(s)t(s)$$

still holds.

The modifications mentioned before apply as:

$$(v_0(s) + v_{in}(s) + v_{free}(s) + \delta_{free} \cdot t(s)) \cdot (w_0(s) + w(s) + \delta_w \cdot t(s)) =$$

$$[h(s) + \delta_{free}(w_0(s) + w(s)) + \delta_w(v_0(s) + v_{in}(s) + v_{free}(s)) + \delta_{free}\delta_w t(s)]t(s)$$

Therefore, with this last replacement

- $h(s) \longrightarrow h(s) + \delta_{free}(w_0(s) + w(s)) + \delta_w(v_0(s) + v_{in}(s) + v_{free}(s)) + \delta_{free}\delta_w t(s)$

the verification will work.

4.2.6 Encoding a problem as a polynomial

The principle that allows to express certain problems as polynomials comes from the fact that the QSP problem is in **NPC**. Much like with the satisfiability problem (SAT), where any problem in **NP** could be expressed with a boolean formula, in this case, the idea is to express a problem as a polynomial with zeroes in the points where the problem is true.

$PolyZero(p) := 1$ if f is a polynomial with a zero and the variables belong to the set $\{0, 1\}$.

A SAT instance formula f can be reduced to the *PolyZero* problem such that when $f(a_1, \dots, a_k)$ is *true*, the polynomial $r(f)$ evaluates as zero in $r(f)(a_1, \dots, a_k)$. The four simple rules that transform a boolean formula to a valid polynomial of *PolyZero* are:

- $r(x_i) := (1 - x_i)$
- $r(\neg f) := (1 - r(f))$
- $r(f \wedge g) := (1 - (1 - r(f))(1 - r(g)))$
- $r(f \vee g) := r(f)r(g)$

For example, the formula $((x \wedge y) \vee \neg x)$ is translated to $1 - (1 - (1 - x))(1 - (1 - y))(1 - (1 - x)) = 1 - x^2y$, that is only zero when the original formula is true.

The four previous rules also allow to transform a valid witness from the $\{\text{true}, \text{false}\}$ space to the binary $\{0, 1\}$ set where the polynomials are evaluated. This is referred as a witness-preserving reduction.

The idea of using polynomials for interactive proof systems comes from Lund, et. al. [42]. The idea is to construct an arithmetic circuit of the program, and then transform it to a polynomial. The method to transform a problem to an arithmetic circuit depends on the given problem.

Let's see another example, but now, instead of a boolean formula of the SAT problem, the prover wants to proof that she knows $c_1, c_2, c_3 \in F$ such that $(c_1 \cdot c_2) \cdot (c_1 + c_3) = 7$.

First, we must represent the above computation with inputs $c_1, c_2, c_3 \in F$ as an arithmetic circuit. An arithmetic circuit consists of gates computing arithmetic operations like addition and multiplication, with wires connecting the gates. Figure ?? shows the arithmetic circuit of our example.

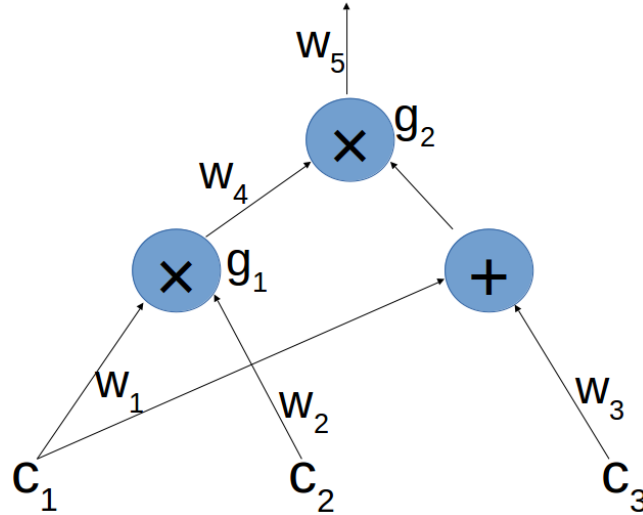


FIGURE 4.6: Arithmetic circuit for $(c_1 \cdot c_2) \cdot (c_1 + c_3)$

The bottom wires are the inputs, the top wire the output of the computation. The wires are named if they go out of an input or a multiplication, as well as only the product gates are named. A *legal assignment* for the circuit is an assignment of values to the labeled wires where the output value of each multiplication gate is indeed the product of the corresponding inputs. For this circuit, a legal assignment is the a tuple

c_1, \dots, c_5 such that $c_4 = c_1 \cdot c_2$ and $c_5 = c_4 \cdot (c_1 + c_3)$. The prover wants to convince the verifier that he knows a *legal assignment* $(c_1, c_2, c_3, c_4, c_5 = 7)$.

This arithmetic circuit is now transformed into an instance of the QSP problem.

Associate each multiplication gate to an element of the field F_p . For example, g_1 to 1 and g_2 to 2. The set $\{1, 2\}$ are the *target points*.

The next step is to define the sets of *left-wire* polynomials, L_1, \dots, L_5 , *right-wire* polynomials R_1, \dots, R_5 , and *output-wire* polynomials O_1, \dots, O_5 . The idea for the definitions is that the polynomials will usually be zero on the target points, except the ones involved in the target point's corresponding multiplication gate.

Let's start with the gate g_1 . The left input wire is w_1 , the right input wire is w_2 and the output wire is w_4 . We search for a polynomial that evaluates to zero on the target point 2, corresponding to the other gate: $L_1 = R_2 = O_4 = 2 - X$, is indeed zero on the point 2, and one on the point 1. The assignments to the polynomials correspond to the indexes of the wires in the gate.

Now with the gate g_2 , the left input wire is w_4 , the right input wires are both w_2 and w_3 , and the output wire is w_5 . We search for a polynomial that evaluates to zero in the target point 1 and non-zero in 2: $L_4 = R_1 = R_3 = O_5 = X - 1$.

There are no more gates in the arithmetic circuit. The rest of the polynomials (L_i, R_i, O_i) are set to the zero polynomial.

Using these polynomials, we create new left, right and output polynomials as:

- $L := \sum_{i=1}^5 c_i \cdot L_i = c_1 \cdot (2 - X) + c_4 \cdot (X - 1)$
- $R := \sum_{i=1}^5 c_i \cdot R_i = c_1 \cdot (X - 1) + c_2 \cdot (2 - X) + c_3 \cdot (X - 1)$
- $O := \sum_{i=1}^5 c_i \cdot O_i = c_4 \cdot (2 - X) + c_5 \cdot (X - 1)$

where the coefficients $(c_1, c_2, c_3, c_4, c_5)$ represent an assignment of the circuit to some input values.

Finally, we define the polynomial $P := L \cdot R - O$. This polynomial satisfies that: $(c_1, c_2, c_3, c_4, c_5)$ is a *legal assignment* to the circuit if and only if P is zero on all target points.

In our case, for the set of target points $\{1, 2\}$:

- $P(1) = L(1) \cdot R(1) - O(1) = [c_1 \cdot (2 - 1) + c_4 \cdot (1 - 1)] \cdot [c_1 \cdot (1 - 1) + c_2 \cdot (2 - 1) + c_3 \cdot (1 - 1)] - c_4 \cdot (2 - 1) + c_5 \cdot (1 - 1)$
 $= c_1 \cdot c_2 - c_4$
- $P(2) = L(2) \cdot R(2) - O(2) = [c_1 \cdot (2 - 2) + c_4 \cdot (2 - 1)] \cdot [c_1 \cdot (2 - 1) + c_2 \cdot (2 - 2) + c_3 \cdot (2 - 1)] - c_4 \cdot (2 - 2) + c_5 \cdot (2 - 1)$
 $= c_4 \cdot (c_1 + c_3) - c_5$

Both statements correspond to the equalities $c_4 = c_1 \cdot c_2$ and $c_5 = c_4 \cdot (c_1 + c_3)$ that made a legal assignment of our arithmetic circuit.

Now we can simplify the verification using the simple property that a polynomial P is zero in a point $a \in F$, $P(a) = 0$, if and only if the polynomial $X - a$ divides P . Therefore, $P = (X - a) \cdot H$, for some polynomial H .

If a legal assignment is the one that makes P zero in all the target points, we know that the polynomial P is divisible by the polynomials $X - a$ for all a in the target points set. The product of all these polynomials are the *target polynomial*, as we aim to make it zero in all target points. For our example, the target polynomial is:

$$T(X) = (X - 1) \cdot (X - 2)$$

which divides P if and only if $(c_1, c_2, c_3, c_4, c_5)$ is a legal assignment.

With all this information, we already have our instance of the QSP problem of degree d and input size m , given the polynomials $L_1, \dots, L_m, R_1, \dots, R_m, O_1, \dots, O_m$ and the **target** polynomial T of degree d .

An assignment (c_1, \dots, c_m) satisfies the instance if T divides $P := (\sum_{i=1}^m c_i \cdot L_i) \cdot (\sum_{i=1}^m c_i \cdot R_i) - (\sum_{i=1}^m c_i \cdot O_i)$. Rearranging the polynomials, we can get the form given in the previous section of $t \mid vw$.

In the example, the prover would perform the zkSNARK of the QSP for an assignment $(c_1, c_2, c_3, c_4, c_5 = 7)$.

4.2.7 Using zkSNARKs for shielded transactions

The previous sections explained in detail each step to create a zkSNARK. In this section, we put all the steps together to see an example of how a currency transaction can take place. In particular, the transaction will be based on the Zcash blockchain transactions.

Chapter 2 showed how a Bitcoin transaction was validated: using all the public information of the token's history, the Scripts verify that the sender address is the receiver address in a previous transaction and knows the private key, therefore he is the current owner of the unspent token. This solves the double spending and the ownership problems, using all public information.

Zcash uses zk-SNARKs to prove that the conditions for a valid transaction have been satisfied without revealing any crucial information about the addresses or values involved. The sender of this shielded transaction constructs a proof to show that, with high probability [71]:

- The input values sum to the output values for each shielded transfer.
- The sender proves that they have the private spending keys of the input notes, giving them the authority to spend.
- The private spending keys of the input notes are cryptographically linked to a signature over the whole transaction, in such a way that the transaction cannot be modified by a party who did not know these private keys.

The same three key points a Bitcoin node verifies. Aside, a Bitcoin node tracks the unspent transaction outputs (UTXOs). In Zcash, the equivalent are the commitments and nullifiers of transactions. Every new transaction has a commitment hash, and every spent transaction has a nullifier hash. The hashes protect against information disclosure of to correlate a commitment and a nullifier.

The commitment of a new transaction consists on the hash of the receiver address, the amount sent, a value ρ unique to this transaction (used later to open the commitment with a nullifier), and a random nonce r .

Commitment = HASH(recipient address, amount, ρ , r)

Later, the sender would use his spending key to create a nullifier, the hash of the secret ρ from an unspent commitment, and a ZKP stating that he is indeed authorized to spend it. To track spent transactions, the nodes track existing nullifiers. If the newly provided nullifier already exists, the transaction is rejected. It is like the opposite of the UTXOs tracking, a nullifier hash tracks spent transactions.

Nullifier = HASH(spending key, ρ)

The ZKP will assert that:

- For each input transaction, a revealed commitment exists.

- The nullifiers and transactions commitments are computed correctly.
- It is infeasible for the nullifier of an output transaction to collide with the nullifier of any other transaction.

From the succinctness property of zkSNARKs, a node creating the transaction does most of the computational work. In Zcash a transaction can take up to 40 seconds to create. But any other node can verify it in a matter of milliseconds, a great benefit for blockchain, where the performance of the network depends on the validation of transactions, rather than their creation.

The privacy of Zcash's shielded transactions comes from the use of zkSNARKs to prove the transaction is valid, and the commitments and nullifiers systems to prove the transaction is unspent. In both cases, the property of Zero-Knowledge is key, and opposite to the Bitcoin naive solution of making all the information public to validator nodes.

4.2.8 The applicability problems of zkSNARKs

With zkSNARKs we could not only perform secret computations, but make them efficiently verifiable by anyone. This would in theory make Smart Contracts based blockchains privacy preserving and even more efficient, as other nodes should not repeat the computation to validate the transaction, but only verify the zkSNARK.

Nonetheless, most of the Smart Contracts blockchains are based on the Turing complete Ethereum Virtual Machine. To implement a zkSNARK Verifier in a Smart Contract, it would require more *gas* (the measure for Smart Contract's execution) than is currently available in a single Ethereum block [57].

Another related problem is to create a new zkSNARK system for each Smart Contract code. Existing zkSNARK systems, like Zcash, use the same computation for every task, the transaction verification, this involves a unique zkSNARK system. As the previous sections have shown, every new zkSNARK system requires a new secret trusted setup phase. Zcash performed it at the bootstrapping of the chain. For a new zkSNARK of a new Smart Contract, a new trusted CRS should be created.

To summarize, even if the parties interested in a certain Smart Contract trust a certain setup phase, the computational cost of a zkSNARK verifier in a Smart Contract is prohibitively expensive. As mentioned in [57], one possibility to allow the execution of the verifier would be to include certain pairing functions and elliptic curve multiplications in the EVM, to reduce the *gas* cost of the Smart Contract. The other possibility is the current work of improving the EVM performance to allow arbitrary elliptic curves and pairing functions.

4.2.9 STARKs

The most recent research in interactive proofs promises a new solution that avoids the necessity of a trusted setup phase to create the CRS. This would solve the trust problems in blockchains, as shown in Zcash.

These new protocols have the same properties that zkSNARKs have (Zero-Knowledge, Succinctness, Non-interactive, Argument, of Knowledge), and a new property called *transparency*. A protocol that is transparent is that where the setup and verifier queries are *public* random coins, i.e. they don't depend on secret values that must be deleted for the security of the system [7].

The origin of STARKs relate to the research of Scalable Computational Integrity and Privacy (SCIP), from Michael Riabzev, et. al. It is still a work in progress

that promises post-quantum security, ZK, succinctness, scalability, etc. The name evolved, thanks to the popularity of SNARKs, to STARKs: Succinct **T**ransparent **A**RGument of Knowledge.

4.3 Ring Signatures

Given a group of members with private and public keys, a ring signature is a type of digital signature performed by one of the members of the group, but the signature itself does not reveal who signed it. The name comes from the shape representing the signature, not the algebraic structure.

Given the set of public and private key pairs of the members, $(P_1, S_1), \dots, (P_n, S_n)$, one can compute the signature over the message m with only the input $(m, S_i, P_1, \dots, P_n)$, where the only private key needed is one from anyone in the group (the anonymous signer), and the rest are all public keys. To verify the signature, it is only required the public keys from all members.

The mathematical idea behind the ring signature is that there exists a function that can be computed with only the public keys (verification), but knowing a private key allows to choose a value that makes the function output a desired specific value (signature).

The signature process consists on applying the function recursively to the previous calculated value, plus a seed, starting with a random *glue* value v ; with the help of the private key, one of those random seeds is overwritten with a chosen seed, with the objective that the final computed value of the function is equal to v , closing the “ring”.

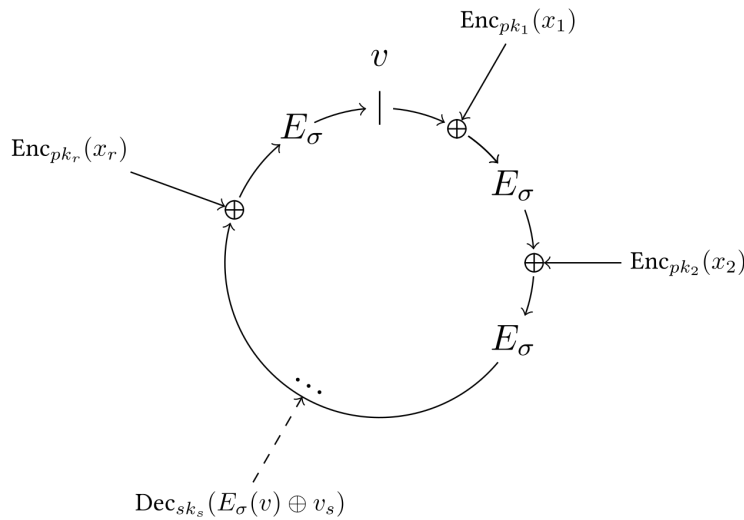


FIGURE 4.7: Rivest, Shamir, Tauman ring signature scheme.

Anyone can verify that a ring signature is valid given the public keys and seeds used in the computation, checking that the first value v equals the last computed value. But it is infeasible to tell which “link” of the ring used the private key to choose the seed, instead of using a random one.

Let’s see the ring signature, e.g. based on RSA (discrete logarithm), on the message m with public keys P_1, \dots, P_n and private key S_s of the signer:

First, let’s define the ring equation. There is a function $C_{k,v}(y_1, y_2, \dots, y_n)$ which is solvable for any single input, but it is infeasible to solve $C_{k,v}(g_1(x_1), g_2(x_2), \dots, g_n(x_n))$

given the values x_1, x_2, \dots, x_n , where each function g_i is the asymmetric encryption using the public key P_i , if the adversary does not know any of the private keys to invert at least one of the functions $g_i(x_i)$. The equation $C_{k,v}(y_1, y_2, \dots, y_n) = v$, where v is the *glue* value, is called the ring equation:

$$C_{k,v}(y_1, y_2, \dots, y_n) = E_k(y_n \oplus E_k(y_{n-1} \oplus E_k(\dots \oplus E_k(y_1 \oplus v) \dots))) = v$$

where E_k is a symmetric encryption function using the key k . This function is trivially invertible given $n - 1$ parameters.

Suppose we are given $y_1, \dots, y_{s-1}, y_{s+1}, \dots, y_n$, and the values k and v are also public. Because we know k , we can decrypt the outer function E_k :

$$y_n \oplus E_k(y_{n-1} \oplus E_k(\dots \oplus E_k(y_1 \oplus v) \dots))) = E_k^{-1}(v)$$

Then inverting $y_n \oplus$ in both sides:

$$E_k(y_{n-1} \oplus E_k(\dots \oplus E_k(y_1 \oplus v) \dots))) = y_n \oplus E_k^{-1}(v)$$

Repeating the process until the s^{th} encryption function E_k :

$$y_s \oplus E_k(y_{s-1} \oplus E_k(\dots \oplus E_k(y_1 \oplus v) \dots))) = E_k^{-1}(y_{s+1} \oplus \dots \oplus E_k^{-1}(y_n \oplus E_k^{-1}(v)) \dots)$$

Because we know k , v and y_{s+1}, \dots, y_n , we can solve the right hand side. And because we are also given y_1, \dots, y_{s-1} , we can solve in the left hand side what is at the right of y_s . Therefore, we can solve the ring equation for y_s . If we also know the private key to invert g_s , we can solve $y_s = g_s(x_s)$ for x_s .

Let's see now how to generate a ring signature, i.e. a valid ring equation, on the message m with public keys P_1, \dots, P_n and private key S_s of the signer:

1. Calculate the digest of the message that will serve as the key for a symmetric encryption function E : $k = \mathcal{H}(m)$.
2. Choose at random the *glue* value v .
3. Choose at random $n - 1$ values x_i , one for each ring member but the signer. The index x_s will denote the signer's x value.
4. Calculate $y_i = g_i(x_i)$ for all $i \neq s$.
5. Solve the ring equation as showed above, for the missing value y_s .
6. Using the signer's private key, calculate the missing value $x_s = g_s^{-1}(y_s)$.

The ring signature is now the tuple: $(P_1, P_2, \dots, P_n; v; x_1, x_2, \dots, x_n)$.

To verify the signature given $(P_1, P_2, \dots, P_n; v; x_1, x_2, \dots, x_n)$ on the message m :

1. Apply the asymmetric encryption on all x_i : $y_i = g_i(x_i)$, using the public keys included in the ring signature.
2. Calculate the symmetric key $k = \mathcal{H}(m)$.
3. Verify the ring equation $C_{k,v}(y_1, y_2, \dots, y_n) \stackrel{?}{=} v$.

	Commitment Scheme	ZKP	Homomorphic Hiding	zkSNARK	STARK	Group Signatures	Ring Signatures
Trusted Third Party	No	No	No	Setup Phase	No	Group manager	No
Universality	No	Yes	No	Yes	Yes	No	No
Non-interactive	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Succinct	*	*	*	Yes	Yes	*	*
Transparent	*	*	*	No	Yes	*	*
Zero-Knowledge	*	Yes	*	Yes	Yes	*	*
Post-Quantum resistant	*	*	*	No	Yes	*	*
Transaction's privacy	Yes*	Yes	No	Yes	Yes	No	Yes
Proof of execution	No	Yes	No	Yes	Yes	No	No
Maturity	High	High	High	High	Low	High	High

TABLE 4.1: Comparison of cryptographic solutions

There exists an extension called *linkable ring signatures* [41], where given two signatures computed with the same private key, that fact can be detected, but not which key is linked.

The predecessor of ring signatures is known as group signatures, where given a group of users, anyone on the group could also sign a message knowing only its private key. The downside to group signatures was the necessity of a group manager, whom was able to de-anonymize any group signature. Ring signatures do not require of such group manager, they only need the public keys of the rest of the members in the ring.

In Blockchain, ring signatures are applied to conceal the sender's identity, using several public keys at random already present in the chain, without the need of a special node that actively participates in the transaction to add privacy. Equivalent to the Zero-Knowledge statement of "I possess the private key of this public key", signing a transaction means "I possess at least one private key from this set of public keys". To protect the recipient node's identity, after each transaction a new public key is generated, and using a key-exchange algorithm like Diffie-Hellman's, the recipient is the only one who can recover the private key. Thanks to the linkability property of some ring signatures, the double spending can be detected, while still not revealing who tried to do it.

4.4 Comparison of technologies

Table 4.1 showcases the some differences and common points of the cryptographic technologies studied.

Notice how many of the most common properties are related to how blockchain works. For example, Non-Interactivity is crucial in a distributed ledger, where each node will validate a proof independently of when the prover created it. Universality refers to how the technology can be applied to *any* problem, in this case, ZKPs, zkSNARKs and STARKs can be applied to proof problems in **NP**, but homomorphic hiding can only hide operations over the algebraic structure. When a * symbol appears, it means that the property may or may not be achieved depending on the specific construction, or that the property is not directly evaluable for the technology. For example, discrete logarithm homomorphic hiding is not post-quantum resistant, but Fan-Vercauteren's homomorphic encryption scheme is post-quantum resistant, although the first allows any number of homomorphic operations, meanwhile the second one is limited depending on some system parameters.

Commitment schemes are a tool for creating Zero-Knowledge Proofs, but are also useful to bind a user to some secret information in the blockchain, giving trust

to the validators and privacy to the user. Also, no commitment scheme could be unconditionally binding or hiding, at least one property was at most computationally limited. Therefore, because blockchain is an immutable technology, when choosing a commitment scheme we must decide whether to unconditionally bind the user, but allowing for a future sufficiently powerful computer to break the commitment, or to prioritize privacy, trusting that the user will not be capable to open the commitment with another value in the future, and that value can corrupt the well functioning of the chain.

Although Zero-Knowledge Proofs are an interactive protocol, thanks to Fiat-Shamir's heuristic they can be applied as Non-Interactive, therefore, applicable to blockchain. They are the foundations for Anonymous Credential Systems. A blockchain can act as a distributed PKI without revealing private information about individual users, and giving trust to off-chain unlinkable proofs. ZKPs are also applicable to transactions, if proper constructions are used (like CL accumulators) so validation of transactions are efficient enough in size and time.

Homomorphic hiding is mostly used for privacy preserving multi-party computations, and, as have been shown, to create zkSNARKs. The homomorphism property between ciphered values usually affects only to one operation, and even obtaining homomorphism of two operands, they may be limited in the number of applications. The homomorphic hiding studied uses the discrete logarithm problem and elliptic curves' pairing functions. Fan-Vercauteren's homomorphic encryption is a more complex construction, which uses polynomials modulo other polynomials and approximation functions from reals to integers, but in exchange obtain post-quantum resistance.

The method to build a zkSNARK was to reduce the original problem to the QSP problem, which is in **NPC** and therefore any other problem in **NP** can be reduced to it. Then, playing with the QSP polynomials and homomorphic encryption we achieve evaluation of the polynomials in a secret point of the field. The simplification from validating polynomials equality to field elements equality brings succinctness to the scheme. Finally, adding some randomization we make the ensembles indistinguishable to add Zero-Knowledge. The main drawbacks are, on one side, the **ARGument** property. A sufficiently powerful prover can deceive the validators with a false proof. On the other hand, the need of a trusted setup phase, which needs to be done only once, but shifts all the trust to those nodes involved in the bootstrapping. Current blockchains made public their randomization distributed secret bootstrapping protocol to ensure their implementation is secure, even so, there exist the probability of collusion.

As a solution to the second problem, STARKs add the Transparency property, removing the need of any secret trusted setup, as any public values are random coin flips. Nonetheless, it is still a novel technology and is not broadly applied.

Finally, the applications of group or ring signatures are not as broad as the general proofs of ZKPs, zkSNARKs or STARKs, but ring signatures are a simple, yet effective way to hide or mix the signer of a transaction without the need of a trusted third party. The evolution from group signatures to ring signatures is somewhat equivalent to that of zkSNARKs to STARKs. Group signatures needed an entity called the group manager, who could deanonymize any signature. For blockchains, ring signatures allow independence of that manager, and any user can choose any set of signatures, according to the privacy degree they desire.

Chapter 5

Applications

As stated at the beginning of chapter 4, we have studied technologies that work within the blockchain. Solutions like Enigma make use of a parallel network that is not a blockchain, but is controlled by the blockchain, and can apply arbitrary solutions that fit their objectives, complementing the existing blockchain technology without its restrictions. With this chapter, we have gained knowledge about the most powerful privacy preserving solutions for blockchain without adding those parallel infrastructures. Therefore, when we begin designing new blockchain solutions, we can decide when it is actually necessary to create off-chain entities. This last section aims to show the design of two solutions, one for Internet of Things (IoT) and another for electronic government administration.

5.1 Internet of Things

IoT devices can act as autonomous agents performing their own transactions in blockchain Blockchain technology in IoT can provide:

- Chain traceability: Habilitates to track devices' history and auditing data exchange.
- Provenance of IoT items: e.g. materials used, where it was manufactured, ...
- IoT Products compliance: conditions during transportation, by automatically record temperatures, locations, ...
- Product Authenticity: validate whether or not the product is genuine.
- Provenance, auditable and verifiable IoT transactions (disputes can be checked and avoided).
- Trusted and reliable transactions.
- Decentralized approach without single point of failure.
- Immutability and transparency.
- Decentralized M2M (Machine to Machine) [micro-]transactions between devices.
- Access control rights can be enforced as blockchain transactions.
- M2M digital assets exchanges.
- Coordination between devices through blockchain.

- Execution of smart contracts for IoT devices, performing commands against a Service provider and other devices.
- Non-monetary IoT assets can be monetizable through tokenization through micro-transaction agents. E.g. online storage, energy...

After our previous works with Identity Mixer (Idemix) and IoT [60], we consider Idemix to be a protocol with many privacy preserving advantages, including data provenance applications by defining tailored presentation policies for each context. Data provenance consists on any additional information that helps to determine the derivation history of a particular data. When a privacy-preserving approach is needed, the data provenance metadata should allow unveiling the real identity of the owner associated to the exchanged IoT data, when the inspection grounds are met (e.g. identity theft or associated crimes). Besides, the data provenance information should be stuck to the data, enabling the tracking and auditing of it. However, currently there is a lack related to the application of proper privacy-preserving data provenance mechanisms for IoT scenarios that meets these requirements.

To add privacy to this set of IoT solutions, our proposal is to sign in a privacy-preserving way the IoT exchanged data, ensuring the ownership, anonymity, integrity and authenticity of the IoT data, prior its outsourcing.

The first step is to modify Idemix from an authentication protocol to a signature protocol. Although Idemix uses the Fiat-Shamir heuristic to reduce the number of messages exchanged during a proof, the protocol starts with a challenge from the verifier, to add freshness to the proof. In order to make a signature from a presentation policy, instead of using the verifier's nonce and the Fiat-Shamir hashing, we make use of the hash of the document to sign, together with the Fiat-Shamir heuristic. This is indeed a secure digital signature, because an Idemix Proof includes *authentication*, *non-repudiation* as it is linked to the CL signature in the Idemix credential, and *integrity* because otherwise the hashed value used in the challenge c would be altered and the proof would become invalid. Nonetheless, the authentication level depends on the presentation policy used, and that will depend on the policy administrator.

In smart cities scenarios, where IoT data provenance becomes very useful, the data producers will be the User's mobile devices, which also hold the secure wallet with the Idemix credentials issued by the Identity Providers (IdP). These devices will report IoT sensors data (e.g. location). The real user identity associated to the collected data needs to be preserved, and therefore, the Idemix proof performed during the signature will allow to demonstrate the possession of a valid Idemix certificate issued by a trusted IdP. The proof does not reveal any of user's attributes included in the credential. It may include a domain pseudonym (dNym) and make the user ID attribute as inspectable by the law enforcement authorities (which play the role of Inspector entity of Idemix). An inspector is capable to de-anonymize an Idemix ZKP, if the presentation policy explicitly states so and the user complies. The use of a domain pseudonym ensures that only one pseudonym can be used for a given domain, and that it is unlinkable to other pseudonyms derived from the same credential.

The second step to take in order to adapt Idemix onto privacy-preserving signature scenarios is to address the revocation by accumulators problem. This accumulator changes every time a credential is revoked, and every remaining valid credential must update itself to be capable of generating valid proofs. The verifiers are up to date with the accumulator value, and therefore a proof is valid as long as the verifier accepts its own challenge and the revocation accumulator does not change.

Instead of searching for revocation alternatives that are as privacy preserving as accumulators, we can break their problem into smaller pieces and identify the key characteristic that is drawing us back: order of events. Just like Bitcoin solved the order of transactions with a distributed time-stamping service, the blockchain, we can solve the accumulators problem by linking a signature ZKP to an accumulator value, and show that it was the last accumulator published *at the time of the proof*.

Therefore, our solution can be applied by having the Revocation Authority upload each new accumulator value to the blockchain at the very same time it is generated, with a digital signature in order to trust the announcement. When a IoT device registers new data, it signs it with a NI-ZKP, and uploads either the signature or a hash of it for enhanced privacy preserving, to the blockchain. When uploading the signature, the IoT device adds a reference to the transaction. By the time a consumer of the data verifies the signature, the accumulator value may have changed, but the transaction reference will show that at the time of the signature, the credential was valid.

For our example, let's consider a context broker, a service that stores IoT data in a structured manner, like FIWARE's Orion Context Broker. The user's device may upload location data as shown below. The domain is defined to create a dNym that anonymizes the user but ensures that there is a historic of the same one. The metadata includes the Identity Mixer signature and the reference of the time-stamp transaction.

```
{ "id": "DomainURI",
  "type": "Domain",
  "location": {
    "value": "41.3763726, 2.186447514",
    "type": "geo:point",
    "metadata": {
      "signature": {
        "type": "StructuredValue",
        "value": {
          "type" : "IdemixSignature",
          "msgHashAlg" : <HashAlg>,
          "timestamp" : <transactionReference>,
          "signatureValue" : <signature>
        }
      }
    }
  }
}
```

This data could now be consumed from the context broker to perform analysis. In the context of a smart city, location data with data provenance that relates it to a user's single history, is a powerful tool to understand the city flows of population, and can help making decisions on how to position future public services. And this all without revealing the user's identity thanks to ZKPs. The inspector entity could be included in this scenarios with the administrative protection of only disclosing the user's identity when an search warrant is issued, or under a catastrophe, like an earthquake, the data would help to identify victims during rescue missions.

Figure 5.2 showcases a complete infrastructure of the IoT data provenance and identity management. Identity Providers and credentials could be managed on-ledger, like the Sovrin project proposes. The pushed data from IoT devices could also be stored in a decentralized file system as IPFS or in another IoT platform. The unique URI of these platforms is then included in the transaction. The storage manager (e.g. IPFS) can sign the document (time-stamp) to proof it was uploaded. The

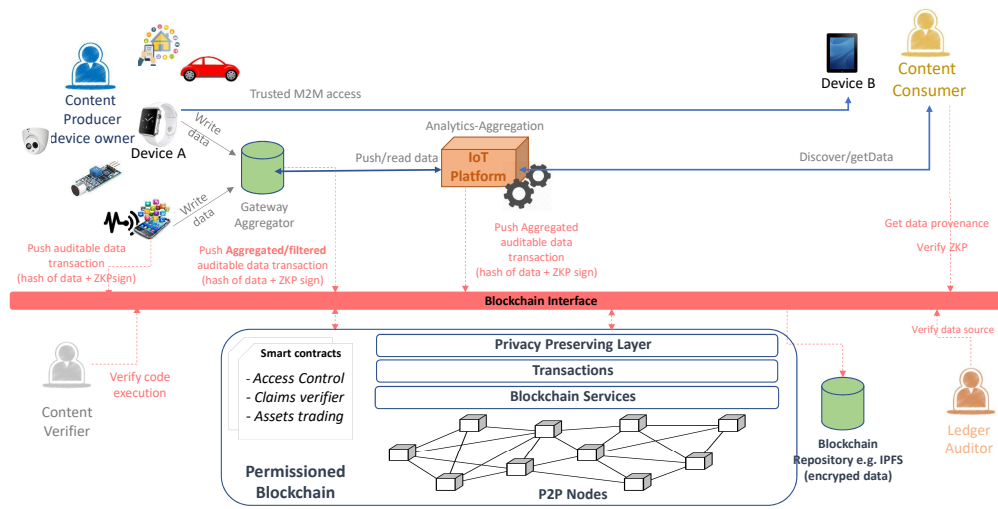


FIGURE 5.1: Envisaged blockchain-aware IoT ecosystem.

transaction could include this signature for auditability. Furthermore, to provide confidentiality, users and smart objects might encrypt the data in the IPFS storage. The producer can define the consumers who will be authorized to access the pushed exchanged data, using transactions with the policy in them.

A consumer reading the blockchain can look for the data transaction. Then, it can download the exchanged data record (or any digital assets) from the URI and verify the integrity and provenance information from the hash or ZKP signature in the transaction. If data was kept encrypted in IPFS, the consumer should be in possession of crypto-keys to decipher the data.

Also, some scalability and feasibility issues may appear in permissionless blockchains. Pushing billion of IoT transactions might require too much time or money. To solve this, IoT data transactions might be aggregated in gateways or trusted-domains, and then being pushed in the blockchain in one go. These gateways could even filter the IoT data and only publish transactions with uncommon IoT data measurements for traceability of singularities.

5.2 e-Government scenarios

Public administration is a trusted entity that keeps information about citizens and organizations, and that regulates the activities of both. Blockchain is a useful tool for interoperability between multiple entities.

Taking, for example, the introduction's scenario of a consortium of companies working together, the government should be able to regulate their fair actions and to respect their rights of privacy. If the consortium uses the blockchain for asset payments, and not only a log system, the government could be able to audit the

accounts. Using commitment schemes and homomorphic encryption, simple accounting aggregation could be done in the blockchain without revealing internal operations. Furthermore, zkSNARKs could be defined to ensure correct accounting inside the companies, revealing only the public results to the government. Remember that zkSNARKs had implementation issues if they were to be used in generic smart contracts, but given well defined common procedures, like the double spending proof in Zcash or accountability in this example, zkSNARKs can be applied with a single bootstrapping phase.

Commitment schemes are the base for interactive coin-flipping and voting (all participants commit to their votes while hiding the value). Given the trust properties of blockchain, voting systems based on privacy preserving authentication, e.g. with Idemix, and then commitment in transactions, allows for anyone to audit the process, ensuring that each user voted only once (via domain pseudonyms) and that they can not change the vote afterwards (unconditional binding). For smaller secret votes, there exists a variation of ring signatures known as *threshold ring signatures*, which are only valid if t out of n owners cooperate to generate it. Given a permissioned blockchain, i.e. authenticated users, the majority that cooperates and adds a transaction with a valid t -out-of- n signature, can keep in secret who participated in the process, while this small group accepts the signature and it is kept immutable in the chain.

Another scenario is, for example, property registration. Instead of an internal token managed by the network, legal documents that represent properties can be assigned a unique identifier, which may be hashed for privacy, and then included in the blockchain to track its history. With the use of privacy preserving signatures, property change can be registered and accounted in blockchain. When a dispute is opened, the government can open the hash from the specific identifier and track the history of the asset.

The research challenges are to endow the only trusted party, the government administration, with a traceable and trusted tool as blockchain, while ensuring the privacy of the citizens and companies.

Our envisaged collaboration with IBM aims to deal with “Trustworthiness in Digital Life”, investigating Privacy-preserving Identity Management for the public sector, including e-Government and higher Education. Their first demonstration scenario focuses on the expedition and usage of education certificates from institutes, universities, state agencies or private sector, all under the same standards. Graduates will be able to proof their competencies and qualifications while applying the data minimization principle. They shall not disclose private information until further phases in the job application.

The proposed scenario works around a decentralized platform powered by blockchain. Public entities certificates will be published to the chain for verification of the issuers’ keys, but not the graduates certificates. Certificate structures and claims schemes, for proof presentations, are also published in the chain for a common reference source and auditability. When an institution sets up their own Degree Certification System, the graduates are authenticated through their credentials and then they can set on the chain access control policies for companies to access a sub-set of their attributes.

With this solution, there are two major issues solved. On one hand, it allows people to legitimately, and with privacy preserving, share their degree and professional certifications. On the other hand, it prevents fraudulent claims that companies before had to investigate on their own.

Thus, the solution provides:

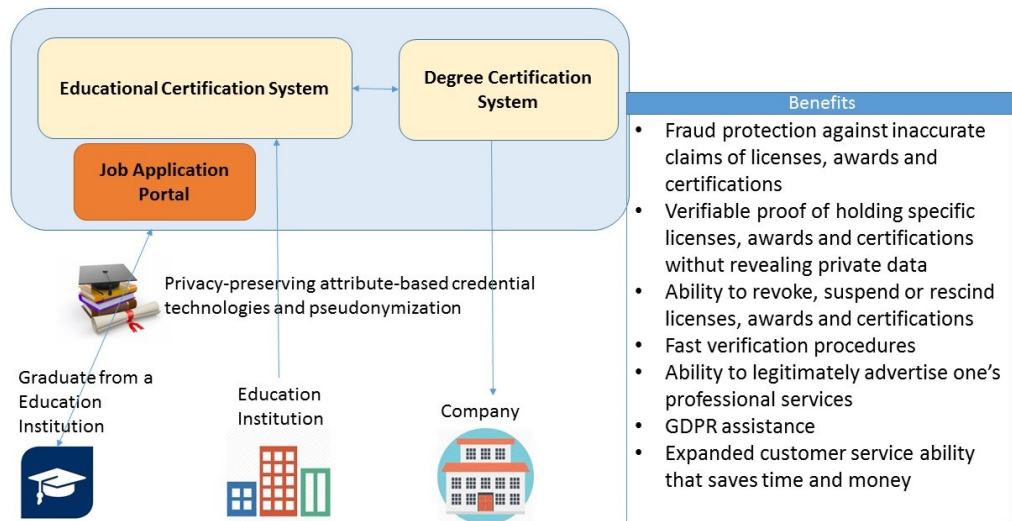


FIGURE 5.2: e-Government and education system overview

- Fraud protection against inaccurate claims of degree certificates and qualification documents.
- Anonymous Verifiable proofs of holding specific certificates or qualification documents.
- Ability to revoke, suspend or cancel certificates.
- Fast, off-line, verification procedures.

The standardization of this solution is carried in part by a W3C working group, focused on *Verifiable Claims* standard protocols and languages. The cryptography underlying these claims is based on the same ZKPs of Identity Mixer, and a new specification called *ZKLang* will transtale high-level claim descriptions into low-level Zero-Knowledge Proof specifications.

Chapter 6

Conclusions and future work

This master thesis has presented the insights of how the blockchain technology works, showcasing how the first design of Bitcoin did not address any privacy issues. After a survey of the current research proposals to improve privacy in blockchain, a series of common technologies arise as the most promising and versatile, reaching privacy milestones that other blockchains only apply with heuristics.

The mathematical foundations of these most promising technologies are explained. Zero-Knowledge Proofs are usually strong even with computationally unbounded provers, and the great research in the area allows to reuse multiple existing and efficient proofs for blockchain applications, which are in the most cases alternative cryptocurrencies. Then we studied the *magical* zkSNARKs, that allowed to verify in very little time that someone did indeed some computation. It has been shown that no *magic* is happening, but a reduction to an NPC problem, which happens to have very good polynomial properties to create a zkSNARK. But this study also let's us to *rise the curtain* to the danger of the *common reference string* (CRL), which depends on the trust on the verifier, or in the case of blockchain, the trust that the first set of nodes that bootstrapped the chain did not collude between them to save the secret values that generate the CRL. Finally, an introduction to a type of signatures that indeed anonymize the signer in a group of possible signers, without the need of any mixing system that randomize the origin of the signature. Ring signatures have great applications in blockchain thanks to the increasing set of public keys available to anonymize any signature, as in smaller scenarios, the sets of signatures needed to hide the signer may not be big enough.

Finally, we saw how broad applications blockchain technology can have while maintaining privacy preserving. From small IoT devices to e-government crucial applications, blockchain gives a common decentralized infrastructure while cryptography ensures the privacy rights of their users.

As future work, we are currently working to publish a survey paper on the multiple blockchain solutions, explaining their real advantages and disadvantages by studying their mathematical basis, in order to compare if each solution is actually capable of achieving the levels of privacy they assure. Also, we are working on proof of concepts of the applications mentioned at the end of the previous chapter.

Other future studies could focus on the heuristic and probabilistic techniques of the mixers, or even on how to apply other privacy techniques to blockchain. For example, differential privacy allows to aggregate data from any data source, with statistical variations that hinder the correlation of queries to reconstruct the data base, therefore obtaining the original data, but the aggregation of data is still statistically meaningful for any study. Data in blockchain is a chain of transactions duplicated by all nodes, which needs to be deterministic, therefore, differential privacy would not be applicable to the *insides* of blockchain. But a private permissioned blockchain shared between a set of companies with respective SLAs, could be used as a data

source with differential privacy anonymization for third parties studies that would benefit of it.

To finish this paper, as the author, I personally think that the compendium of all the foundations included in it is a powerful tool for anyone interested to begin researching blockchain. The documentation is scattered between papers and code, where many articles treat the mathematics as some *black magic* that has great properties and just works, and then apply them in their designs. Thanks to blockchain, many of this technologies that already existed thirty years ago can have practical applications, and it is important to know how they actually work in order to handle them securely.

Appendix A

Survey Paper Draft

In this appendix we include the draft for the survey paper on privacy on the block-chain.

Article

Privacy-preserving solutions for Blockchain: review and challenges

Jose Luis Canovas Sanchez¹, Jorge Bernal Bernabe ^{1*}, Jose L. Hernandez-Ramos², Rafael Torres Moreno¹ and Antonio Skarmeta¹

¹ Department of Information and Communication Engineering, University of Murcia, Spain; jorgebernal@um.es, joseluiscanovas@um.es, rtorres@um.es, skarmeta@um.es

² European Commission, Joint Research Centre, Ispra 21027, Italy; jose-luis.hernandez-ramos@ec.europa.eu

* Correspondence: jorgebernal@um.es

Academic Editor: name

Version June 25, 2018; Typeset by L^AT_EX using class file mdpi.cls

Abstract: As we move into the Big Data era privacy issues are getting more and more importance. User's digital transactions and their personal data are exposed to a massive management, sometimes without user awareness and control. To cope with these issues, the Blockchain can offer a security-by-design, distributed and verifiable ledger that records efficiently transactions between parties. Novel privacy-preserving solutions for blockchain are emerging to empower users with mechanisms to become anonymous and take control of their personal data during their digital transactions of any kind in the ledger, following a self-sovereign identity approach. This paper performs a systematic study of the current state of the art on privacy-preserving techniques and research solutions in blockchain, as well as the main associated issues and challenges in this promising and disrupting technology. The review covers both, privacy techniques in public and permissionless blockchains, e.g. Bitcoin and Ethereum, as well as privacy-preserving research proposals and solutions in permissioned and private blockchains.

Keywords: blockchain; privacy; security; survey; bitcoin

1. Introduction

The Big Data era is undermining the user's privacy in multiple digital scenarios. Large third-parties benefit from the management of their users data, by collecting, analyzing, correlating and controlling massive amounts of personal data. These organizations, and their services, are subject to security breaches and user data misuse, which might compromise users' privacy, even without user-awareness. Besides, individuals are given few options to control their personal data and their privacy during their transactions online, encompassing how, when, where, by whom, and which particular personal information is disclosed in each particular transaction.

This situation is aggravated with the adventure of IoT scenarios where billions of constrained smart objects, with scarce capabilities to enforce proper security mechanisms, strive to deal with cyber-attacks that might leak their handled data, and ultimately, sensitive and private information of their owners/users. Besides, in IoT, user privacy controls are difficult to apply, as the smart objects usually act on behalf of the user without user control and consent, undermining the adoption of the minimal personal disclosure principle.

In this context, emerging blockchain [1] proposals [] [] and platforms, such as uPort [2] or Sovrin [3], propose a decentralized ledger that empower users with mechanisms to preserve their privacy in their digital transactions. The management of user's related information in permissioned

Blockchains is being characterized by its privacy-preserving nature. With the rise of blockchain, Identity Management (IdM) systems are switching from traditional web-centric approach or identity federation approaches, towards the self-sovereign identity (SSI) paradigm [3].

Self-sovereign identities empower citizens to take control of their data in any-time in any online situation. Under this approach, user personal data is no longer kept in raw in third-parties services, neither in Service Providers or Identity Providers, and information regarding transactions and interactions of users in services can be anonymized by integrating anonymous credentials system like [4] in the blockchain platform. It avoids that third-parties can leak personal data, and, in the worst case, become a potential source of other, more important, risks, such as identity-related cybercrimes (e.g. identity-theft).

The desintermediation provided by blockchain is changing the democratization and universal access to tokenized digital assets of any kind (not only money), causing a revolution on disparate kind of scenarios [?], ranging from healthcare [5], cyber-physical systems [], Decentralized Internet of Things [], Vehicular transports systems [6], notary management [], to name a few. Nonetheless, still today, the main Blockchain applicability are Cryptocurrencies, that allows transferring securely money in a decentralized fashion using the Ledger, without central third-parties, while enabling public verifiability. Although the usage of pseudonyms avoids linking payments to the real identity, users are not totally anonymous in their transactions, since all movement of money behind these pseudonyms is traceable and linkable, specially when handling multiple-entries transactions with several addresses from various accounts belonging to the same user [7]. To cope with this issue, some proposals such as mixers services [8] [9] tries to provide a third party in charge of concealing a transaction within a big amount of unrelated transactions. Thus, critical information such as the payer, payee, or pay amount [10] can be fully anonymized [11], although some times at expenses of transaction delays and more costs.

Blockchain related scenarios and solutions are subject to many different threats and privacy issues [12] [13] [14] to ensure full anonymity and confidential issues [] to conceal transactions. Scenarios relying on Blockchain technologies need to face diverse privacy challenges in the upcoming years XX [], above all, when applied to certain cyber-phsyical and critical systems [].

This paper performs a systematic review of the current state of the art on privacy-preserving techniques and solutions in blockchain, as well as the main associated issues and challenges in this promising technology. Besides, the paper covers the study of both, privacy techniques in public and permissionless blockchain solutions e.g. Bitcoin [15] and Ethereum, as well as privacy-preserving research proposals and solutions in permissioned blockchains. To the best of our knowledge, this is the first review paper about privacy-preserving solutions for blockchain.

The rest of the paper is organized as follows. Section 2 overviews main concepts about privacy and blockchain and oversees the traditional privacy-preserving techniques. Section 3 reviews the main privacy concerns, issues and threats in blockchain scenarios, and lays down the privacy requirements and objectives. Section 4 surveys the current state-of-the-art about privacy solutions for both, permissioned and permissionless blockchains. Section ?? is devoted to describe the main privacy challenges in blockchain to be addressed in upcoming proposals and platforms. Section ??, presents our architecture for a privacy-preserving self-sovereign IdM system in blockchain. Finally, the conclusions are drawn in Section 5.

2. Privacy aspects in blockchain

Introducir privacidad y tecnicas tradicionales, ideas de aqui:

- - [16] provides a review on the current state of information privacy research in Information systems.
- - [17] Privacy enhancing technologies: A review
- - [18] review A critical review of 10 years of Privacy Technology

- [19] INFORMATION PRIVACY RESEARCH: AN INTERDISCIPLINARY REVIEW
- privacidad como confidencialidad (data confidentiality, anonymity in communications)
- privacy as control (minimal disclosure, ACS)
- hablar del hype actual de blockchain

2.1. Traditional Privacy-preserving solutions

- Privacy concept, anonymity, pseudonymity, ...
- traditional Identity management systems
- Anonymous credential systems

2.1.1. Blockchain concepts

Blockchain shifts trust from a classical centralized approach to a fully decentralized network of nodes. It is based on a synchronized Distributed Ledger Technology (DLT), which acts as a decentralized database, keeping the information replicated and shared among multiples nodes spread in remote locations.

The concept of blockchain was first introduced by Satoshi Nakamoto in [15] as the technical foundations of a new peer-to-peer version of electronic cash.

Gatteschi et al. [20] briefly defined the blockchain as “a public ledger distributed over a network that records transactions (messages sent from one network node to another) executed among network participants. Each transaction is verified by network nodes according to a majority consensus mechanism before being added to the blockchain. Recorded information cannot be changed or erased and the history of each transaction can be re-created at any time.”

Crosby et al. [1] defined the blockchain as a “distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system. Once entered, information can never be erased. The blockchain contains a certain and verifiable record of every single transaction ever made.”

The Decode project presents a more technical description of a blockchain [21]:

The building block of blockchain is the hash pointer. A hash pointer is simply some information (typically called transaction) along with a hash of the information. The hash serves to identify the information, and also allows verification of its integrity. A blockchain is a linked list of hash pointers, where usual pointers have been replaced with hash pointers. The first block in the chain points to a special block called the genesis block. Each block contains a hash of the previous block and information specific to the current block. A key result of iterative hashing is that a block implicitly verifies integrity of the entire blockchain before it. Thus given the current head of the blockchain, any party can independently verify the entire blockchain by generating hashes from the beginning of the chain up until the end. The final hash should match that in the head, otherwise the blockchain has been tampered with.

All definitions agree on a collection of concepts that shape what a blockchain is:

- *Transaction*: A single record in the ledger that can specify a piece of information or an operation over previous transactions, e.g. to send the funds from a previous transaction to another public address.
- *Block*: An group of transactions, establishing a chronological order between them. A block also includes a hash pointer to the previous block of the blockchain.
- *Hash pointer*: Given a transaction or a block their hash is included in the blockchain, which gives integrity and a means to point to the transaction or block, like a memory pointer to a variable. When a new block of transactions is added to the chain, the hash pointer of the previous block is appended. To compute the hash pointer of the new block, the hash pointer of the previous one must be included in the calculation.

- *Genesis block*: The first block in a blockchain, establishing a starting point for the linked list of hash pointers.
- *Chain*: The linked list of hash pointers from the genesis block to the last block. The chain determines the chronological order of all transactions, as well as integrity of the entire blockchain before it. One can verify the integrity of the chain by computing the hashes from the genesis block to the last one, if any hash pointer to the previous block differs from the one computed, the chain has been altered.
- *Merkle Tree* [22]: A binary tree where the leaves are the hash pointers of the transactions in a block, and each parent node is the hash of the two children nodes. The root of the Merkle tree is a hash that gives integrity to all transactions in a block, including their order within it. The complete block's hash pointer is the hash of the Merkle tree root, with the hash pointer of the previous block and any consensus information that makes the node valid. The Merkle tree allows to save space on disk by deleting transactions, but keeping part of the binary tree. When verifying integrity of the chain, the Merkle tree allows to compute the valid hash of the block, without having the entire transaction information on disk.
- *Network*: Referring to the *blockchain network*, it is the set of nodes that interact between them in a peer-to-peer (P2P) fashion, exchanging the blockchain data, adding transactions, validating them, and agreeing on what new blocks are added to the head of the chain.
- *Consensus*: The set of all transactions, in order, defines the state of the blockchain. When a new transaction is added, the state changes. In order for all the nodes of the network to agree on the state of the blockchain, they run the consensus algorithm over the new transactions.
- *Fork*: Depending on the consensus algorithm, the network may accept two blocks at the same height of the chain. This creates a fork of the chain. Part of the network may continue to add blocks using one of the forks, while the other part uses the other fork. Because the hash pointers of each block will be different, the chains are incompatible. The network must agree on which fork to use when they appear. In Bitcoin, the longest fork is the valid one. This solves the problem of malicious nodes creating a fork before spending some funds, to regain them. During the consensus algorithm, the new shorter fork is rejected.
- *Script*
- *Smart Contract*

Because blockchain technology was created to support the Bitcoin cryptocurrency, the widest use of blockchain is to create alternative cryptocurrencies. The key issue blockchain solves is known as the *double spending* problem. Without a central entity, a network of untrusted peers must agree on valid transactions, controlling that no malicious peer spends twice the same funds. Bitcoin solves it by making all transactions public on the ledger, so any node can keep track of the spent transactions.

blockchain characteristics, ideas de aqui reescribir copiado:

full decentralization, traceability and transparency of transactions, proof of transaction viability, prohibitively high cost to attempt to alter transaction history, i. e. 51 attacks, an automated form of resolution, e. g., avoiding double spending, incentives required to participate, and trust enabling among non-trusted peers.

stakeholders do not have to share a common trust basis, blockchains decentralized data storage, typically in a peer-to-peer-based network structure and replicated to all interested peers, making data loss impossible, besides act-of-god situations.

From the point of view of **blockchain types**, there are three different architectures:

- *Public Permissionless Blockchain*, where anyone can read, write, participate in consensus and where transactions are transparent but with participants with some anonymity or pseudo-anonymity.

- *Private Permissioned Blockchain*, where the participating nodes must be granted access to the network, via an invitation or *permission*, in order to perform operations over the distributed ledger or participate in consensus. The access control mechanism could vary: existing participants could decide future entrants; a regulatory authority could issue licenses for participation; or a consortium could make the decisions instead [23].
- *Public Permissioned Blockchain*, a concept introduced by the Sovrin Foundation [3], identifies those blockchain instances that are open for all to use, that is, read or change the state of the ledger, but the network of nodes performing consensus is permissioned.

2.2. Self-sovereign identities in blockchain

Centralized Identity management solutions, based on traditional Central Authorities, are subject to different problems and threats such as data breaches, identity theft and privacy concerns. The rise of federated Identity management models helped to mitigate partially those problems. This kind of server-centric systems enable users to adopt the same identity system across different domains. The user is redirected for authentication and user identity data retrieval to his home identity provider. Some federated IdM systems such as Stork [] go an step forward implementing a user-centric approach, since users are put in the middle to take control of their personal data, asking their consent each time their user data is released in the federation from its home identity provider (data controller) to the Service provider (data processor).

Several technologies such as, OpenId [], OAuth [], SAML [] or Fido [], allows implementing this user-centric approach, enabling users to share its identity across different services. Nonetheless, user-centric identity federations are still subject to privacy issues, as user data related to his identity is still hold in server side, and authentication is validated in the server (usually through a Knowledge-base and some other weak authentication e.g. passwords).

Unlike those traditional approaches, the self-sovereign identities [3] focuses on providing a privacy-respectful solution, enabling users with fully control and management of their personal identity data without needing a third-party centralized authority. Thus, citizens are not anymore data subjects, instead, they become the data controller of their own identity. This is, they can determine the purposes, and ways in which personal data is processed.

Blockchain enables sovereignty as users can be endowed with means to transfer digital assets to anyone privately, without rules in behind, which ultimately increases the global democracy in the world.

Figure ?? shows a representation of the evolution of these Identity Management models.

FIGURA evolucion

Chritofer Allen described in the detail this path [24] to self-sovereign identities to achieve a full user autonomic, and detailed the ten Principles of Self-Sovereign Identity: Existence, Control, Access, Transparency, Persistence, Portability, Interoperability, Consent, Minimalization, Protection.

Identity sovereign have been already realized in the past following a user-centric and privacy-preserving approach for mobiles using Anonymous credential systems, e.g. in the scope of Irma project [25], and even, for IoT scenarios, e.g. in our previous works [26] [27]. Unlike those proposals, nowadays, self-sovereign identities are being lead to a new stage, as they are being materialized through blockchain, which increases facilitates the governance of the SSI system, increases the performance to internet scale and enables the accessibility of identities to everyone. In this sense, latest solutions [2] [3] make use of distributed ledger technologies (DLT), along with user-centric and mobile-centric approaches, thereby allowing users to maintain securely their needed crypto-credentials linked to the blockchain, which acts as distributed and reliable identity verifier, providing provenance and verifiability.

[28]

3. Privacy challenges in Blockchain scenarios

Main Privacy open challenges in the application of blockchain in different domains, including IoT.

- Multi-entry transactions: this kind of transaction require having different addresses belonging to the same user in the same transaction, making them linkable [7]. There should be a different address for every received transaction of digital asset.
- Transactions with change: it allows to trace the user when he employs the same address to get the change []. There should be different addresses to get the change of the transaction.
- Curious/Malicious Mixing services: outsourced and centralized services can be adopted by users to improve their privacy by mixing transactions. However, it can become a privacy issue as the service might know both the input and output addresses pairs [].
- Network privacy: association of IP addresses to users to transactions, by listening the blockchain nodes that participate in the P2P network traffic of the blockchain [29].
- Private-keys management Outsourcing: delegation of the management in external centralized Wallets, might raise security an privacy issues [30].
- Malicious Smart contracts: might access the data being processed in the transactions. It means that privacy might be compromised in case the code leaks information about the data being handled by the smart-contract.
- Private-keys recovery: in case the user loses the keys of his wallet (e.g. in smartphone). Some blockchain solutions such as Uport [2] and Sovrin [?] already deals with that challenge.
- Non erasable Data: privacy is also about the right to erasure data, but the blockchain is immutable. Therefore, personal data should not be stored in blockchain. In this regard, [31] present a Block matrix data Structure for Integrity Protection with Erasure Capability. It allows continuous addition of hash-linked records and enabling,at the same time, deletion of records.
- Brute force attack to Hashes: hashes are supposed to be one-way, however, they have recently forged using brute-force attacks, when time to guess is not a restriction to find out the input of the digest. It is aggravated with the upcoming Quantum Computing. Therefore, no hashes should be stored on-chain, at least without a previous randomization process.
- Privacy Usability: non-technical people will struggle to deal with the key management required in new IdM system in blockchains, specially during key recovery process.

[32] Privacy in bitcoin transactions: new challenges from blockchain scalability solutions

Blockchain is currently a disruptive technology which is revolutionizing the way organizations handle their data by enabling mechanisms to improve the security and non-repudiation in their operations. Essentially, Blockchain is a distributed and immutable database that allows anyone to audit the data and where nobody can modify the content once is stored. Maintaining the privacy and anonymity of the users who participate in the blockchain is one of the aspects to be considered in the deployment of this type of technology.

Talking about users and privacy, we should mention the existence of regulations such as General Data Protection Regulation (GDPR)[33], which empowers citizens with the rights to have their data rectified, erased or forgotten. In this sense, the existing regulations may come into conflict with Blockchain technology in the way of when you keep something in the chain it is supposed to be immutable, persistent and unmodifiable. For example, if a user registers his personal data in a blockchain solution, he must be aware that maybe he will not be able to exercise all his rights collected by the GDPR.

In addition to the possible problems with this type of regulations, there are another series of inconveniences that affect the privacy of users when they are operating on a platform based on blockchain technology. Above others, the loss of anonymity is highlighted through the relationship between transactions and end users, so that the activity of users can be monitored with the consequent associated privacy problems.

Privacy issues appear in both types of blockchains although they can affect differently or be more or less serious depending on the context.

In public blockchain scenarios like Bitcoin or Ethereum. The **Linkability** problem compromises the user privacy by the possibility of correlate his real identity with his transactions. In Bitcoin users tend to assume that their operations are anonymous without being aware that their transactions could be deanonymized if they do not have care and follow some good practices to make their transactions.

In these types of scenarios, two types of keys are involved to guarantee the correct operation, the public key and a private key. Meanwhile the private key is generated, in most cases, by Elliptic Curve Digital Signature Algorithm (ECDSA), the public key is obtained or derived from the private one.

Time before, the public key was used for send or receive Bitcoin but now, it is used to derive an address for the same purpose.

Regarding to the private key, nobody except the user must know it. If someone else knows your private key, he will be able to perform transactions on your behalf (identity theft) and, in the case of Bitcoin, steal your coins. This happens because the private key is used to sign each transaction that we made in the blockchain. The signature is used to confirm that the transaction has come from the legitimate user, and also prevents the transaction from being altered once is issued.

Looking at privacy in a Bitcoin transaction, by observing the addresses derived from the public key or the public key itself we could link addresses with real users despite the supposed anonymity. This identification may be achieved through different techniques (e.g tracing software) or by third parties able to provide identification data to public authorities based on the user's public key.

For example, let's say that we have donated to any platform through Bitcoin. By knowing the donation address we could track everyone who donated to them, furthermore, by knowing a certain address we could trace the coins until the creation moment or when they have been received and from where. In addition, we could try to link addresses to particular nicknames by scanning the web looking for the address in forums or other kind of exchange sites.

Even if we are careful, there is another risk that can expose our privacy and even the others involved in the transactions. In scenarios such as Bitcoin, **reusing addresses** is also privacy risk. The public history of blockchain transactions can unveil economic activity between parties which ultimately harms their privacy [].

Many problems are caused because a malicious user relate transactions with wallets and he is able to discover balances, destinations or other sensitive information. Avoiding use the same address many times have some advantages:

- Protects you against a bad implementation of ECDSA, typically found in wallets.
- Potentially protect you against Quantum computing. When we generate new addresses, new pairs of public and private keys must be generated. In the case that the ECDSA algorithm is broken, obtain the private keys from the public ones will be something trivial so if we always use the same address we would be completely exposed.
- Prevents to be exposed from undiscovered failures in ECDSA theory.

So, in order to make an unlinked transaction between two users in public blockchains like Bitcoin, the payee must deliver his address to the payer through a private channel. Then, to avoid that different transactions can be linked to the same owner, the payee should generate different addresses and avoid linking them to his own pseudonym based on his public key.

In the context of public blockchain, there are also tools that automate the observation of transactions made. **Blockchain crawlers** exists like in the traditional web. This type of software analyze transactions and is able to look its content and extract knowledge that could allow identify users by the use of linking algorithms.

Privacy problems also exist in private blockchain however, linkability loses importance and now the main problem it is in the identity management. Which attributes about us are share with, how and under what circumstances.

Blockchain for data storage

It is possible to use Blockchain for data storage, as we have said, it is an open database in which anyone can look. The direct solution for this purpose is to store the data directly over the chain.

We should consider that the stored data might be sensitive, for example, health data or family pictures. Taking this into account we could apply encryption and store the data encrypted in the chain then, only authorized users will be able to decrypt the content, while any user would be able to verify the stored data.

This approach have several drawbacks. Due to the immutability of the content, the storage will grow exponentially making transactions confirmation very slow. More worrying is where the credentials needed to decrypt the content were compromised. The encrypted content will remain in the blockchain forever visible for anyone.

3.1. Privacy requirements

GDPR [33] aims to avoid the collection (and processing) of personal data which are not reasonably essential to achieve the intended research purposes. Providing privacy-by-design and by-default.

GDPR article 5, defines 6 clauses corresponding to 6 principles regarding processing personal data.

1. **Lawfulness, fairness and transparency:** processed lawfully, fairly and in a transparent manner in relation to the data subject. Transparency: informing the subject about the kind of data processing to be done. Fair: the data processing must correspond to what has been described. Lawful: Processing must meet the tests described in GDPR.
2. **Purpose limitations:** Personal data can only be obtained for “specified, explicit and legitimate purposes”. Data cannot further processed in a manner that is incompatible with those purposes without further consent.
3. **Data minimisation:** Data collected on a subject should be “adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed”.
4. **Accuracy:** Data must be "accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay"
5. **Storage limitations:** personal data is “kept in a form which permits identification of data subjects for no longer than necessary for the purposes for which the personal data are processed”, that is, data no longer required should be removed.
6. **Integrity and confidentiality:** requires that data is processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures.

4. Review of Privacy-preserving solutions for blockchain

4.1. Main Privacy-preserving approaches in blockchain

This section provides a first insight in the main technologies that enable privacy-preserving in blockchain.

ZERO-KNOWLEDGE PROOFS

A Zero-Knowledge Proof (ZKP) is a cryptographic protocol that allows a party, the prover, to proof to another entity, the verifier, that a given statement is true, without revealing any information except that the proof itself is correct. For example, the statement of knowing a secret value is proved with a *Zero-Knowledge Proof of Knowledge*, where the secret value is kept private to the prover even

after the ZKP is performed. For an in-depth understanding of the mathematical principles of ZKPs, we recommend reading [34].

The three requisites for a protocol to be a Zero-Knowledge proof are:

- *Completeness*: If the statement to proof is true, the prover can always carry out a successful proof.
- *Soundness*: If the statement to proof is false, a cheating prover can not convince the verifier that it is true, except for a small probability.
- *Zero-Knowledge*: There exists a polynomial-time bound algorithm that can generate on its own messages that could be part of a successful proof between a prover and a verifier. We call this algorithm the *simulator*. It is Zero-Knowledge because if a third party who doesn't know if the statement is true or false can generate a valid transcript of the protocol, then, neither the verifier or an eavesdropper could obtain any extra information from a real transcript.

Because a ZKP is an interactive protocol, the applicability is limited to scenarios where a prover and verifier are synchronized. To solve this problem, we can apply the Fiat-Shamir heuristic [35], where the prover generates a proof replacing the verifier by a random oracle (in practice, a hash function).

Zero-Knowledge Proofs are applied in different ways over the blockchain, either to validate transactions or to provide privacy to the data in the blockchain:

– ZKP based cryptocurrencies

Zerocoin [36] introduced *zero-knowledge proofs of set membership* as a privacy solution to Bitcoin's pseudonymity. They use ZKPs to solve the double spending problem without revealing the transaction the funds come from. First, the prover, owner of the money, previously committed the funds with a secure digital commitment scheme. The funds correspond to a serial number S , and the commitment opens with a random r , both kept secret. When the prover wants to transfer the funds, it accumulates multiple commitments from the blockchain history, and proofs that she knows the secret r such that one of the commitments opens to the value S . This proof doesn't reveal neither the secret r , nor which commitment of the chosen set was opened. This is called a *zero-knowledge proof of set membership*. In the transaction, the ZKP hides the origin of the funds, and the revealed serial number S prevents the double spending.

– ZKP based Anonymous Credential Systems on blockchain

Another application of Zero-Knowledge Proofs in the blockchain are the Anonymous Credential Systems (ACS). Roughly speaking, the blockchain acts as a decentralized PKI, where the credentials public information is stored. These credentials allow off-ledger interactions between a prover and a verifier, such that the verifier can check on the ledger the validity of the Issuer's signature.

Between the most notable ledgers is Hyperledger Indy, based on the Sovrin project [1]. The ACS are based on Camenisch-Lysyanskaya's signature over Attribute-Based Credentials. Zero-Knowledge Proofs over these credentials allow the prover to create unlinkable pseudonyms, to selectively disclosure her attributes, to proof that a value lives in a specified range, etc.

The credentials are never stored on the blockchain, to keep the prover's attributes secure. The Issuer's public keys are stored on the ledger, so the credentials can be verified and trusted. Because in this type of blockchain the ZKPs do not affect the validity of a transaction, the trust over an issuer depends on the reputation a verifier gives him. The reputation could be calculated with other data on the blockchain or with real world, off-chain information, like a web browser trusts the CA certificates pre-installed within it.

zkSNARK

The acronym zkSNARK means *zero-knowledge Succinct Non-Interactive ARgument of Knowledge*. They are similar to Zero-Knowledge Proofs, but they offer more possibilities that the blockchain technology is taking advantage of as the research around them evolves.

Like with ZKPs, there are a prover and a verifier, and the prover wants to convince the verifier about a statement. The acronym reveals the properties of a zkSNARK:

- They are *zero-knowledge*, meaning that the verifier learns nothing from the argument, except that it is valid.
- They are *succinct*, in a way that the verification can be performed in a short lapse of time, and the proof can be stored in a relatively small amount of bytes.
- They are *Non-Interactive*, so the prover and verifier don't need to communicate synchronously, it's only needed a message from the prover (called the proof), which the verifier can validate off-line. This is also called the "public verifier" property, very useful in blockchain.
- They are *ARGuments*, unlike the *proofs* from ZKPs. In a ZKP, a prover can be considered computationally unbounded. In an argument, the prover is limited in polynomial time. This means that an argument has only *computational soundness*, instead of *perfect soundness*. The *completeness* property is the same as in Zero-Knowledge Proofs.
- They are *of Knowledge*, forcing the prover to know a *witness* in order to being able to construct the argument. This witness may be the same as in the NP class of problems, e.g. the prover knows the discrete logarithm to a public value.

The importance of zkSNARKs is that they can be used to proof the correct computation of any function, in other words, there exists a zkSNARK that produces a valid proof of the function being properly executed. In blockchain, the node who wants to change the state of the ledger, i.e. add a transaction, instead of sharing with the network the function to run and the input parameters, he can keep them secret, uploading instead a zkSNARK to proof that he performed the correct calculation, and the rest of the peers will trust him. The succinctness makes the proof suitable to be stored in a transaction, and the verification speed allows any other node to verify efficiently the proof.

They were applied firstly in blockchain in Zerocash [37] and Zcash [?], blockchain based token systems, also known as cryptocurrencies.

The problem any cryptocurrency must address is the double spending of tokens. To achieve privacy-preserving and solve the double spending problem, the mentioned systems apply zkSNARKs to the creation of transactions. A user who transfers tokens from existing transactions to another user must create a proof which usually states something like "I know the secret keys of the input accounts, allowing me to spend them, and they don't match any of the already spent ones". In Zcash this is achieved with the commitments and nullifiers, such that the network tracks the nullified (spent) transactions avoiding the double spending problem.

Comparing Bitcoin transactions with these *shielded* transactions, showcases the privacy-preserving achieved by zkSNARKs. A Bitcoin transaction is validated with the public addresses of the source and destination transactions, and the amounts transferred.

The great disadvantage of zkSNARKs is the requirement of a *common reference string* (CSR) of cryptographic values known by the verifier and prover, but generated by the verifier from secret values. In blockchain, this translates to a first secret trusted setup phase. At the beginning of the blockchain life, a trusted set of peers generate the random secret values, from them the CSR and then delete the secret values. It is important that this setup process is trusted in order to trust any proof based on the CSR.

- [38] SNARKs for C: Verifying program executions succinctly and in zero knowledge
- zkSNARK has been studied intensely in the past [39] and recently [40]
[41] zk-stark

The method to build a zkSNARK was to reduce the original problem to the QSP problem, which is in **NP**C and therefore any other problem in **NP** can be reduced to it. Then, playing with the QSP polynomials and homomorphic encryption we achieve evaluation of the polynomials in a secret point of the field. The simplification from validating polynomials equality to field elements equality brings succinctness to the scheme. Finally, adding some randomization we make the ensembles

indistinguishable to add Zero-Knowledge. The main drawbacks are, on one side, the ARGument property. A sufficiently powerful prover can deceive the validators with a false proof. On the other hand, the need of a trusted setup phase, which needs to be done only once, but shifts all the trust to those nodes involved in the bootstrapping. Current blockchains made public their randomization distributed secret bootstrapping protocol to ensure their implementation is secure, even so, there exist the probability of collusion.

As a solution to the second problem, STARKs add the Transparency property, removing the need of any secret trusted setup, as any public values are random coin flips. Nonetheless, it is still a novel technology and is not broadly applied.

HOMOMORPHIC ENCRYPTIONS Another method to share and perform operations over data without revealing private values is homomorphic encryption, where the ciphering function has some homomorphic properties that allow to operate over the ciphertext and obtain the same ciphered solution as if we operated over the cleartext and then ciphered with the same encryption function.

Given an RSA public key (e, n) and private key (d) , i.e. integers that validate $n = p \cdot q$ with p and q prime and $d \cdot e \equiv 1 \pmod{\phi(n)}$. The encryption of a message x is given by $E(x) \equiv x^e \pmod{n}$.

The group multiplication is then a homomorphic property of the RSA encryption: $E(x)E(y) = x^e y^e \equiv (xy)^e \pmod{n} = E(xy)$.

Homomorphic hiding is mostly used for privacy preserving multi-party computations, and, as have been shown, to create zkSNARKs. The homomorphism property between ciphered values usually affects only to one operation, and even obtaining homomorphism of two operands, they may be limited in the number of applications. The homomorphic hiding studied uses the discrete logarithm problem and elliptic curves' pairing functions. Fan-Vercauteren's homomorphic encryption is a more complex construction, which uses polynomials modulo other polynomials and approximation functions from reals to integers, but in exchange obtain post-quantum resistance.

RING SIGNATURES

Given a group of members with private and public keys, a ring signature is a type of digital signature performed by one of the members of the group, but the signature itself does not reveal who signed it. The name comes from the shape representing the signature, not the algebraic structure.

Given the set of public and private key pairs of the members, $(P_1, S_1), \dots, (P_n, S_n)$, one can compute the signature over the message m with only the input $(m, S_i, P_1, \dots, P_n)$, where the only private key needed is one from any of the group. But to verify the signature it is only needed the public keys from all members.

The mathematical idea behind the ring signature is that there exists a function that can be computed with only the public keys (verification), but knowing a private key allows to choose a value that makes the function output a desired specific value (signature). The signature process consists on applying the function recursively to the previous calculated value plus a seed, starting with a random *glue* value v ; with the help of the private key, one of those random seeds is overwritten with a specific value, with the objective that the final computed value of the function is equal to v , closing the "ring".

Anyone can verify that a ring signature is valid given the public keys and seeds used in the computation, checking that the first value v equals the last computed value. But it is infeasible to tell which "link" of the ring used the private key to choose the seed.

There exists an extension called *linkable ring signatures*, where given two signatures computed with the same private key, that fact can be detected, but not which key.

The applications of group or ring signatures are not as broad as the general proofs of ZKPs, zkSNARKs or STARKs, but ring signatures are a simple, yet effective way to hide or mix the signer of a transaction without the need of a trusted third party. The evolution from group signatures to ring signatures is somewhat equivalent to that of zkSNARKs to STARKs. Group signatures needed

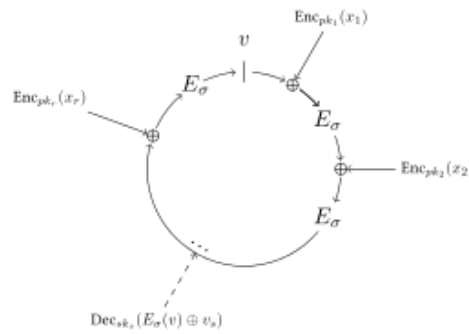


Figure 1. Rivest, Shamir, Tauman ring signature scheme.

an entity called the group manager, who could deanonymize any signature. For blockchains, ring signatures allow independence of that manager, and any user can choose any set of signatures, according to the privacy degree they desire.

In Blockchain ring signatures are applied to conceal the sender's identity, using several public keys at random, without the need of a special node that actively participates in the transaction to add privacy. Equivalent to the Zero-Knowledge statement of "I possess the private key of this public key", signing a transaction means "I possess at least one private key from this set of public keys". To protect the recipient node's identity, after each transaction a new public key is generated, and using a key-exchange algorithm like Diffie-Hellman's, the recipient is the only who can recover the private key. Thanks to the linkability property of some ring signatures, the double spending can be detected, while still not revealing who tried to do it.

MIXERS

DIFFERENTIAL PRIVACY

Differential privacy [42] addressed the privacy-preserving data analysis, going beyond data de-identification, it addresses whether a methodology for data analysis reveals or not information about an individual.

It is proven that given a de-identified dataset, one can still learn something about individuals performing statistical analysis.

Differential privacy consists on introducing a certain amount of random noise to queries against the data, in a way that any statistical analysis over the whole set is significantly close to the real results, but inference over any individual is infeasible.

In *data mining* applications this allows learning useful information about a population, without learning nothing about a particular individual.

However, data cannot be fully anonymized and be useful for analysis, it is a trade-off between utility and privacy. Certain differential privacy techniques achieves a certain degree of privacy, measured with the definition's constant ϵ , and the number of queries performed over time.

4.2. Data anonymization methods

4.3. Encryption methods

The data running on the blockchain can be protected through different encryption methods, thereby achieving confidentiality and therefore, a holistic vision of privacy.

Secure multi-party computation [] homomorphic encryption [] PEP (Polymorphic encryption and pseudonymisation) [] CP-ABE and KP-ABE []

Table 1. Comparison of cryptographic solutions

	Commitment Scheme	ZKP	Homomorphic Hiding	zkSNARK	STARK	Group Signatures	Ring Signatures
Trusted Third Party	No	No	No	Setup Phase	No	Group manager	No
Universality	No	Yes	No	Yes	Yes	No	No
Non-interactive	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Succinct	*	*	*	Yes	Yes	*	*
Transparent	*	*	*	No	Yes	*	*
Zero-Knowledge	*	Yes	*	Yes	Yes	*	*
Post-Quantum resistant	*	*	*	No	Yes	*	*
Transaction's privacy	Yes*	Yes	No	Yes	Yes	No	Yes
Proof of execution	No	Yes	No	Yes	Yes	No	No
Maturity	High	High	High	High	Low	High	High

4.4. Privacy-preserving technologies for blockchain taxonomy??

Table 1 showcases the some differences and common points of the cryptographic technologies studied.

Notice how many of the most common properties are related to how blockchain works. For example, Non-Interactivity is crucial in a distributed ledger, where each node will validate a proof independently of when the prover created it. Universality refers to how the technology can be applied to *any* problem, in this case, ZKPs, zkSNARKs and STARKs can be applied to proof problems in NP, but homomorphic hiding can only hide operations over the algebraic structure. When a * symbol appears, it means that the property may or may not be achieved depending on the specific construction, or that the property is not directly evaluable for the technology. For example, discrete logarithm homomorphic hiding is not post-quantum resistant, but Fan-Vercauteren's homomorphic encryption scheme is post-quantum resistant, although the first allows any number of homomorphic operations, meanwhile the second one is limited depending on some system parameters.

4.5. Privacy-preserving research proposals

4.5.1. Permissionless (public) Blockchains

In Bitcoin a transaction between a source payer address A_{Ps} and target payee A_{Pt} can be defined as $T(A_{Ps} \rightarrow A_{Pt}) = (\tau, B, A_{Pt}, \text{Sig}_{sk_{A_{Ps}}}(\tau, B, A_{Pt}))$. Where the payer A_{Ps} signs with its secret key $sk_{A_{Ps}}$ the amount B of bitcoins, including a reference to the most recent previous transaction τ in which A_{Ps} acquired the amount B of bitcoins. Afterwards, when the bitcoin transaction T is confirmed, A_{Pt} can use this transaction as reference to spend the acquired bitcoins. The bitcoins transactions set up a public record that can be verified by anyone, by checking the signatures of the chain. The bitcoins addresses are pseudonyms that are mapped to the ECDSA public/private key pair, and users have hundreds of them kept in their wallets. When the payment amount exceeds the value of each of the available bitcoins in one address it performs multi-input transaction, aggregating different input addresses from the same user and linking different addresses each other.

When it comes to privacy in blockchain-based cryptocurrencies two main properties can be identified. *Anonymity*, that is, hiding the identities of payer and payee in a transaction and, *Confidentiality*, i.e. hiding the amount of coins transferred. However, as it has been analysed in several works [12] [13] [14], Bitcoin lacks of confidentiality and it is subject to diverse privacy issues, users are not actually anonymous in their transactions, only pseudonymity is ensured. The usage of different pseudonyms achieves certain degree of unlikability, but it not sufficient to become anonymous, especially when different user's addresses need to be combined in one transaction to sum up the transferred amount, as different transaction can be link each other.

Moreover, several thefts have occurred in Bitcoin with fraudulent mix services, in 2015 Darknet Marketplace more than 10 million were stolen in bitcoins, thereby it is important to devise

anonymity-enhancing solutions to prevent, by design, coins theft. In this sense, recently a plethora of solutions have been proposed to deal with privacy concerns in BitCoin.

Transaction mixing services or tumblers such as Bitcoin Fog [43] can leverage privacy by splinting transactions into smaller ones, obfuscating and scheduling them. Mixing services [] [] provide a mixing address to receive coins from different users and send them randomly to a fresh address for each user. The main concern is that the mixing service might know both the input and output addresses pairs, so anonymity is not preserved against the mixer service. Besides, the delay to reclaim coins must be big as it requires to have enough amount of coins to be mixed. Moreover, the mixing service might theft the coins in case it does not deliver the transaction to the appropriate agreed output address. To deal with this, different proposal like Mixcoin [8] provides accountability prior mixing, giving to the payee a signed warranty which will enable him to prove whether the mixing has been forged.

Blindcoin [30] modifies Mixcoin protocol to ensure that the input/output address mapping for is hidden with respect to the mixing server. It uses a blind signature scheme as well as an append-only public log.

Similarly, Tumblebit [9] is a coin mixing protocol that avoid users to trust in the tumbler for privacy or security, to perform the transactions anonymously, even with respect to the tumbler service. In the majority of mixer services including Tumblebit and Mixcoin, transactions must transfer the same amount of funds, to avoid linkability of input and output accounts.

Maxwell [44] introduced the Confidential Transactions (CT) approach, defining a transaction format that ensures payment value privacy in the blockchain. The input and output amounts in a transaction are hidden in Pedersen commitments [45]. The transaction includes a ZKP that proofs that all the outputs are positive, and the sum of the committed inputs is greater than the sum of the committed outputs.

Stealth Addresses (SA) [46] allows payees to publish a single, fixed, address that payers can send funds efficiently and privately. It aims to avoid that payers and third-parties can learn what other payments have been made to the stealth address.

Coinjoin [47] support multi-input multi-output (MIMO) of transactions, to hind the relation between payers and payees. Transferring funds from potentially concealed inputs to output address. However, CoinJoin can mix only small amounts of coins, between the users who are currently online. Besides, requires effort to find out appropriate mixing partners and it might be subject to DoS attacks by third parties and users if they reuse to provide signature during the signing round of the protocol.

Dark Wallet [48], is a plugin for browsers that permits private anonymous transactions of Bitcoin transactions.

Dash [49] is a cryptocurrency that focuses on ensuring anonymity and privacy, it provides a mixing service interacted by user's wallet that mixes user inputs with the inputs of two other users.

Coinshuffle++ [50] is a mixing protocol employed to create CoinJoin transactions.

Xim [51] is a two-party mixing protocol that is compatible with Bitcoin. It includes a decentralized system for anonymously pooling mix partners based on advertisements placed in the blockchain. Reducing Sybil attacks, denial-of-service attacks, and timing-based inference attacks.

Valueshuffle [10] extends the mixing protocol CoinShuffle++, and combines Stealth Addresses and the Confidential Transactions to provide payer and payee anonymity as well as payment value privacy. In ensures that inputs and outputs of the CoinJoin transaction cannot be linked. Since it uses SA, it provides one-time addresses for receiving payments, thereby providing payee anonymity. Furthermore, as it is based on CoinJoin it inherits theft resistant capabilities, compatibly with pruning of spent outputs. Unlike Tumblebit, or traditional mixers, in Valueshuffle funds are directly sent in one single transaction to the receivers of the CoinJoin transaction.

Möbius [52] uses an Smart Contract for performing the mixing, instead of using the the mixing service or Tumbler, it accepts any kind of blockchain token and can use anonymity protocol such

Table 2. My caption

Cite	Given Name	Year	Full Anonymity	Theft prevention	Coordination	Target Scenario	Cryptography technique	Implemented Validated	BlockChain Platform
[44]	Confidential Transactions	2016	yes	N/A		payment value privacy	Confidential Transactions	?	Bitcoin
[47]	CoinJoin		no	yes	Decentralized	payer anonymity	coin mixing		Bitcoin
[46]	Stealth Addresses	2014	N/A	N/A		payee anonymity	Stealth Addresses		Bitcoin
[50]	CoinShuffle++	2017	no	yes	decentralized	sender anonymity payer anonymity		yes	bitcoin
[10]	ValueShuffle:	2017	yes	yes	decentralized	payment value privacy, Payer anonymity, payee anonymity	CT,SA, mixing	yes	bicoïn
[52]	Mobius	2018		yes	decentralized		SA, Ring signatures	yes	ethereum
[8]	MixCoin	2014	TTP	TTP	Centralized	payer anonymity	Coin Mixing		bitcoin
[9]	Tumblebit	2017			Centralized		Coin Mixing		
[36]	ZeroCoin	2013	yes	yes	decentralized	Payer anonymity, payee anonymity	ZKSoK	prototype	
[11]	ZeroCash	2014	yes	yes	decentralized	payment value privacy, Payer anonymity, payee anonymity	zkSNARKS		
[54]	BulletProofs	2017	yes	yes	decentralized	payment value privacy, Payer anonymity, payee anonymity	short ZKP		
[51]	XIM	2014	yes		centralized between two-parties				bitcoin
[30]	BlindCoin	2015		TTP	Centralized	payee anonymity	Coin mixing, blind Signatures		bitcoin

as ZCash o Monero. It reduces high coordination costs of decentralized solutions and standalone cryptocurrencies, enhancing availability.

ZeroCoin [36] is a cryptographic extension to Bitcoin that augments the protocol to allow fully anonymous currency transactions. It uses signature of knowledge on message ZKSoK [] to sign the bitcoin transaction hash instead of using ECDSA. ZeroCoin authenticates coins using zero knowledge proofs to demonstrate that coins are in a public list of valid coins maintained on the blockchain. One of the main issues of ZeroCoin is performance as it uses double-discrete-logarithm proofs of knowledge that takes around 450 ms to be verified (at the 128-bit security level). Besides, ZeroCoin does not hide the amount of transactions, and it does not support payments of exact values.

ZeroCash [11] is an decentralized anonymous payment scheme that provides a anonymity-by-design solution leveraging zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). ZeroCash outperforms ZeroCoin, reducing size of transactions and verification time, hides transactions amounts and allows transactions of any kind.

One of the drawbacks of ZeroCash and ZeroCoin is that it is not possible to see the outputs that have been spent already, therefore, blockchain pruning is not possible. ZeroCash uses zkSNARKS [?] meaning that a trusted setup must be ensured. Besides, zkSNARKS might be subject to non-falsifiable cryptographic hardness assumptions [53].

BulletProofs [54] is a protocol based on non-interactive Zero-Knowledge Proof that employs short proofs and without the requirement of a trusted setup. It uses the Fiat-Shamir heuristic for making it non-interactive. This is built on the techniques of Bootle et al., which relies only on the discrete logarithm assumption, and the proofs are made. Multiple range proofs are aggregated in BulletProofs, e.g. for transactions with multiple outputs, into a single short proof. It is used for CT in Bitcoin, as the transactions can have two or more outputs. It also allows to aggregate multiple range proofs from different users into one single aggregated range proof.

4.5.2. Permissioned Blockchains

Luin Li et al. [6] defined and evaluated a research novel proposal for privacy-preserving a blockchain incentive vehicular network via an efficient anonymous announcement aggregation protocol, called CreditCoin. It allows that different users can send announcements and generate signatures and in a potentially untrusted environment. CreditCoin features conditional privacy as it is able to trace anonymous announcements of malicious users' identities.

Hardjone et al. [55] proposes a privacy-preserving ChainAnchor system on permissioned blockchain that uses zero-knowledge sent by the user to proof their membership against a

665 permissioned verifier. They propose a mixed permissioned-permissionless blockchain in which an
 666 organization to overlay their own permissioned group atop the public permissionless Blockchain,
 667 while maintaining the anonymity of its group-members.

668 Dunphy et al. [56] provides an interesting analysis and comparison of the main DLT-based
 669 Identity management schemes, concretely, UPort, ShoCard and Sovrin.

670 The work [57] proposes a privacy-preserving solution for healthcare scenarios, that enables
 671 patients to control and share his data easily through Multiparty Computation (MPC).

672 [58] Blockchain for IoT security and privacy: The case study of a smart home

673 Mimblewimble [59]

674 Non-interactive zero-knowledge (NIZK) proofs have been proposed as a tool to enable complex
 675 privacy-preserving smart contracts [60] [?]. However, performance is a problem as communications
 676 and computational power required in public blockchains with a smart contracts can be high.

677 Blockchain e-voting protocol [61] that does not require a trusted entity to protect the voter's
 678 privacy.

679 Zero-Knowledge Contingent Payments [?]

680 Baars et al. [62] studied the self-sovereign identity concept, and designed an architecture for
 681 a Decentralized Identity Management System (DIMS) using claim-based identity and blockchain
 682 technology.

683 **tabla permissionned BC-papers del excel**

684 Hawk [63] is a privacy-preserving decentralized smart contract system where contractual parties
 685 interact with the blockchain, using cryptographic primitives such as zero-knowledge proofs.

686 4.6. Privacy-preserving Identity Management systems and platforms in Blockchain (Jorge, all)

687 soluciones IdM fundamentalmente para permissioned blockchain

688 **Uport [2]**

689 uPort uses 20-byte hexadecimal identifier to represent the user's uPortID, with the address of a
 690 Proxy smart contract deployed in Ethereum. Such a contract introduces a layer of indirection between
 691 the user's private key - maintained on their mobile device - and the application smart contract being
 692 accessed by users. The user app contacts an instantiation of a Controller smart contract (which holds
 693 the main access control logic such as Authentication/authorization), which in turn, is the unique
 694 entity capable of interacting with the proxy. uPort support certain degree of unlinkability, as user can
 695 create many unlinkable uPortIDs. Selective disclosure of attribute is allowed encrypting an attribute
 696 with a symmetric encryption key, which in turn, is individually encrypted with the public key of the
 697 identity allowed to read the attestation attribute. uPort also support identity recovery and rely on
 698 DID¹ standard and Verifiable Claims², both being standardized by the W3CC.

699 **Sovrin [3]**, is open-source decentralized identity network for permissioned blockchain. Sovrin
 700 is an utility identity deployed over Hyperledger-Indy³ that implement Plenum [?] byzantine BPT
 701 consensus algorithm for consensus. Sovrin supports DPKI (Decentralized Public Key Infrastructure),
 702 where every public key has its own public address in the ledger (DID, decentralized identifier) that
 703 enable universal verification of claims. Sovrin allows having different DID for every relationship the
 704 user has, with different keypairs, unlinkable each other. Like in uPort, Sovrin user generates crypto
 705 keypairs and maintain the private key in the user's mobile. It supports identity recovery, and make
 706 use of software Agents that can act on behalf of the user to facilitate interactions with third party
 707 service provider agents. Unlike uPort, Sovrin supports, not only attestation and verifiable assertions,
 708 but also Anonymous credentials with Zero-knowledge proofs to achieve fully unlinkability and

¹ <https://w3c-ccg.github.io/did-spec>

² <https://www.w3.org/2017/vc/chapter.html>

³ <https://www.hyperledger.org/projects/hyperledger-indy>

Table 3. Privacy-preserving platforms in blockchain

GDPR principles	Lawfulness, fairness and transparency	Purpose limitations	Data minimization	Accuracy	Storage limitations:	Integrity and confidentiality
uPort [2]	(-) usability privacy-implications are not totally shown to users.	(+) Prove ownership of uPortId and control of revealed attributes to whom	(+) unlinkability (uPortIDs) (+) verifiable claims, (+) Selective disclosure, (+) ID-KeysRecovery, (+) permits usage without IdProofing		(-) Even if encrypted, attributes stored in central registry could be analyzed, as they are public accessible	(+) attributes and attestations encrypted in FIPS repository (-) security in permissionless DLTs is more difficult to handle
Sovrin [3]	(+) Proof of Sharing Consent between DIDs, receipt without PPI (+) Consent for Proof Request (+) freedom to change Wallet providers and Agents (-) usability privacy-implications are not totally shown to users.	(+) Prove ownership of DID and control of revealed attributes to whom (+) Proof request state the purpose of data being requested	(+) unlinkability (+) verifiable claims (+) zero-knowledge proofs (+) ID recovery	(+) revocation is decentralized, asynchronous, and private, using cryptographic techniques (accumulators)	(+) Keeps private data, including hashes, off the Sovrin ledger.	(+) Permissioned DLT. stewards enforce security and run BFP consensus protocol (+) Confidential Agent-to-Agent communications off-ledger
ShoCard [64]	(+) User consent prior releasing	(+) permission-based user data release, (+) proof user consent	(+) user claims (-) unlinkability against the ShoCard server is not fully ensured	(+) user consent can be revoked	(+) blockchain certification records only maintain "signatures of hashes of data + code"	(-) ShoCard server might store encrypted sensitive data (biometrics) (+) blockchain agnostic, sidechain speed up process
Civic [65]	(+) user consent using QR codes (-) private-solution. Lack of implementation and technical details of the system"	(+) multiple identity validation service providers (+) verification of submitted PII data	(+) attestations (+) release only necessary information to SP (-) unclear anonymity procedures (-) unclear ID-Keysrecovery"	(+) Identity data is revocable in blockchain by the authenticating authority.	(-) Civic SIP server can act as intermediary Authz Server between user and SP, which might trace user's.	(+) PII is maintained encrypted in the app, accessed with biometrics (+) multifactor authentication"

comply with the minimal disclosure principle. Sovrin allows demonstrate proofs offchain directly in a secure channel with the third party, without storing the attributes in the ledger. In this case, the blockchain is used to identify the trusted service endpoint to interact with. The web of trust supported by the sovrin trust anchors provides verifiability of the target party being interacted.

Shocard [64] uses the ledger as repository of certifications that maintain signatures-of-hashes-of each personal data along with with a code to avoid brute-force discovery. Shocard is blockchain agnostic, and uses private parallel sidechains to speed up the transactions in the ledger. It supports that third party can verify and then certify an individual's identity and credentials. Shocard provides App that hold cryptokeys and entities can verify a user's claims of identity through certifications and signatures hold in the ledger. The ShoCardId can be bootstrapped from trusted breeder document e.g. ePassport, through IDproofing stage to validate user's identity checking biometrics in the ePassport chip. However, the enrollment with biometrics requires storing encrypted sensitive data in the Shocard server. This server might trace interactions between Relying parties and ShoCardIDs.

Civic [65] is a decentralised trusted Identity applications that provides identity proofing relying on existing eID documents like ePassports. The app stores private keys of user Civic ID used for record signed hashes of attestations in the blockchain. Support multiple-factor authentication, user-consent, and minimal disclosure. However, anonymity and unlikability mechanisms are not seem to be implemented in Civic yet. And the civic server can act as a intermediary Authz Server between user and SP, which raises privacy concerns.

CryptoNote [28], the destination of each CryptoNote output is a public key, derived from recipient's address and sender's random data.

In cryptonote unlike Bitcoin each destination key is different. It uses ring signatures, and concretely a custom one-time ring signature scheme. In CryptoNote, the sender performs a Diffie-Hellman in order to obtain a shared secret, that is derived from his data along with first part of the recipient's address. Afterwards, a one-time destination key is computed, derived from such a shared secret the other part of the address. For these two esteps, tt requires two EC-keys for the recipient, meaning that address in CryptoNote are larger that a Bitcoin wallet address. Similarly, receiver also carries out the Diffie-Hellman to get the secret key.

CryptoNote might have concerns dealing with the ring signatures depending on the large of anonymity set n , as it requires that each transaction contains a ring signature of size $O(n)$. Besides, storing ring signatures in public blockchain might become a problem. Unlike CryptoNote, CoinJoin [47] or ValueShuffle [10] facilitate pruning, which is a drawback in Cryptonote, as rings signatures make the pruning difficult.

Enigma [66], developed by MIT, allows secure data sharing, using multi-party computation (MPC) and homomorphic encryption. Enigma aims to allow developers to build *privacy by design* and decentralized applications, avoiding a trusted third party (TTP). Enigma cannot be defined as a cryptocurrency or a blockchain platform. Instead, Enigma connects to an existing blockchain, and performs computation offloading of the private and intensive computations on an off-chain network. Unlike in blockchain schemes, the incentives are based on fees instead of mining rewards, where nodes are compensated for providing computational resources. The nodes pay a security deposit, which deters malicious nodes.

Enigma uses secure multi-party computation (sMPC) technology, in which data queries are distributively computed, and data is divided across different nodes each one computing certain functions in a distributed way without leaking information to the other.

Different solutions such as the Civic platform [65], are starting to apply the user-centric and decentralized blockchain approach to provide real-time authentication through biometrics, where identity data is encrypted in the mobile app. These solutions, uses Merkle tree randomized hashes using nonces signed by the validators entities, and supporting selective disclosure of the identity information (certain portions of the merckle tree hashes are revealed) in the blockchain, after user

consent, enabling user control privacy and enhancing security, since the attestations and proofs cannot be tampered in the blockchain.

IoTA (The tangle) [67]

H2020 project **DECODE** [68]

H2020 project **PRIViLEDGE** "Privacy-Enhancing Cryptography in Distributed Ledgers" (with IBM on board) (aun no hay papers ni web)

H2020 EU project "**My Health - My Data (MHMD)**" [5] defines a privacy-by-design blockchain solution by defining smart contracts that enforces dynamic consent mechanisms and peer-to-peer data transactions between public and private healthcare providers and patients.

5. Conclusions

Acknowledgments: This work has been partially funded by the stay (20177/EE/17) of "Fundacion Seneca-Agencia de Ciencia y Tecnología de la Region de Murcia", under the program "Jimenez de la Espada de Movilidad Investigadora, Cooperacion e Internacionalizacion". The research has been also supported by a postdoctoral INCIBE grant within the "Ayudas para la Excelencia de los Equipos de Investigación Avanzada en Ciberseguridad" Program, with code INCIBEI-2015-27363, as well as by the H2020 EU project ARIES under Grant 700085

Conflicts of Interest: Declare conflicts of interest or state "The authors declare no conflict of interest."

Bibliography

1. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. Blockchain technology: Beyond bitcoin. *Applied Innovation* **2016**, *2*, 6–10.
2. Lundkvist, C.; Heck, R.; Torstensson, J.; Mitton, Z.; Sena, M. Uport: A platform for self-sovereign identity **2017**. https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.
3. Tobin, A.; Reed, D. The Inevitable Rise of Self-Sovereign Identity. *The Sovrin Foundation* **2016**.
4. Camenisch, J.; Van Herreweghen, E. Design and Implementation of the Idemix Anonymous Credential System. Proceedings of the 9th ACM Conference on Computer and Communications Security; ACM: New York, NY, USA, 2002; CCS '02, pp. 21–30.
5. Panetta, R.; Cristofaro, L. A closer look at the EU-funded My Health My Data project. *Digital Health Legal* **2017**, pp. 10–11. <https://doi.org/10.5281/zenodo.1048999>.
6. Li, L.; Liu, J.; Cheng, L.; Qiu, S.; Wang, W.; Zhang, X.; Zhang, Z. CreditCoin: A Privacy-Preserving Blockchain-Based Incentive Announcement Network for Communications of Smart Vehicles. *IEEE Transactions on Intelligent Transportation Systems* **2018**, *PP*, 1–17.
7. Herrera-Joancomartí, J. Research and Challenges on Bitcoin Anonymity. Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance; Garcia-Alfaro, J.; Herrera-Joancomartí, J.; Lupu, E.; Posegga, J.; Aldini, A.; Martinelli, F.; Suri, N., Eds.; Springer International Publishing: Cham, 2015; pp. 3–16.
8. Bonneau, J.; Narayanan, A.; Miller, A.; Clark, J.; Kroll, J.A.; Felten, E.W. Mixcoin: Anonymity for Bitcoin with accountable mixes. International Conference on Financial Cryptography and Data Security. Springer, 2014, pp. 486–504.
9. Heilman, E.; Alshenibr, L.; Baldimtsi, F.; Scafuro, A.; Goldberg, S. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. NDSS Symposium 2017, 2017.
10. Ruffing, T.; Moreno-Sanchez, P. ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin. Financial Cryptography and Data Security; Springer International Publishing: Cham, 2017; pp. 133–154.
11. Sasson, E.B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E.; Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014, pp. 459–474.
12. Androulaki, E.; Karame, G.O.; Roeschlin, M.; Scherer, T.; Capkun, S. Evaluating user privacy in bitcoin. International Conference on Financial Cryptography and Data Security. Springer, 2013, pp. 34–51.
13. Koshy, P.; Koshy, D.; McDaniel, P. An analysis of anonymity in bitcoin using p2p network traffic. International Conference on Financial Cryptography and Data Security. Springer, 2014, pp. 469–485.

14. Reid, F.; Harrigan, M. An analysis of anonymity in the bitcoin system. Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. IEEE, 2011, pp. 1318–1326.
15. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system **2008**. <https://bitcoin.org/bitcoin.pdf>.
16. Bélanger, F.; Crossler, R.E. Privacy in the Digital Age: A Review of Information Privacy Research in Information Systems. *MIS Q.* **2011**, *35*, 1017–1042. <http://dl.acm.org/citation.cfm?id=2208940.2208951>.
17. Shen, Y.; Pearson, S. Privacy enhancing technologies: A review **2011**.
18. Danezis, G.; Gürses, S. A critical review of 10 years of privacy technology. *Proceedings of surveillance cultures: a global surveillance society* **2010**, pp. 1–16.
19. Smith, H.J.; Dinev, T.; Xu, H. Information privacy research: an interdisciplinary review. *MIS quarterly* **2011**, *35*, 989–1016.
20. Gatteschi, V.; Lamberti, F.; Demartini, C.; Pranteda, C.; Santamaría, V. To Blockchain or Not to Blockchain: That Is the Question. *IT Professional* **2018**, *20*, 62–74.
21. Al-Bassam, M.; Bano, S.; Danezis, G.; deVilliers, M.; Sonnino, A. D3.1 Survey of technologies for ABC, entitlements and blockchains. Technical report, DECODE H2020 project, 2017.
22. Merkle, R.C. Protocols for public key cryptosystems. Security and Privacy, 1980 IEEE Symposium on. IEEE, 1980, pp. 122–122.
23. Jayachandran, P. The difference between public and private blockchain - Blockchain Unleashed: IBM Blockchain Blog, 2017. <https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/>.
24. Allen, C. The Path to Self-Sovereign Identity **2017**.
25. van den Broek, F.; Hampiholi, B.; Jacobs, B. Securely derived identity credentials on smart phones via self-enrolment. International Workshop on Security and Trust Management. Springer, 2016, pp. 106–121.
26. Bernabé, J.B.; Ramos, J.L.H.; Gómez-Skarmeta, A.F. Holistic Privacy-Preserving Identity Management System for the Internet of Things. *Mobile Information Systems* **2017**, *2017*, 6384186:1–6384186:20.
27. Sanchez, J.L.C.; Bernabe, J.B.; Skarmeta, A.F. Integration of Anonymous Credential Systems in IoT Constrained Environments. *IEEE Access* **2018**, *6*, 4767–4778.
28. Van Saberhagen, N. Cryptonote v 2. 0, 2013.
29. Koshy, P.; Koshy, D.; McDaniel, P. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. Financial Cryptography and Data Security; Christin, N.; Safavi-Naini, R., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2014; pp. 469–485.
30. Valenta, L.; Rowan, B. Blindcoin: Blinded, Accountable Mixes for Bitcoin. Financial Cryptography and Data Security; Brenner, M.; Christin, N.; Johnson, B.; Rohloff, K., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2015; pp. 112–126.
31. Kuhn, D.R. A Data Structure for Integrity Protection with Erasure Capability. Technical report, NIST, 2018.
32. Herrera-Joancomartí, J.; Pérez-Solà, C. Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. Modeling Decisions for Artificial Intelligence. Springer, 2016, pp. 26–44.
33. European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union* **2016**, *L119*, 1–88. <http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC>.
34. Pieprzyk, J.; Hardjono, T.; Seberry, J. *Fundamentals of computer security*; Springer Science & Business Media, 2013; pp. 409–431.
35. Fiat, A.; Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. Conference on the Theory and Application of Cryptographic Techniques. Springer, 1986, pp. 186–194.
36. Miers, I.; Garman, C.; Green, M.; Rubin, A.D. Zerocoin: Anonymous distributed e-cash from bitcoin. Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013, pp. 397–411.
37. Sasson, E.B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E.; Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014, pp. 459–474.

38. Ben-Sasson, E.; Chiesa, A.; Genkin, D.; Tromer, E.; Virza, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology—CRYPTO 2013*; Springer, 2013; pp. 90–108.
39. Bitansky, N.; Canetti, R.; Chiesa, A.; Tromer, E. Recursive composition and bootstrapping for SNARKs and proof-carrying data. *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 2013, pp. 111–120.
40. Wahby, R.S.; Tzialla, I.; Thaler, J.; Walfish, M. Doubly-efficient zkSNARKs without trusted setup.
41. Ben-Sasson, E.; Bentov, I.; Horesh, Y.; Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity.
42. Dwork, C.; Roth, A.; others. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* **2014**, 9, 211–407.
43. BitcoinFog **2018**.
44. Maxwell, G. Confidential transactions. https://people.xiph.org/%7Egreg/confidential_values.txt.
45. Pedersen, T.P. Non-interactive and information-theoretic secure verifiable secret sharing. *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.
46. Todd, P. Stealth addresses. *Post on Bitcoin development mailing list*, <https://www.mail-archive.com/bitcoindevelopment@lists.sourceforge.net/msg03613.html>.
47. Maxwell, G. CoinJoin: Bitcoin privacy for the real world. *Post on Bitcoin forum*, 2013.
48. Greenberg, A. ‘Dark Wallet’ is about to make Bitcoin money laundering easier than ever **2014**.
49. DASH **2017**.
50. Ruffing, T.; Moreno-Sanchez, P.; Kate, A. P2P Mixing and Unlinkable Bitcoin Transactions. **2017**.
51. Bissias, G.; Ozisik, A.P.; Levine, B.N.; Liberatore, M. Sybil-resistant mixing for bitcoin. *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014, pp. 149–158.
52. Meiklejohn, S.; Mercer, R. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies* **2018**, 2018, 105–121.
53. Gentry, C.; Wichs, D. Separating succinct non-interactive arguments from all falsifiable assumptions. *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 2011, pp. 99–108.
54. Bünz, B.; Bootle, J.; Boneh, D.; Poelstra, A.; Wuille, P.; Maxwell, G. Bulletproofs: Efficient range proofs for confidential transactions. *Technical report*.
55. Hardjono, T.; Smith, N.P. Anonymous Identities for Permissioned Blockchains. **2016**.
56. Dunphy, P.; Petitcolas, F. A First Look at Identity Management Schemes on the Blockchain. *IEEE Security and Privacy Magazine* **2018**, to be published.
57. Yue, X.; Wang, H.; Jin, D.; Li, M.; Jiang, W. Healthcare Data Gateways: Found Healthcare Intelligence on Blockchain with Novel Privacy Risk Control. *Journal of Medical Systems* **2016**, 40, 218.
58. Dorri, A.; Kanhere, S.S.; Jurdak, R.; Gauravaram, P. Blockchain for IoT security and privacy: The case study of a smart home. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 618–623.
59. Jedusor, T.E. MIMBLEWIMBLE **2016**.
60. Campanelli, M.; Gennaro, R.; Goldfeder, S.; Nizzardo, L. Zero-knowledge contingent payments revisited: Attacks and payments for services. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 229–243.
61. McCorry, P.; Shahandashti, S.F.; Hao, F. A smart contract for boardroom voting with maximum voter privacy. *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 357–375.
62. Baars, D. Towards self-sovereign identity using blockchain technology. *Master’s thesis*, University of Twente, 2016.
63. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *Security and Privacy (SP)*, 2016 IEEE Symposium on. IEEE, 2016, pp. 839–858.
64. ShoCard, S. Travel Identity of the future **2016**. https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.
65. Technologies, C. Civic white Paper **2017**. <https://tokensale.civic.com/CivicTokenSaleWhitePaper.pdf>.
66. Zyskind, G.; Nathan, O.; Pentland, A. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *CoRR* **2015**, abs/1506.03471, [1506.03471].

- 913 67. Popov, S. The tangle **2016**.
- 914 68. Meessen, P.; Sonnino, A. D3.4 Initial decentralised models for data and identity management: Blockchain
915 and ABC MVPs. Technical report, DECODE H2020 project, 2017.
- 916 © 2018 by the authors.

Bibliography

- [1] Mustafa Al-Bassam et al. *D3.1 Survey of technologies for ABC, entitlements and blockchains*. Tech. rep. DECODE H2020 project, 2017.
- [2] Christopher Allen. “The Path to Self-Sovereign Identity”. In: (2017). URL: <https://github.com/ChristopherA/self-sovereign-identity/blob/master/ThePathToSelf-SovereignIdentity.md>.
- [3] Elli Androulaki et al. “Evaluating user privacy in bitcoin”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2013, pp. 34–51.
- [4] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. “Rbft: Redundant byzantine fault tolerance”. In: *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE. 2013, pp. 297–306.
- [5] DS Baars. “Towards self-sovereign identity using blockchain technology”. MA thesis. University of Twente, 2016.
- [6] Niko Barić and Birgit Pfitzmann. “Collision-free accumulators and fail-stop signature schemes without trees”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1997, pp. 480–494.
- [7] Eli Ben-Sasson. “Towards Transparent scalable computational integrity”. In: (2017).
- [8] Josh Benaloh and Michael De Mare. “One-way accumulators: A decentralized alternative to digital signatures”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1993, pp. 274–285.
- [9] Jorge Bernal Bernabé, José Luis Hernández Ramos, and Antonio Fernandez Gómez-Skarmeta. “Holistic Privacy-Preserving Identity Management System for the Internet of Things”. In: *Mobile Information Systems 2017* (2017), 6384186:1–6384186:20. DOI: [10.1155/2017/6384186](https://doi.org/10.1155/2017/6384186). URL: <https://doi.org/10.1155/2017/6384186>.
- [10] George Bissias et al. “Sybil-resistant mixing for bitcoin”. In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM. 2014, pp. 149–158.
- [11] Manuel Blum. “How to Prove a Theorem So No One Else Can Claim It”. In: (1986).
- [12] Joseph Bonneau et al. “Mixcoin: Anonymity for Bitcoin with accountable mixes”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 486–504.
- [13] Fabian van den Broek, Brinda Hampiholi, and Bart Jacobs. “Securely derived identity credentials on smart phones via self-enrolment”. In: *International Workshop on Security and Trust Management*. Springer. 2016, pp. 106–121.
- [14] Benedikt Bünz et al. *Bulletproofs: Efficient range proofs for confidential transactions*. Tech. rep.

- [15] Jan Camenisch and Anna Lysyanskaya. "Dynamic accumulators and application to efficient revocation of anonymous credentials". In: *Annual International Cryptology Conference*. Springer. 2002, pp. 61–76.
- [16] Jan Camenisch and Markus Stadler. "Efficient group signature schemes for large groups". In: *Advances in Cryptology—CRYPTO'97* (1997), pp. 410–424.
- [17] Matteo Campanelli et al. "Zero-knowledge contingent payments revisited: Attacks and payments for services". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 229–243.
- [18] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.
- [19] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.
- [20] Michael Crosby et al. "Blockchain technology: Beyond bitcoin". In: *Applied Innovation 2* (2016), pp. 6–10.
- [21] Ivan Damgard and Jesper Buus Nielsen. *Commitment Schemes and Zero-Knowledge Protocols* (2011).
- [22] Evan Duffield and Daniel Diaz. *Dash: A privacy-centric crypto-currency*. 2014.
- [23] Paul Dunphy and Fabien Petitcolas. "A First Look at Identity Management Schemes on the Blockchain". In: *IEEE Security and Privacy Magazine* to be published (2018).
- [24] The Royal Fork. "Graphical Address Generator". In: (2014). <http://royalforkblog.github.io/2014/08/11/graphical-address-generator/>.
- [25] Sovrin Foundation. "Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust". In: (2018).
- [26] Valentina Gatteschi et al. "To Blockchain or Not to Blockchain: That Is the Question". In: *IT Professional 20.2* (2018), pp. 62–74.
- [27] Rosario Gennaro et al. "Quadratic span programs and succinct NIZKs without PCPs". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 626–645.
- [28] Craig Gentry and Daniel Wichs. "Separating succinct non-interactive arguments from all falsifiable assumptions". In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM. 2011, pp. 99–108.
- [29] Thomas Hardjono and Ned P. Smith. "Anonymous Identities for Permissioned Blockchains". In: 2016.
- [30] Ethan Heilman et al. "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub". In: *NDSS Symposium 2017*. 2017.
- [31] Jordi Herrera-Joancomartí. "Research and Challenges on Bitcoin Anonymity". In: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*. Ed. by Joaquín García-Alfaro et al. Cham: Springer International Publishing, 2015, pp. 3–16.
- [32] Praveen Jayachandran. *The difference between public and private blockchain - Blockchain Unleashed: IBM Blockchain Blog*. <https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/>. 2017.

- [33] Louis C. Guillou Jean-Jacques Quisquater and Thomas A. Berson. "How to Explain Zero-Knowledge Protocols to Your Children". In: *Advances in Cryptology - CRYPTO '89* (1990). Proceedings 435: 628-631. <http://pages.cs.wisc.edu/~mkowalczyk/628.pdf>.
- [34] Jennifer Seberry Josef Pieprzyk Thomas Hardjono. *Fundamentals of Computer Security*. Springer, 2003.
- [35] KLMoney. "Bitcoin: Dissecting Transactions". In: (). <https://klmoney.wordpress.com/bitcoin-dissecting-transactions/>.
- [36] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 839–858.
- [37] Philip Koshy, Diana Koshy, and Patrick McDaniel. "An analysis of anonymity in bitcoin using p2p network traffic". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 469–485.
- [38] D Richard Kuhn. *A Data Structure for Integrity Protection with Erasure Capability*. Tech. rep. NIST, 2018.
- [39] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.
- [40] L. Li et al. "CreditCoin: A Privacy-Preserving Blockchain-Based Incentive Announcement Network for Communications of Smart Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* PP.99 (2018), pp. 1–17. ISSN: 1524-9050. DOI: [10.1109/TITS.2017.2777990](https://doi.org/10.1109/TITS.2017.2777990).
- [41] Joseph K. Liu and Duncan S. Wong. "Linkable Ring Signatures: Security Models and New Schemes". In: *Computational Science and Its Applications – ICCSA 2005*. Ed. by Osvaldo Gervasi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 614–623. ISBN: 978-3-540-32044-9.
- [42] Carsten Lund et al. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39 (1990), pp. 859–868.
- [43] Christian Lundkvist et al. "Uport: A platform for self-sovereign identity". In: (2017). https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.
- [44] Greg Maxwell. "CoinJoin: Bitcoin privacy for the real world". In: *Post on Bitcoin forum*. 2013.
- [45] Greg Maxwell. *Confidential transactions*. https://people.xiph.org/%7Egreg/confidential_values.txt.
- [46] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. "A smart contract for boardroom voting with maximum voter privacy". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 357–375.
- [47] Sarah Meiklejohn and Rebekah Mercer. "Möbius: Trustless tumbling for transaction privacy". In: *Proceedings on Privacy Enhancing Technologies* 2018.2 (2018), pp. 105–121.
- [48] Ralph C Merkle. "Protocols for public key cryptosystems". In: *Security and Privacy, 1980 IEEE Symposium on*. IEEE. 1980, pp. 122–122.
- [49] Ian Miers et al. "Zerocoin: Anonymous distributed e-cash from bitcoin". In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 397–411.

- [50] Malte Moser, Rainer Bohme, and Dominic Breuker. "An inquiry into money laundering tools in the Bitcoin ecosystem". In: *eCrime Researchers Summit (eCRS)*, 2013. IEEE. 2013, pp. 1–14.
- [51] Malte Moser. "Anonymity of bitcoin transactions". In: (2013).
- [52] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: (2008). <https://bitcoin.org/bitcoin.pdf>.
- [53] Rocco Panetta and Lorenzo Cristofaro. "A closer look at the EU-funded My Health My Data project". In: *Digital Health Legal* November 2017 (2017). <https://doi.org/10.5281/zenodo.1048999>, pp. 10–11. DOI: 10.5281/zenodo.1048999.
- [54] Torben Pryds Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: *Annual International Cryptology Conference*. Springer. 1991, pp. 129–140.
- [55] Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of computer security*. Springer Science & Business Media, 2013.
- [56] Fergal Reid and Martin Harrigan. "An analysis of anonymity in the bitcoin system". In: *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*. IEEE. 2011, pp. 1318–1326.
- [57] Christian Reitwiessner. *zkSNARKs in a nutshell*. 2016.
- [58] Tim Ruffing and Pedro Moreno-Sanchez. "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin". In: *Financial Cryptography and Data Security*. Cham: Springer International Publishing, 2017, pp. 133–154.
- [59] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "P2P Mixing and Unlinkable Bitcoin Transactions." In: (2017).
- [60] J. L. Canovas Sanchez, J. Bernal Bernabe, and A. F. Skarmeta. "Integration of Anonymous Credential Systems in IoT Constrained Environments". In: *IEEE Access* 6 (2018), pp. 4767–4778. DOI: 10.1109/ACCESS.2017.2788464.
- [61] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. "Survey of consensus protocols on blockchain applications". In: *Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on*. IEEE. 2017, pp. 1–5.
- [62] Eli Ben Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE. 2014, pp. 459–474.
- [63] SITA: ShoCard. "Travel Identity of the future". In: (2016). https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.
- [64] Civic Technologies. "Civic white Paper". In: (2017). <https://tokensale.civic.com/CivicTokenSaleWhitePaper.pdf>.
- [65] Andrew Tobin and Drummond Reed. "The Inevitable Rise of Self-Sovereign Identity". In: *The Sovrin Foundation* (2016).
- [66] P Todd. "Stealth addresses". In: *Post on Bitcoin development mailing list*, <https://www.mail-archive.com/bitcoindevelopment@lists.sourceforge.net/msg03613.html> ().

- [67] Kiran Vaidya. "Bitcoin's implementation of Blockchain". In: (2016). <https://medium.com/all-things-ledger/bitcoins-implementation-of-blockchain-2be713f662c2>.
- [68] Luke Valenta and Brendan Rowan. "Blindcoin: Blinded, Accountable Mixes for Bitcoin". In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 112–126. ISBN: 978-3-662-48051-9.
- [69] Luke Valenta and Brendan Rowan. "Blindcoin: Blinded, accountable mixes for bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 112–126.
- [70] Nicolas Van Saberhagen. *Cryptonote v 2. 0*. 2013.
- [71] Zcash - How zk-SNARKs work in Zcash. URL: <https://z.cash/technology/zksnarks.html> (visited on 06/20/2018).
- [72] Guy Zyskind, Oz Nathan, and Alex Pentland. "Enigma: Decentralized Computation Platform with Guaranteed Privacy". In: *CoRR* abs/1506.03471 (2015). eprint: [1506.03471](https://arxiv.org/abs/1506.03471).