Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Matematica

Addressing Privacy and Fungibility Issues in Bitcoin: Confidential Transactions

Relatori: Prof. Ferdinando Maria AMETRANO
Prof. Daniele MARAZZINA

Tesi di Laurea di:
Alessandro MIOLA
Matr. 862753

Anno Accademico 2017-2018

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

Insufficient privacy is recognized to be one of the major vulnerabilities of the Bitcoin's protocol, even because it undermines its fungibility. Bitcoin eliminates the need for a trusted third party, but mainly faces users' privacy by hiding them behind pseudonymous addresses.
This work aims at presenting *confidential transactions*, the first proposal for a transaction format with encrypted amounts in Bitcoin, which would strongly increase value privacy. It exploits homomorphic encryption which does not remarkably hurt universal validation of transactions, a crucial premise for the achievement of a distributed consensus on the order of valid transactions.

Acknowledgements

Chapter 1

Introduction

Privacy is fundamental in every financial and monetary system. Bitcoin should not make any exception. The same whitepaper [18] recognizes the need for privacy: it explains how users' privacy is hidden behind pseudonymous addresses,[1] but admits the flaws of this approach in terms of the possibility of the history of transactions being linked.
Bitcoin's blockchain structure seems not to be ideal for privacy and seems to presume an insurmountable trade-off for its achievement. Bitcoin's security model requires universal verification of the validity of each transaction, in turn needing public transaction data; indeed, the Bitcoin's blockchain is globally accessible and immutable, it takes track of all of the transactions ever happened and all these characteristics can be effectively harmful to privacy.
Moreover, this is also crucial as the lack of privacy affects Bitcoin's capacity to serve as money. For instance, it is detrimental for its fungibility.
Developers have worked since long time in the direction of improving privacy in Bitcoin. But on the one hand, integration to the main protocol purposely takes time; on the other hand, privacy-based solutions are often costly and this precludes their possibility of being soft-forked.
Confidential transactions [14] is the proposal for a kind of transactional format where each output amount is encrypted and thus hidden, but without preventing each transaction to be successfully validated by each node of the network. This is a consequence of homomorphic encryption and it demonstrates that it is possible to overcome the previously addressed trade-off.
The introduction of confidential transactions would bring consistent value privacy, but cannot protect against the possibility of linking transaction his-

[1] In the reality the white paper speaks of public keys rather than addresses. Indeed, in the whole paper there's no mention of addresses at all, but the first code release already provided this functionality.

tories. However it is the case that this proposal well integrates with some others affecting transactional graph privacy.
Such benefits would not come at zero cost. The described implementation of confidential transactions suffers from excessively burdening each transaction size and consequently is not ready yet for integration in the main protocol.

The present thesis has been written during the author's fellowship at the Digital Gold Institute, `https://www.digitalgoldinstitute.org`. The author has partly contributed to the Python library available at `https://github.com/dginst/BitcoinBlockchainTechnology`.

1.1 Structure of the thesis

In Chapter 2 we provide an introduction to transactions in Bitcoin which can help to highlight some aspects that would be recurrent when speaking of confidential transactions. In section 2.1 we give an overview of Bitcoin transactions, starting from their building block. Then from section 2.2 onwards we describe the basis of the Bitcoin scripting language.

In Chapter 3 the main aspects behind the lack of privacy and fungibility in Bitcoin are discussed. After a brief introduction, section 3.1 presents some of the tracks that privacy-based solutions are following and should follow in the future; moreover, it briefly describes some of these solutions. Sections 3.1.1 and 3.1.2 explain that confidential transactions address value privacy only, but show their compatibility with different privacy-based solutions. Then the discussion moves on fungibility; section 3.2.1 explores Bitcoin's lack of fungibility, while section 3.2.2 considers the relation with scalability.

In Chapter 4 we explore the cryptographic primitives which underlie the construction of a confidential transaction. In section 4.1 we provide a general definition and some examples of commitment scheme and we present the security properties it should satisfy. We proceed by defining the class of commitment schemes which succeed in making confidential transactions achieve their purpose: we introduce the additively homomorphic commitment schemes in section 4.1.1 and the Pedersen commitment in section 4.1.2. In section 4.2 various examples of Zero-Knowledge proofs of knowledge together with their distinctive properties are presented. In section 4.3 ring signatures are introduced; eventually in section 4.4 the elliptic curve Diffie-Hellman primitive is presented.

In Chapter 5 we eventually get into confidential transactions. We describe how the cryptographic primitives presented in the chapter above are specialized for their deployment. In particular, sections 5.1 and its related subsections describe how the amount encryption is achieved through Pedersen commitments and its consequences; section 5.2 explores the features of homomorphic encryption and particularly the consequences on transaction validation in section 5.2.1 and the setting of the blinding factors in section 5.2.2. Section 5.3 describes the first solution enabling the construction of the Zero-Knowledge range proofs, which play the fundamental role of preventing the possibility to exploit homomorphic encryption to "invisibly" build invalid transactions; sections 5.3.1, 5.3.2, 5.3.3, 5.3.4 describe how particular ring signature schemes can help in providing such proofs in Zero-Knowledge. Section 5.4 deals with the details of ECDH; in section 5.5 we eventually describe benefits and disadvantages.

Chapter 6 draws the conclusions to the work.

1.2 Notation

For what concerns cryptographic notation we specify it when needed (e.g. at the beginning of Chapter 4 or in the Appendix A).
For what concerns algorithms,

- $||$ refers to byte array concatenation;
- $a \leftarrow b$ refers to the operation of assignment;
- $z \xleftarrow{\$} Z$ denotes uniform sampling from the set Z and assignment to z;
- $a >>= b$ means $a = a >> b$, namely right-shift bits of a by b and assign the RHS to a.

Chapter 2

Transactions in Bitcoin

The aim of this chapter would be to endow the reader with a first set of basic instruments to follow the incoming description of the *confidential transactions*, which represent the core of this thesis. Indeed, the idea to start with transactions in Bitcoin should not surprise. This chapter can be considered a brief introduction that the informed reader can skip; a deeper inspection of the argument can be found for instance in [1, 6, 9].[1] Furthermore the chapter does not deal with SegWit[2] as it is out of the scope of this work.

2.1 Transaction outputs and other details

Transactions in Bitcoin enable the transfer of value between users of the network. One of the points that deserves to be stressed is that bitcoins only exist as *unspent transaction outputs* (UTXO) on the blockchain. Transactions spend these UTXO and generate new transaction outputs and at the same time transfer their ownership. Thus they configure as a transfer of these outputs which embed value in some way (we can say that bitcoins reside into them).[3] The so called UTXO set, which is the whole collection of the unspent transaction outputs scattered over the blockchain, continuously reduces in size as a consequence of spending the previously unspent outputs and grows as a transaction creates new ones.
Transaction outputs are indivisible chunks of currency (indivisible in the

[1] We even suggest the talk available at `https://www.youtube.com/watch?v=np-SCwkqVy4` which gives a general yet effective overview for what concerns Scripts.

[2] Refer to `https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki` or `https://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki`.

[3] Wallets store the private keys needed to spend these outputs, not bitcoins. Indeed, the ideas of the coin entity, rather than the balance of a wallet etc. are only abstractions that can be useful in figuring out the whole picture.

sense that can only be fully spent) which are recorded on the blockchain and are associated to addresses.[4] They embed the associated amount (denominated in satoshis[5]) and the mathematical puzzle, the so called *locking script*, that determines the conditions to spend them in the future.
Up to now we have only spoken of transaction outputs, which indeed are the building blocks of a Bitcoin transaction. However, it is the case that transactions can be visualized as a list of transaction inputs referencing and spending UTXO and generating new transaction outputs.[6] A transaction input holds a pointer[7] to the UTXO that it consumes (together with a zero-based index which identifies which UTXO from the previous transaction it references), it embeds an *unlocking script* that satifies the conditions for spending set in the UTXO and thus proves that the funds can be effectively spent. The unlocking script generally holds a digital signature, whose role is to prove the ownership of the referenced output and consequently allow its tranfer. Indeed, the whitepaper [18] defines a coin as a chain of digital signatures.
A transaction groups one or several inputs to be spent in order to generate the desired outputs. Indeed, given that outputs can be only fully spent, on the one hand transaction inputs could need to reference several UTXO to generate the output, on the other hand the transaction will generally spend back a change. The comparison with banknotes should be immediate. The action of combining UTXO in a proper way is provided by the user's wallet application; it is even the case that each user's wallet try to minimize the number of spent coins (and it is also incentivized to group inputs for an amount which is close to the one of the desired output) by running a coin selection algorithm. This incentive has several reasons: at first, linking outputs or even spending an high-valued output, while creating a small one is detrimental for privacy; then, because fees are paid based on transaction size as we will see in a while.
Moreover, it should be noticed that output amounts are not encrypted, but in the clear and this marks a crucial difference with respect to what will come later with *confidential transactions*.
Then, transactions are collected in blocks by some special nodes of the network which are called *miners*. Miners compete in a hash-based Proof-of-Work contest which determines who will add the next block of transactions to the

[4]Without entering into the details of their encoding, addresses are basically the result of hashing a public key.

[5]1 satoshi = 10^{-8} BTC.

[6]Transaction inputs are simply references to previously created outputs and this is the reason why it can be possible to explain transactions avoiding their introduction, from which the title of the section.

[7]A hash pointer to the previous transaction where the specific UTXO was created.

longer chain of blocks. Basically, miners do work over each block header; the last field of each block header is represented by a nonce. Mining is a race to find the proper nonce that makes the hash of the block start with a target number of zeroes (for a more detailed explanation about mining we refer to [6]). The winner of the contest (the first one that finds the solution and sends the proof of his work) is rewarded with the issuance of new bitcoins plus the fees of each transaction included in the block. The transaction which rewards the winning miner is the so called *coinbase transaction*; it is the first transaction in each block and it breaks the logic explained above because it does not consume UTXO, but instead has a special input called *coinbase*.
Fees are part of the compensation miners receive. They are computed based on the size of the transaction in kilobytes and for each transaction they are basically the difference between sum of inputs and outputs.
The last aspect we would like to mention in this brief introduction is transaction validation as it too highlights a significant difference with respect to what will come later with confidential transactions. When a node broadcasts a transaction, all the other nodes will verify its validity; they will check that each referenced output is effectively unspent, that the sum of the inputs exceeds the sum of the outputs and eventually check whether unlocking and locking scripts are consistent.
To conclude the section we think that it can be useful to describe an example of the workflow the transaction creator (let's say Alice) uses to send a transaction to the recipient (let's say Bob) and then what the recipient does to spend the output of this transaction.[8] Though the explanation of some of the possible constructions of locking and unlocking scripts (which in turn allow to define different types of transactions) is postponed to the section that covers the scripting language, we consider here the standard case of Alice paying to an address owned by Bob (the common Pay-To-Public-Key-Hash (P2PKH) transaction type). Alice creates a transaction output whose locking script allows whoever will be able to provide knowledge of the private key corresponding to Bob's address (thus hopefully Bob) to spend the output later. Then she broadcasts the transaction and the nodes of the network label the output as a UTXO. When Bob spends this UTXO, he creates a transaction input referencing the considered output and provides the unlocking script satisfying the conditions for spending. After broadcasting, when it comes to the network validating the transaction created by Bob, nodes evaluate the unlocking script provided by Bob and the retrieved locking script embedded in the UTXO previously created by Alice.

[8]Sender and recipient are some sorts of abstractions too, but well serve the purpose of supporting the comprehension.

2.2 Bitcoin scripting language

As we have already observed, in [18] a coin is defined as a chain of digital signatures, which allows the transfer from one user to another. It could be the case that on the one hand Bitcoin was initially designed to spend transaction outputs to public keys only, in such a case a digital signature provided with the private key associated to the public key in favour of which the referenced transaction output was spent would be enough to prove ownership and that on the other hand the scripting functionality was added together with the support for Bitcoin addresses. Indeed, there's no clue about scripts in the entire whitepaper although the first code release had already added the scripting functionality.
Restricting transactions just to the simple form described above would have not endowed Bitcoin transactions with the flexibility and the versatility that the scripting language effectively provides.
The Bitcoin scripting language, also called Script, is a stack-based programming language; data are pushed on top of a stack, its commands (the op codes) operate with these data (and eventually pop them). Moreover, it is not Turing-complete by design, namely scripts have limited complexity (e.g. no loops are admissible) and finite and predictable execution time. Indeed, this prevents giving the power to arbitrary users of the network to submit scripts that may create infinite loops, bringing nodes evaluating these to stall. Then, it is based on stateless verification, meaning that all the information needed to execute a script is contained in the script itself.
A script is represented by a list of data and commands (op codes) which are sequentially pushed on top of the stack and run and which implement a contract. Two only outputs are possible, namely true or false; if at the end of the execution the stack is non-empty and the top element is non-zero it returns true, false otherwise. The locking and unlocking scripts placed on a UTXO are written in this scripting language; when a transaction is broadcast, each node of the network will verify whether the transaction is valid through the execution of the unlocking and the locking script one at-a-time.[9]
Eventually, it could be the case that in the near future optimizations to the scripting language will take place. Indeed, the scripts execute on all nodes, but this is not really consistent with each node's intention to validate transactions; verification of the output would be enough.
Next section is devoted to the description of some of the most common script

[9]Before turning to separate execution of the scripts through a shared stack, code implementation concatenated unlocking and locking scripts and ran them together (as it were one only). This was changed because it was a bug which gave the possibility to spend anyone's coins.

templates, which allow to implement the different contracts (i.e. the different types of transactions) that Bitcoin provides.

2.2.1 Bitcoin script templates

- Pay-to-Public-Key (P2PK): it represents the very first transaction type. Its locking script allows to spend the output by providing proof of knowledge of the private key associated to the given public-key (i.e. providing a digital signature where the message signed is a hash of a specific subset of the data in the transaction).[10] P2PK was soon abandoned as publishing a public key on the blockchain is not quantum resistant. Here unlocking and locking scripts are reported.

```
scriptSig: <sig>
scriptPubKey: <pubKey> OP_CHECKSIG
```

  How the validation proceeds is described in Table 2.1. Consider a situation like the one of the simple example of section 2.1, with Alice and Bob involved in the transaction.[11] The unlocking script provided by Bob would be executed first and will add data on top of the stack; then each node will retrieve the locking script provided by Alice and run it.

Stack	Command	Description
	<sig>	
<sig>	<pubKey>	The scriptSig is pushed on top of the stack.
<sig> <pubKey>	OP_CHECKSIG	The public key is pushed on top of the stack.
1		The op code OP_CHECKSIG requires two inputs. It checks the consistency of the signature with the associated public key; if valid, it pushes true.

Table 2.1: Verification procedure for a P2PK transaction.

[10] Indeed, the locking script was first known as scriptPubKey. The reason is exactly that initially it contained the public key of the receiver. The same for the unlocking script: known as scriptSig as it contained a digital signature.

[11] Though there a P2PKH transaction was considered.

- Pay-to-Public-Key-Hash (P2PKH): it is the most common transaction type. Similar to the previous one, but the locking script allows spending the output by providing proof of knowledge of the private key associated to the given address. Publishing the address rather than the public key no longer suffers from not being quantum resistant. Here unlocking and locking script are reported.

```
scriptSig: <sig> <pubKey>
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY
              OP_CHECKSIG
```

  Here's the validation procedure.

Stack	Command	Description
	<sig> <pubKey>	
<sig> <pubKey>	OP_DUP	The scriptSig is pushed on top of the stack.
<sig> <pubKey> <pubKey>	OP_HASH160	OP_DUP takes the element on top of the stack and duplicates it.
<sig> <pubKey> <pubKeyHash>	<pubKeyHash>	OP_HASH160 takes the element on top of the stack and applies to it SHA-256 and RIPEMD160 in sequence.
<sig> <pubKey> <pubKeyHash> <pubKeyHash>	OP_EQUALVERIFY	The address <pubKeyHash> is pushed on top of the stack.
<sig> <pubKey>	OP_CHECKSIG	The op code OP_EQUALVERIFY takes the two elements on top of the stack and checks if they are equal. If not, the validation fails.
1		The op code OP_CHECKSIG requires two inputs. It checks the consistency of the signature with the associated public key; if valid, it pushes true.

Table 2.2: Verification procedure for a P2PKH transaction.

From now on, we will no more describe the verification procedure step by step as the logic should be clear.
It is worth noticing that with P2PKH the potential of the scripting language (with respect to the pure chain of digital signatures) begins to emerge; but it is then with *multi-signature* transactions that it starts to be truly evident.

- Multisignature (or *multisig*) scripts: called *m-of-n*, are such that n public keys are recorded in the locking script and require at least signatures for m of these public keys ($m \leq n$). These are limited to at most 15 public keys. Various use-cases are related to multi-signature transactions. Commonly these are used by multiple parties possessing the private keys necessary to unlock the funds; for instance, a wife and a husband could choose to lock funds in a 2-of-2 multisignature output that would prevent any of the two to unlock the funds without the other's approval. Then, multisig can be even useful for a single user to build a two-factor authentication wallet, where private keys are kept on different devices (which would make theft harder) or to protect against fortuitous loss of one of the keys (for instance with a 2-of-3 multisig). Here unlocking and locking scripts of a (bare) multisig are reported.

```
scriptSig: 0 <sig1> <sig2> ... <sigm>
scriptPubKey: m <pubKey1> <pubKey2> ... <pubKeyn> n
              OP_CHECKMULTISIG
```

  The above implementation of the scripts has some limits.[12] At first, the size increases linearly with the number of public keys and signatures. Then, it requires the signatures to be ordered with respect to the public keys, but especially it is beneficial for the receiver (who selects the multisig policy) and expensive for the sender, which is kind of odd. Indeed, the sender has to provide a locking script including all of the addresses, the number of valid signatures and public keys required and would eventually pay high fees being the script quite big in size; moreover, whatever policy the receiver choices the sender does not need to know it, quite the opposite. This would be detrimental to privacy. The solution to these issues came with P2SH.

- Pay-to-Script-Hash (P2SH): being the receiver in charge of the spending policy, he should pay for the increased script size. The way this burden is shifted on receiver side is by letting the locking script only hold the hash of the script used to spend a transaction (the *reedem script*).

[12]The starting 0 in the unlocking script is due to a bug in the OP_CHECKMULTISIG which has become part of the consensus rules.

```
scriptSig: 0 <sig1> <sig2> ... <sigm>
Redeem script : m <pubKey1> <pubKey2> ... <pubKeyn> n
                  OP_CHECKMULTISIG
scriptPubKey: OP_HASH160 <20 bytes hash of the redeem script>
              OP_EQUAL
```

The bigger redeem script has to be provided now by the sender, the locking script is, instead, much less big and only holds the hash of the redeem script. For what concerns verification, it will be checked whether the redeem script hashes to the same value specified in the locking script, then the redeem script would be processed against the unlocking script.

These are only some of the examples showing how far script flexibility can be pushed. Scripts are also useful to encode more complicated contracts.

Chapter 3

Privacy and fungibility issues in Bitcoin

Traditional, centralized financial institutions provide a level of privacy in their systems with respect to the outside world which is necessary for personal and commercial reasons at first (to preserve the freedom to transact, not to let commercial competitors monitor your own activity). The obvious concern, however, is that this level of privacy is not guaranteed against the same institutions providing the service and this opens a lot of issues regarding data collection and data privacy.
Bitcoin's breakthrough to have succeeded in building a decentralized distributed network where distributed consensus is reached among the nodes of the network has put some constraints on the underlying security model which seem to conflict with the privacy issues addressed above. However this is quite unavoidable: whereas decentralized systems are easy to build without consensus and, on the other hand, consensus is easy to achieve in centralized systems, maintaining both properties in the same system proves to be a hard challenge.
Bitcoin's security model is based on the achievement of distributed consensus which in turn requires (among the others) universal and independent verification of the validity of each transaction, made possible by having public transaction data. That is, transparency is needed to obtain and strengthen security.
This lack of privacy is recognized to be a weak point within the Bitcoin's protocol and several proposals to improve different aspects of privacy in Bitcoin have appeared in the years, despite never being effectively softforked. The main reason for this being basically the high costs that the massive adoption for privacy-based solutions would imply: privacy is costly and requires commitments. Nevertheless, advances in monitoring capabilities of

the blockchain, newborn businesses in Bitcoin blockchain's analysis urge the need for privacy-based solutions in the long run.
The original protocol has addressed the problem mainly through the adoption of pseudonymous addresses, that on one hand seem robust if one does not know who owns which addresses, but suffer both from some unsafe users practices (like address reuse) and from the possibility to exploit coins linkability[1] to trace the transaction graph.
Instead, a point in favour of having less privacy and more transparency comes from whom is mainly concerned with the use of cryptocurrencies like bitcoin for illicit activities, who argues that this could be a feature that can help investigations.

3.1 Types of privacy in Bitcoin

Due to how the Bitcoin's protocol has been conceived and implemented, the aim to strengthen privacy should regard different aspects whose improvement can be beneficial. In particular, the concern should be at least on *association* privacy, *balance* privacy, *identity* privacy and *transaction value* privacy.
Improving association privacy would mean enhancing transaction graph privacy not letting the possibility to understand who is paying who, thus addressing the previously discussed linkage between transactions and making transaction graph analysis harder. A typical practice undermining it is address reuse, that is however nowadays reduced by wallets using a Hierarchical Deterministic derivation of keys (and so addresses). Moreover, different solutions have been proposed in this field in the years. We present the idea behind only some of them, but it is worth to notice that solutions trying to address association privacy are many more.
One of the first is Coinjoin [13], which starts from the important premise that when a transaction spends from multiple addresses it is not necessarily the case that these addresses all come from the same party, but instead people could eventually cooperate to agree on a set of inputs to spend and a set of outputs to pay to and individually sign their own inputs only. On top of this it even exploits the absence of a mapping between inputs and outputs in a transaction or better the *many-to-many* mapping existing between them (in a transaction with more than one input, it is not possible to say which

[1]The need of a user to generally spend the change back to himself when transacting (due to transaction outputs generally not embedding the right value to be spent in a successive transaction) basically links transaction outputs and so addresses. Moreover in case previously collected changes are too small to cover a transaction output in its entirety, this makes it necessary to combine changes and so further linking transactions.

input ends up being which of the outputs or which part of) to create a single transaction jointly authored[2] by several participants in such a way that they do not have to trust each other. Indeed, each participant is only signing his own inputs (thus making it unnecessary to know who other is involved in the coinjoin) and it is the case that if some of the inputs are not signed the transaction would be invalid.
Observe moreover that the users involved in the coinjoin would even agree on a uniform output size and on burning inputs of at least that size. Indeed, unless all of them trading to the same amount it would be easy to discover the correlation between inputs and outputs.
Another solution which even provides association privacy, although was not born primarily for this purpose, is certainly Lightning Network [23], whose gain in transaction graph privacy derives for instance from the possibility of the parties taking part to a multi-hop channel to basically transact without sending data to the blockchain.
Improving balance privacy, instead, should aim at protecting against the possibility of deducing the balance of a wallet. Thus it is somehow linked to achieving association privacy.
Achieving identity privacy refers to the possibility of each user to prevent his identity to be associated with the coins. Identities could be at risk first of all because Internet itself is not really identity preserving (and not very anonymous); many services such as exchanges or on-line stores accepting bitcoins generally require and have access to personal information (credit card or bank account details, shipping addresses, IP addresses and so on).
The last privacy aspect which deserves credit is transaction value privacy, which actually is the main concern of the Confidential Transactions [14] solution. The idea is to protect against other people knowing the value of everyone's transactions, which is kind of standard for traditional financial services. That is, it affects the *confidentiality* Bitcoin transactions lack at all. Some concrete examples could stress the need for such improvements. For instance, a common implication for employees of a company paying wages in bitcoin is that they'll have their wages public, which is not that nice. Another possible example where amount privacy turns out to be necessary, though less conflicting common bitcoins' owners, configures when somebody's wallet spends a large sized input for a small payment, thus paying back to itself a high change; in this case, the possibility of being targeted for theft would be at least real.
As a final and general note, it should be observed that we have just described

[2]All of the transaction inputs are shuffled among several participants, each signing their inputs only.

a few of the bunch of proposals that could help achieving better privacy in Bitcoin and in particular we have considered the proposals more closely related to the addressed one. But, for instance, it is worth mentioning the impact that the introduction of Schnorr signatures [28] would have on privacy in Bitcoin mainly through signatures aggregation (even across signers), which is obtained by exploiting the linearity property of the Schnorr scheme.

3.1.1 Confidential transactions address value privacy

Confidential transactions is the first and only solution addressing value privacy in the Bitcoin ecosystem. All previous solutions mainly addressed association privacy. However, as well explained by Bitcoin Core developer G. Maxwell,[3] there should be a broader awareness on issues related to value privacy. For instance in the comparison with the Internet protocols, providing association privacy means anonymizing the identity of people communicating, which is not much of a worry unless for people using Tor. On the other hand, what people worry about is making the content of communications private, which is what value privacy effectively provides.
The way confidential transactions achieve value privacy is by encrypting the transaction amount (which is instead available in the clear in a standard Bitcoin transaction) and more precisely they exploit homomorphic encryption[4] to preserve the ability of the network to verify and validate transactions.
It should be noticed that confidential transactions only provide value privacy, not affecting transaction linkability, but they naturally integrate with various proposals addressing association privacy.

3.1.2 Compatibility with different solutions

As mentioned, it turns out that confidential transactions can not only integrate previous solutions addressing association privacy, but also help in solving some of their problems. In particular we focus again on the relation with Coinjoin [13], bearing in mind that most of the following arguments are even valid for similar proposals[5].
We have briefly described how Coinjoin works, but we have not focused yet on all of its limits. The first one is certainly coordination between users: it is not that easy to find people agreeing on transacting at the same time and for the same exact amount. The second one was briefly addressed above and

[3] During a conference whose video is available at `https://www.youtube.com/watch?v=LHPYNZ8i1cU`.

[4] More details will be available in the next chapters.

[5] E.g. Coinswap or Tumblebit.

it is basically the fact that Coinjoin achieves some sort of privacy provided that input and output values are somehow matching.
If we integrate with confidential transactions some of the previous issues disappear because having amounts encrypted prevents from the necessity to mix inputs of almost the same size to pay to outputs of the same size, while taking the advantage of achieving association privacy.

3.2 Fungibility

It turns out that fungibility is quite relevant in this discussion. Thus, we can start explaining what fungibility is. Fungibility is the property of a unit of a good to be completely indistinguishable from any other unit of the same good (or at least treated as such) and consequently completely interchangeable. To give some examples, diamonds are not completely fungible[6]: little differences in their properties (cut, hardness, color etc.) make it difficult to find diamonds expected to be equally valued. Another possible example of non-fungible good is a piece of art as it is clearly not possible to exchange one for one other.
For what concerns currencies, fungibility is a crucial property: we do not want to care of receiving a physical banknote being worth less than a different banknote of the same denomination nor we want to care of the possibility of the possession of this same banknote being revoked as it was involved in a robbery some transactions ago.[7] The possibility of blacklisting physical banknotes, other than being quite unfeasible, would destroy confidence in receiving them, thus impacting the whole economy. And actually that's not just a matter of practice but has been established by law. Differently from a stolen piece of art, whose possession would be revoked in the same moment of the discovery, that would not be ever the case for physical banknotes in most countries of the world.

3.2.1 Bitcoin is weakly fungible

When speaking of fungibility, what makes discussion on Bitcoin intertwine and compare with that on money in its cash-like forms, rather than with that on its inter-mediated means of payment is Bitcoin's peculiarity to be (substantially) immediate and final payment, exactly like cash. Moreover, inter-

[6]Gold is much more fungible.

[7]Though observe that even € or US$ or most of the other currencies are not completely fungible, they have serial numbers, but we basically treat them as such because non-fungible solutions for currencies wouldn't work.

mediated means of payment compromise some desirable features of money (among which fungibility itself).
Bitcoin turns out to be weakly fungible, which is the other recognized weak point of the protocol. The main reason for it should be found, again, in the transparency of the blockchain (feature and bug), which makes it possible in principle to trace the provenance of every coin and so discern between them. What actually happens is on one side to have some coins which are worth more than others,[8] the ones with less value becoming the preferred ones to be exchanged; on the other side, the growth of businesses specialized in the analysis of the transactions flows on the blockchain might turn somebody unwilling to accept certain coins further or exchanges to freeze accounts just for the "bad history" of a coin.
Lack of fungibility could even have more risky consequences for Bitcoin itself. It can jeopardize its permissionless nature because receiving coins and be prevented from spending can make users doubt of whether it's safe to receive and in turn can make them begin to consult blacklist services before transacting again. Or it can lead to a generalized loss of confidence which would make prices drastically decline.
As a side note, it should be said that there have been in time various proposals to create services to register Bitcoin users (kind of blacklisting services) and among the invoked reasons behind their possible adoption was always the idea of reducing Bitcoin-related crimes. However, it is likely that criminals already circumvent the regulated exchanges when buying or selling bitcoins, thus being probably not affected.
Based on the described picture it is evident that the same aspects making Bitcoin non-private make it also non-fungible.
Thus among the solutions favouring fungibility we can include the same privacy-based solutions outlined before. Then, another important aspect fungibility benefits from is mining decentralization[9] which guarantees that sooner or later a user's transaction will be processed and mined without miners discriminating between coins in the act of processing transactions.

3.2.2 Fungibility vs scalability

Scalability is another highly debated aspect in Bitcoin. It refers to the possibility of increasing the transaction capacity, making the network able to process more transactions per second. In particular, it is in the comparison with centralized payment solutions, such as Visa, that the discussion gets

[8]Think of freshly minted coins that, being "clean", can be traded for a higher premium.
[9]Though note that mining in Bitcoin is quite centralized.

going. The reality is that, by design, Bitcoin is not suited to process the volume of transactions of a centralized circuit like Visa[10] and this for simple reasons: each transaction in Bitcoin is broadcast to and through all of the nodes of the network, each of whom has to keep an updated copy of the entire ledger of transactions; centralized solutions on the other side only require a centralized ledger to which all transactions are committed and a few back-ups. If Bitcoin processed the same number of transactions of the Visa circuit this would result in a bloat of the blockchain size, running totally outside the realm of processing power of available computers.
Moreover, scalability in Bitcoin is also affected by transactions confirmation being slow and probabilistic compared to centralized systems, where confirmations happen in fractions of a second.
Thus, it is likely that higher scalability will mainly come through off-chain solutions exploiting the blockchain for verification of balances (and settlement of disputes, see e.g. [23]) rather than transfers.
Far from being a discussion on scalability issues and proposed solutions, this paragraph only wants to briefly investigate the connection between fungibility (and privacy) on one side and scalability on the other side. Indeed, the described expensive nature of privacy-based solutions generally put fungibility and scalability in a seemingly insurmountable trade-off. However, there are situations where fungibility helps scalability: the reduction of information leakage (in particular that information enabling transactions to be linked and reducing anonymity) would potentially help scalability by preventing relevant information to go and appear in the blockchain. Mimblewimble [12] is an example of an application that provides privacy and fungibility (mainly through the adoption of confidential transactions) but even achieves better scalability than Bitcoin by the possibility to remove most of historic data by pruning spent transaction outputs and possibly validating the whole history without downloading these already spent transactions.
Conversely, having a more scalable network would obviously favour fungibility and let open the possibility to exploit more expensive privacy-based solutions, but it is indeed difficult to achieve.

[10] According to `https://usa.visa.com/run-your-business/small-business-tools/retail.html`, Visa can be able to process 24000 tps at its peak, Bitcoin only 7 tps.

Chapter 4

Cryptographic primitives

In this section we present the cryptographic primitives which are necessary to build a *confidential transaction*.
Prior to this, however, we start with a crash dive into some pillars of Elliptic Curve Cryptography (ECC), the focus being just on what can be useful to follow the incoming narration. We refer to [2, 20] for a more deep approach.
Elliptic Curve Cryptography is a public-key cryptosystem built on elliptic curves defined over finite fields and, for our purposes, it is the cryptosystem Bitcoin uses to secure the transactions. It is based on the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP), namely the infeasibility of computing the discrete logarithm of a random elliptic curve point with respect to a publicly known base point[1]. The benefits over its prior alternatives (in the field of public-key or asymmetric cryptography) come from the possibility of providing the same security level with shorter operands, which is in turn a consequence of the problem being harder to solve. Indeed, it is even the latest solution which has come out among the mentioned alternatives.
Referring to the Appendix A for both the definition of finite field (and how to get to it) and the presentation of the DLP (in its non-elliptic curve formulations) to avoid making this introduction unintentionally cumbersome, we give instead here the definition of *elliptic curve* and we present the known

[1] At the base of public-key cryptography there is always the intractability of a particular mathematical problem:

- RSA public-key schemes: hardness of factoring large integers.
- DLP-based public-key schemes: hardness of solving the discrete logarithm problem.
- ECDLP-based public-key schemes: hardness of solving the generalized discrete logarithm over an elliptic curve.

translation of the DLP over elliptic curves (ECDLP).
The need to introduce elliptic curves is motivated by the necessity of searching a cyclic group where to build the cryptosystem[2]; observe, however, that the mere existence of a cyclic group is not sufficient, the problem being to find such one where the DLP is computationally hard to solve. It turns out that elliptic curves are fitted for the purpose and thus the goal becomes to find elliptic curves with a large cyclic group. Later in this section, a theorem will support the suitability of elliptic curves in providing such a result, thus explaining their fundamental role in the discussion.
As remarked within the Appendix A, for cryptographic use, the focus is just on elliptic curves defined over a finite field (in particular we consider $K = \mathbb{F}_p$, the finite field with p elements) rather than over generic fields (the set $\mathbb{R}$ being an example).

Definition 4.0.1. *The elliptic curve over* $\mathbb{F}_p$[3] *is the set of all pairs* $(x, y) \in \mathbb{F}_p$ *such that* $\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + ax + b \mod p\} \cup \{\infty\}$ *with* $a, b \in \mathbb{F}_p$ *and such that the consistency condition* $4a^3 + 27b^2 \neq 0 \mod p$ *holds true.*

The problem now becomes both identifying the group elements and defining a group operation with these elements. Group elements are nothing else than the points fulfilling the curve equation in Definition 4.0.1. The definition of the group operation is not presented here, for the details we refer to [20].
The only point we want to stress is the meaning of $\{\infty\}$ in Definition 4.0.1:

Remark 4.0.1. $\{\infty\}$ *is the so called infinity point and turns out to be the identity element of the group defined by the points over the elliptic curve together with its group operation.*

At this point, it is possible to state the following theorem, which eventually closes the circle by explaining the reason why it is possible to build a DLP with elliptic curves.

Theorem 4.0.1. *The points on an elliptic curve, together with the infinity point* $\{\infty\}$ *(other than defining a group by themselves once the group operation is defined) have cyclic subgroups. Moreover, under certain conditions (merely if the group order is prime) all points on an elliptic curve form a cyclic group.*

Remark 4.0.2. *By specializing the notation seen in the Appendix A to the elliptic curve case, we denote by* G *the generator of a cyclic (sub)group defined over an elliptic curve (being an element of the (sub)group itself, it is nothing*

[2]See DLP arguments in section A.3 for more details.
[3]Which we denote by $E(\mathbb{F}_p)$.

else than a point on the curve). Thus, starting from G it is possible to explore the entire (sub)group (thus recovering all the (sub)group points) by repeatedly applying the group operation to G.

Remark 4.0.3. *If the EC (sub)group has order m, the application of the EC group operation m times gives back the identity element of the EC group.*

Remark 4.0.4. *If the EC group order n is prime, any point of the curve is a generator G. This basically comes from Theorem A.1.3.*

With this theoretic framework at our disposal, we can conclude the introduction to this chapter by presenting the ECDLP.

Definition 4.0.2. *Given the elliptic curve $E(\mathbb{F}_p)$ with generator G, consider another element of the curve, Q. The ECDLP is finding the integer $q, 1 \leq q \leq \#E(\mathbb{F}_p)$, such that $\underbrace{G + G + \cdots + G}_{q \quad times} = qG = Q$.*

For what concerns notation, in ECC q represents the *private key*, which is an integer, while Q represents the *public key*, a point on the curve with coordinates (x_Q, y_Q).

4.1 Commitment schemes

Commitment schemes are widely used in modern cryptographic protocols and play a fundamental role in confidential transactions, as will be soon evident.
A commitment scheme is a cryptographic primitive that allows a prover to keep a piece of data secret to a verifier, but commit to it so as to prevent a prover's change of mind aimed at tampering with the underlying committed secret data. Moreover, a commitment scheme provides the following characteristics, namely the fact that it should be at least computationally infeasible to deduce the underlying data from the commitment itself, while being immediate to verify (once the secret is known[4]) that with high probabilistic assurance the committed data has effectively produced that commitment.
The most trivial example of commitment scheme in cryptography is represented by a cryptographic hash, which indeed fulfills previous requirements (even if an input to a hash function is generally salted to prevent brute-force attacks). Alternatively, it could be funny to observe that the use of commitment schemes would enable two players to play a game of head-or-tails

[4]Though observe that the prover does not have to reveal his choice, while he might do at a later time in some protocols.

remotely[5] (*coin flipping by telephone*), a game that couldn't be played with the certainty of not being cheated.
We give now a more formal definition of commitment scheme and then we concentrate on its security properties. Notice that different definitions are possible depending on the setting and at the same time it is difficult to find one covering all the possibilities. Driven by this evidence, the definition we present is mainly fitted to the application we are going to present in the next chapters and, despite arrangements in notation, is taken from [22].
Let $(\mathbb{G}, \circ)$ be an elliptic curve group of prime order n. Let C denote the (public) commitment.

Definition 4.1.1. *Let r, $v \in \mathbb{Z}_n$. A commitment scheme is a pair of algorithms:*

- $commit(r, v) \to \mathbb{G}$
- $open(r, v, C) \to \{True, False\}$

such that $open(r, v, commit(r, v)) \mapsto True$ $\forall (r, v)$ in the domain of commit.

Then, when it comes to security properties of commitment schemes, the so called *hiding* and *binding* properties should be defined.

Definition 4.1.2. *A commitment scheme is said to be perfectly (computationally) hiding if the distribution of $commit(r, v)$ for uniformly random r is equal (computationally indistinguishable) for fixed values of v.*

Basically, a hiding commitment does not reveal the data it commits to. In particular, at its highest extent it prevents even an adversary endowed with infinite computing power to deduce information on the committed values from the commitment only. If computationally hiding, it makes it computationally unfeasible (hard in PPT).[6]

Definition 4.1.3. *A commitment scheme is said to be perfectly binding if $\forall (r, v)$ in the domain of commit, $\nexists (r', v') \neq (r, v)$ such that $open(r', v', commit(r, v)) \mapsto True$; it is said to be computationally binding if no PPT algorithm can produce such a (r', v') with non-negligible probability.*

[5] In this case with the obvious necessity of the prover revealing the committed data at some time.

[6] Probabilistic Polynomial Time. PPT algorithms are those algorithms taking polynomial time ($\sim O(n^\alpha)$, $\alpha > 1$) to get a probabilistically correct solution.

A binding commitment cannot be opened to a different input. Once a commitment $C(r, v)$ to (r, v) has been made, one cannot later open it as a commitment to (r', v'). Same considerations as before are valid here, with obvious shift in reasoning.
Moreover, although it would be nice to have a commitment scheme satisfying both properties at the highest extent, we anticipate here that it is not actually possible to have such one as perfect hiding excludes perfect binding and viceversa. More formal arguments will come with the description of the Pedersen commitment.[7]

4.1.1 Additively homomorphic commitment

Additively homomorphic commitment schemes constitutes a class of commitment schemes which is at the basis of confidential transactions and which falls under the wider research area of homomorphic encryption. Homomorphic encryption schemes allow performing computations on encrypted data without leaking information about underlying ones (and without having access to them) and getting the same results as if operations were performed on unencrypted data.
In particular, as the name should suggest, additively homomorphic commitments preserve the sum and more precisely are those for which the following holds (again, we refer to the notation introduced above).

Definition 4.1.4. *A commitment scheme is additively homomorphic if* $commit(r, v) + commit(r', v') = commit(r + r', v + v')$ *and the distributions of* $commit(r, v) + commit(r', v')$ *and of* $commit(r + r', v + v')$ *coincide.*

4.1.2 Pedersen commitment

The Pedersen commitment is a commitment scheme initially introduced in [21] as one with homomorphic properties. In [21], it is needed to construct an efficient and non-interactive scheme for verifiable secret sharing (when combined with the Shamir's scheme[8]). This is an application where a dealer holds a secret and wants to distribute it *in shares* to n shareholders he does not trust (and by whom he is not trusted) in such a way that any subgroup of at least k shareholders can recover the secret by exploiting the additively homomorphic property of the Pedersen commitment, while any subgroup of less than k shareholders can learn nothing about the secret.
Going further this application, here we concentrate on the commitment

[7] See section 4.1.2.
[8] Needed to make the entire scheme non-interactive.

scheme only and we present its elliptic curve version. Some preliminaries are necessary before the definition.
Let $\mathbb{G}$ be a EC group of prime order n. Let G, H be fixed *nothing-up-my-sleeve* (NUMS) generator points of $\mathbb{G}$.

Definition 4.1.5. *The EC NUMS points are points whose elliptic curve discrete logarithm (ECDL) relative to each other are unknown.*

Remark 4.1.1. *Indeed, the following are unknown:*

- *x s.t. $xG = H$ ($G^x = H$ in multiplicative notation, $x = \log_G H$);*
- *y s.t. $yH = G$ ($H^y = G$ in multiplicative notation, $y = \log_H G$).*

Definition 4.1.6. *Let r,$v \in \mathbb{Z}_n$. Let r be chosen at random.*
Define a Pedersen commitment (to v) as the following scheme:

$$\begin{aligned} commit\colon \mathbb{Z}_n^2 &\to \mathbb{G} \\ (r, v) &\mapsto rG + vH \end{aligned}$$

where we refer to the general definition 4.1.1 for the *open* algorithm (which accordingly takes input in $\mathbb{Z}_n^2 \times \mathbb{G}$).
The Pedersen commitment turns out to satisfy the following properties.

Proposition 4.1.1. *Let the ECDLP in $\mathbb{G}$ be hard.*
The Pedersen commitment is an additively homomorphic commitment scheme providing the following security properties:

- *perfect hiding;*
- *computational binding.*

Here's the intuition driving the proof.
Additively homomorphic: the property comes from the point addition operation on EC points defining a group (when adding the $\{\infty\}$ point) and the Pedersen commitment being basically an EC point.

$$\begin{aligned} commit(r, v) + commit(r', v') &= (rG + vH) + (r'G + v'H) \\ &= (r + r')G + (v + v')H \\ &= commit(r + r', v + v'). \end{aligned}$$

Perfect hiding: different (r, v) pairs satisfy $C = rG + vH$.
In particular, fixed v, $\exists!\ r = G^{-1}(C - vH)$ s.t. $C = rG + vH$ (indeed $G^{-1} \neq 0$ being G a generator).

Thus, an exhaustive search cannot discern in between (r, v) pairs, which in turn implies that an adversary cannot get information about the committed data v from the commitment C itself, while r is chosen to be random according to Definition 4.1.6. As a side note, it will be soon clearer that it's the randomness of r to make the scheme perfectly hiding.
Moreover, not being able to distinguish in between (r, v) pairs also means that more than one (r, v) pair can be a valid opening for the commitment C, making it intrinsically non-perfectly binding and confirming the incompatibility between perfectly hiding and perfectly binding commitments stated before.
Computational binding: given a Pedersen commitment to (r, v), opening it to a different pair (r', v') would require breaking the ECDLP on $\mathbb{G}$ which is computationally unfeasible for regular[9] computers . In particular:

$$rG + vH \stackrel{?}{=} r'G + v'H \leftrightarrow r' = G^{-1}(rG + vH - v'H) = r + (v - v')\underbrace{G^{-1}H}_{x:xG=H}$$

Observe that it is the NUMS hypothesis on generators G, H to make it computationally binding.

The emphasis on the security properties of the Pedersen commitment makes it interesting to take a further step in explanation. As anticipated, its non-perfectly binding nature makes it exposed to attacks focused on breaking the ECDLP and it is widely known that developments in quantum computing would probably enable to provide such a result (for instance by Shor's algorithm[10]). On the other hand, its perfectly hiding nature would provide privacy guarantees (impossibility to unblind the commitment) even in a scenario in which the ECDLP is broken: indeed, $C = rG+vH \leftrightarrow G^{-1}C-v\underbrace{G^{-1}H}_{known} = r$, but still for any v $\exists! r$ s.t. $C = rG + vH$.
Moreover, the discussion is active on whether to prefer perfectly hiding or perfectly binding solutions on newborn protocols (like Mimblewimble[11]). The highly debated trade-off turns in balancing the desire for privacy with the belief in the likelihood of quantum computers coming into being. But, on the one hand, except for undisclosed advances in quantum computing, ECDLP breaking won't happen overnight; on the other hand, a viable perspective could be having both a perfectly hiding and a perfectly binding chain with cross-chain pegs (see [7]), with coins coming into existence on one of the two,

[9] As opposed to quantum.
[10] `https://en.wikipedia.org/wiki/Shor%27s_algorithm`.
[11] See e.g. `https://lists.launchpad.net/mimblewimble/msg00114.html` up to the end of the thread.

but possibly transferable on the other chain and back.
Finally, it should be noticed that other protocols make use of different commitment schemes providing perfect binding instead (Chain[12] for instance exploits the so-called ElGamal commitment scheme).

4.2 Zero-Knowledge Proofs of Knowledge

Zero-Knowledge proofs of knowledge (ZKPoK) are proofs that yield nothing but their validity.
There is a subtle distinction between Zero-Knowledge proofs and (Zero-Knowledge) proofs of knowledge: in a ZKP the prover tries to convince the verifier that some statement is true and the proof reveals to the verifier no additional information apart from the fact the statement is true; in a PoK the prover tries to convince the verifier that he is in possession of some secret information and he cannot succeed in convincing the verifier unless he truly knows some secret. Then a PoK can be proved in zero-knowledge, thus providing a ZKPoK.
Proofs that can be performed in zero-knowledge have the following characteristics. First of all, if the statement is true, the verifier will be eventually convinced of this by the prover; instead, if the statement is false, no cheating prover can convince the verifier of the contrary. Finally zero-knowledgeness is enforced by the fact that if the statement is true, no verifier can learn anything else than the fact that the statement is true. Interestingly enough, various examples of ZKPoK can be successfully presented without entering into the math that underpins these primitives.

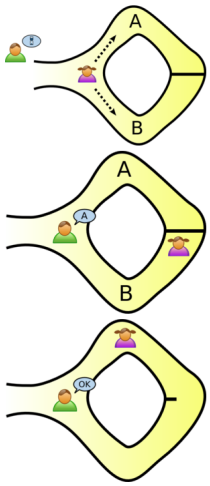

Figure 4.1: Ali Baba cave

This one is the Ali Baba cave example. The cave is ring-shaped and has a magic door (whose opening requires a secret key) at the opposite side of the entrance (Victor the verifier is aware of its presence). Peggy the prover enters the cave and randomly chooses a path (A or B) without being seen by Victor. Then Victor eventually enters and shouts the name of the path he wants Peggy to take when coming outside.

[12] `https://github.com/chain`

Provided that Peggy effectively knows the secret key, she can simply open the door and exit where requested. On the other hand, if she doesn't know the secret key, her 50% probability of guessing the right path at the first attempt would eventually vanish after several repetitions of the experiment. Another common example is the graph 3-colorability problem, that the interested reader can find in [9].
Eventually, we would like to present an example with higher cryptographic relevance. It turns out that a prover can possibly prove in zero-knowledge that he knows the discrete logarithm q of a given value $c = g^q$, g being the generator of the underlying cyclic group (according to notation in Appendix A.1.1). There are many of such proofs; we present the Schnorr protocol (in its interactive form).

- The prover selects a random number r, computes $g^r \mod n$ and sends it to the verifier.
- The verifier challenges the prover by sending to him a random value e.
- The prover replies to the challenge computing $u = r + e \cdot q \mod n$ and sending it to the verifier.
- The verifier accepts if $g^u \mod n = g^r \cdot c^e \mod n$. Indeed, $g^u \mod n = g^{r+e \cdot q} \mod n = g^r (g^q)^e \mod n = g^r c^e \mod n$.

Once again, the proof is successfully carried out in zero-knowledge as the verifier can get no clue about q and at the same time would unmask a cheating prover.
The provided examples are kinds of interactive proofs we are not strictly interested in[13], but succeed in giving an impressive idea of how these proofs work. Their non-interactive counterparts (in which there's no interaction between prover and verifier) are obtained via the Fiat-Shamir heuristic, where the challenges by the verifier are replaced by the result of the hash of some intermediate result.

4.3 Ring signatures

Initially introduced in [25], ring signatures are a cryptographic primitive allowing for any actor in a group to provide a digital signature[14] on behalf of

[13]Indeed as we will see, the solution confidential transactions adopt is non-interactive. Moreover, even in previous subsections we have always presented the primitives in their non-interactive form.

[14]We are not interested in giving a formal definition of digital signature and we refer to [20].

the group itself, but preserving the anonymity of the signer (*signer ambiguity*). More precisely, the actual signer can select any set of potential signers which includes himself (provided he knows a public key for each of them) and signs a message with his own private key in a way that the whole signature does not leak any information about who has effectively produced the signature. Moreover, the selected possible signers are usually not aware of the fact that their public key has been used in a ring signature scheme (like a multiparty 1-of-m signature scheme without cooperation from the other $m-1$ members). Thus, it shouldn't be much of a surprise that one of the described application in [25] was whistleblowing, namely the possibility of leaking a disgraceful secret of a group of people (think as an example to an episode of corruption inside a city council) by one of the members of the group in a way that the whistleblower could not be identified, but at the same time with certainty that the leak came from a member of the group.
Given that, a ring signature scheme could be seen as a variant of a digital signature scheme in which the single verification (public) key is replaced by a ring of verification (public) keys and one private key only is required to produce a valid signature; moreover, given that the actual signer does not know the private keys corresponding to the public keys of the other group members, these are generally forged (but obviously indistinguishable to real ones).
A ring signature scheme involves two procedures:

- $ring-sign(m, Q_1, Q_2, \ldots, Q_r, q_r)$: it produces a ring signature σ for the message m given the public keys $Q_1, Q_2, \ldots, Q_r$ of the r ring members, together with the private key q_r of the r^{th} member (the signer).

- $ring-verify(m, \sigma)$: it accepts a message m and a signature σ (including the public keys of all the possible signers) and outputs either true or false.

We are not going to present here the algorithm that underlies the construction of ring signatures in [25] both because it would require migrating to the RSA[15] setting and because in the next section we will directly describe the version adopted by confidential transactions. Before passing to the next primitive, we only summarize all of the features of ring-signature schemes. Other than being signer-ambiguous signature schemes (there is no possibility to revoke the anonymity of the signer and more precisely a verifier has a probability equal to $\frac{1}{r}$ to guess the identity of the real signer), they are setup-free signature schemes. This means that there is not a pre-arranged

[15] `https://en.wikipedia.org/wiki/RSA_(cryptosystem)`

group of members (this is particularly relevant in the comparison with group signature schemes from which ring-signatures derive and whose underlying mechanism is basically the same apart from the presence of a trusted group manager who selects the ring members and can unmask the anonymity of misbehaving signers). In the end, it has to be noticed that the signature size grows linearly with the number of rings.

4.4 Elliptic Curve Diffie-Hellman

The elliptic curve Diffie-Hellman primitive is a cryptographic primitive which is at the basis of the ECDH Key Exchange scheme.
In turn, the elliptic curve Diffie-Hellman Key Exchange (ECDH) is a key agreement scheme based on ECC, thus relying on the hardness of the ECDLP. It is the EC counterpart of the well known[16] Diffie-Hellman Key Exchange (DHKE) protocol. In this section we just present the scheme for the elliptic curve version.
It allows two entities, both endowed with an elliptic curve private-public key pair, to engage in a key agreement scheme and establish a shared secret over an insecure (yet authenticated) channel. The shared secret can then be both used directly as key or as seed to derive other key(s), for instance (but it is just a possibility) via a deterministic generation procedure like RFC6979 (see [24]).

4.4.1 ECDH primitive

The primitive is built in such a way that both the parties by means of one of their own private key and one of the public key of the other party can recover, autonomously, the same (shared) secret.
Here how's the primitive built. Suppose Alice and Bob want to establish a shared secret. The primitive is run autonomously by both and takes as input valid elliptic curve domain parameters T[17], a private key owned by who is running the procedure (q_A for Alice, q_B for Bob) and the public key corresponding to the other party private key (Alice takes $Q_B{=}q_BG$ as input, Bob takes $Q_A{=}q_AG$ as input[18]). The output is a shared secret field element

[16] In cryptography at least.

[17] $T = (p, a, b, G, n, h)$; p specifies the prime finite field $\mathbb{F}_p$; $a, b \in \mathbb{F}_p$ are the coefficients of the elliptic curve equation; $G \in E(\mathbb{F}_p)$ is the elliptic curve generator point; n is the order of G, that coincides with the number of points of the cyclic subgroup generated by G; $h = \#E(\mathbb{F}_p)/\ n$ is the so called cofactor.

[18] Though, Alice does not obviously know q_B, nor Bob q_A.

z or the string "invalid" otherwise.
The algorithm below refers to the generation of the shared secret from Alice. The same can be done for Bob, by carefully exchanging the roles of the parameters.

Algorithm 4.1 ECDH primitive

1: **procedure** ECDH(T, q_A, Q_B)
2: $S = (x_S, y_S) \leftarrow q_A Q_B$
3: assert $S \neq \{\infty\}$
4: **if** $S = \{\infty\}$ **then**
5: **return** "invalid"
6: **end if**
7: **return** $z \leftarrow x_S \mod n$
8: **end procedure**

It turns out to be straightforward to prove that the shared secret computed by both parties is the same. Alice: $q_A Q_B = q_A(q_B G)$; Bob: $q_B Q_A = q_B(q_A G)$ $\rightarrow$ by associativity of the group operation (point addition), the result holds and both compute $S = q_A q_B G$.

4.4.2 ECDH Key Exchange protocol

According to notation in [2], the key exchange protocol involves a *setup* phase, a *key deployment* procedure and a *key agreement* operation (which actually exploits the ECDH primitive). Here's a simple graphical representation of the key exchange protocol combining the key deployment phase (in its associated public keys exchange phase) and an instance of the ECDH primitive explained above.

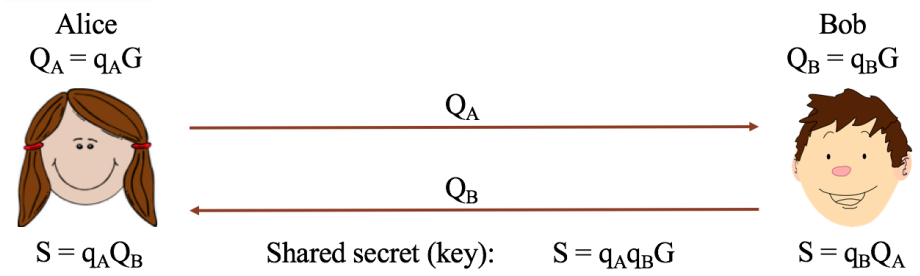

Figure 4.2: ECDH key exchange

The *setup* phase defines the choice of the elliptic curve domain parameters. The *key deployment* phase requires both parties establishing valid private-public key pairs $\{q_A, Q_A\}$, $\{q_B, Q_B\}$ and exchanging their public keys Q_A

and Q_B (with further assurance of the public keys being valid ones).
The *key agreement* operation defines a way to obtain shared keying data from a shared secret by means of a suitable key derivation function. Observe, however, that the keying data might not directly be used as a key, this just depends on the application.
The algorithm below refers to the generation of the shared key by Alice. Bob's algorithm is straightforward and just requires rearrangement of the parameters. We first describe the key derivation function (KDF), taking as inputs an octet[19] string Z derived from the field element $z \in \mathbb{F}_n$ outputted by the ECDH primitive,[20] the integers *keydatalen*, *hashlen*, *hashmaxlen* which refer to the length in octets respectively of shared key, hash values and maximum lenght of messages that can be hashed with the given hash function. It outputs a shared key K (octet string) or "invalid".

Algorithm 4.2 Key Derivation Function

procedure KDF(Z, $keydatalen$, $hashlen$, $hashmaxlen$)
  assert $|Z| + 4 < hashmaxlen$, "invalid"
  assert $keydatalen < hashmaxlen * (2^{32} - 1)$, "invalid"
  $count \leftarrow 1$ ▷ 4 octet, big-endian string
  for $i \leftarrow 1, \lceil keydatalen/\ hashlen \rceil$ **do**
    $K_i \leftarrow hash(Z||count)$
    $count \leftarrow count + 1$
    $i \leftarrow i + 1$
  end for
  $K \leftarrow K$>>=$(keydatalen - hashlen)$ ▷ leftmost keydatalen octects of $K_1||\ldots||K_{\lceil keydatalen/\ hashlen \rceil}$
  return K
end procedure

Thus, the whole key agreement operation requires deriving a shared field element $z \in \mathbb{F}_n$ by running an instance of the ECDH primitive, converting it to an octet string to give as input to the KDF and consequently deriving a shared key.
For what concerns security arguments, it is required the ECDHP (elliptic curve Diffie-Hellman problem) to be hard to solve. It comes out it is closely related to the ECDLP, a third party willing to break it has to solve either $q_A = \log_G Q_A$ or $q_B = \log_G Q_B$. Consequently, provided a care choice of the parameters during the setup phase, the more powerful algorithms to break

[19] To keep it simple we can consider an octet being equivalent to a byte.
[20] For more details on the *field-to-octet* conversion, see [2].

it are basically the same presented above for the ECDLP ($\sim O(|\mathbb{G}|^{\frac{1}{2}})$ steps). What that means practically is that with an elliptic curve group order higher than 2^{160} (i.e. 160 bits)[21] the ECDHP wouldn't be possibly broken.
Then, another important security assumption relies on the so-called channel authentication for public key exchange, required to prevent *man-in-the-middle* attacks. The latter consist in an adversary eavesdropping over the channel and substituting Alice and Bob's public keys with his own public keys. If the channel is not authenticated,[22] this would prevent Alice and Bob being aware of sharing a secret with the eavesdropper rather than themselves.

[21]Bearing in mind that, as a rule of thumb, security levels of more than 80 bits can be considered satisfactory.

[22]Where authentication means that the recipient has strong reasons to believe that the communication is in place with the "designated" sender and can be achieved by means of some protocols we do not present here.

Chapter 5

Confidential transactions

The aim of this chapter is to present the construction of a *confidential transaction*; in particular, we will see how the cryptographic primitives presented in the previous chapter are deployed (or specialized) and we try to give an overview of benefits and downsides.
For this reason we will initially see how a confidential transaction achieves value privacy by encrypting the output amounts through Pedersen commitments and we will describe all of the consequences of homomorphic encryption with an impact on the way confidential transactions are built; then, we will add the range proofs to the scheme, which prevent a malicious transaction's creator to exploit modular arithmetic (and encryption of values) to build invalid transactions in an undetectable way. These proofs are provided in Zero-Knowledge through a particular variant of ring signature. Then, we will enter in the details of the communication between transaction's sender and receiver. Indeed encryption would prevent even the receiver to know the amount associated to each output; thus, some sort of communication is necessary.
We conclude the chapter stressing the stunning fact that this whole construction can be obtained without requiring any new cryptographic assumption with respect to the ones already securing Bitcoin; on the other side, we recap its disadvantages.

5.1 Transaction amount encryption

Confidential transactions achieve their goal of obscuring output amounts by replacing output amounts in the clear (Bitcoin-like) with Pedersen commitments to the output amount. This results in substituting the 8-byte integer representing the output amount in a standard Bitcoin transaction with a 33-

byte commitment to the amount. Indeed, recalling Definition 4.1.6, Pedersen commitments are nothing else than elliptic curve points (the co-domain of the mapping being the elliptic curve group $\mathbb{G}$) which in compressed form can be represented by a 33-byte sequence.
On the basis of the previous description of the cryptographic primitive in section 4.1.2, this section will be devoted first of all to its transposition in confidential transactions.
As an example, we start describing a step-by-step possible strategy to obtain safe encryption of the amounts which culminates in having amounts coded by Pedersen commitments. At each step, we will present pros and cons of the considered solution.
Consider a standard transaction with 2 inputs and 2 outputs. The "balance" of such a transaction in Bitcoin would be represented by the following equation: $v_{i1} + v_{i2} = v_{o1} + v_{o2}$ (where i stands for input, o stands for output). While allowing easy verification that no money has been created out of thin air, such a representation lacks of confidentiality. Next step could be embedding vH in a transaction rather than v only: $v_{i1}H + v_{i2}H = v_{o1}H + v_{o2}H$. Indeed, being H an elliptic curve generator (refer to section 4.1.2 for details) the equality holds and verification is still easily guaranteed to work. However, this solution suffers from the number of values v being finite and thus from the possibility of an attacker trying to guess the data underlying the commitment. Moreover, observe that knowing v_iH would unmask all of the outputs of value v_i across the blockchain. The final solution to these issues consists in keeping the skeleton of the previous proposal, but blinding the committed value via a random factor, thus effectively coding the amounts through Pedersen commitments: $(r_{i1}G + v_{i1}H) + (r_{i2}G + v_{i2}H) = (r_{o1}G + v_{o1}H) + (r_{o2}G + v_{o2}H)$. In this case the verification of the balance of the equation is guaranteed by the additively homomorphic property stated above, its perfectly hiding nature preserves the confidentiality.
The interpretation of the parameters r and v should be now immediate: r represents the secret random *blinding factor*, v represents the committed amount. Thus, in confidential transactions the Pedersen commitment commits to an amount v and requires a random blinding factor r (a private key basically) to obtain the security property of perfect hiding. Each committed amount v and blinding factor r (one per input/output) has to be known only to participants in the transaction; the whole commitment C has to be public instead.

5.1.1 NUMS generators construction

Pedersen commitments are built through nothing-up-my-sleeves (NUMS) generators. On the one hand, generator G has not a clear origin (it is not NUMS in principle); on the other, the Bitcoin protocol specification does not define a second generator associated to the curve.[1]
The problem becomes to construct one satisfying Definition 4.1.5. After all, the consequences of having a second generator H not properly built would be catastrophic, because it would make possible to open the commitment to a different value (possibly inflating the currency), tampering with its binding property.
For instance, consider the case in which the second generator H is chosen by a malicious designer who knows the (elliptic curve) discrete logarithm with respect to G, i.e. he knows r_H such that $H = r_H G$. Thus, $C = rG + vH = rG + vr_H G$. In such a case, it is easy to produce an example in which a commitment to a value v is effectively opened to a different value v'. It would be sufficient to make a commitment to a value v, $C_v = rG + vH = (r + vr_H)G = ((r - (v' - v)r_H) + v'r_H)G$ and then make the same commitment commit to value v' by publishing $\hat{r} = (r - (v' - v)r_H)$ instead of r as blinding factor. Indeed, $C_{v'} = (\hat{r} + v'r_H)G$ turns out to be a commitment to value v'. It is clear instead that he would be unable to use the same trick without knowing r_H.
The way such a problem could be generically solved[2] is by picking H through the hash of an encoding of generator G (of its x-coordinate in particular) and coercing the hash to a curvepoint. Indeed, just hashing the generator could not be sufficient as it couldn't possibly result in obtaining a curve point. In such a case, it suffices to keep on incrementing the obtained hash-value until getting a valid curve point. Confidential transactions exploit this technique.
To conclude, it is worth to notice that being the procedure to build the second generator public (and consequently being H hardcoded and available) is not a sufficient condition to deduce the elliptic curve discrete logarithm as at least hashing operations are involved.

[1] The elliptic curve Bitcoin uses is the so called secp256k1, which is a 256-bit elliptic curve over the finite field $\mathbb{F}_p$ and the specification defines the tuple of parameters $T = (p, a, b, G, n, h)$ among which generator G, but not generator H.

[2] See for instance `https://crypto.stackexchange.com/questions/25581/second-generator-for-secp256k1-curve`.

5.1.2 Explicit fees

Although we need some more steps to figure out completely how a confidential transaction is built, the reader could have possibly already raised an issue with respect to the way fees are managed in such a design.
Indeed if one thinks to a standard Bitcoin transaction, fees are what comes out from the difference of amounts associated to transaction inputs and transaction outputs[3] and can be easily deduced being the amounts associated to inputs and outputs public.
In confidential transactions, the encryption of amounts effectively prevents the same mechanism to work. Not being possible to deduce the fee amount f implicitely, this is explicitely published as a plaintext.
In turn this has consequences on the commitment scheme. Being fees paid "unmasked", a Pedersen commitment to fee amount is conceived as $C_f = fH$, rather than $C_f = rG + fH$. Indeed, there's no need to blind anything and it is built with blinding factor $r_f = 0$.
For what concern transaction propagation, nodes will just check that $\sum_i C_i^{inp} - \sum_i C_i^{out} - fH = 0$ holds, before propagating the transaction further.

5.2 Homomorphic encryption features

It turns out that the additively homomorphic property of the Pedersen commitment plays a key role in the design of confidential transactions. In particular it affects the very first issue that one could think of when speaking of confidential transactions, namely how can transaction verification happen. Indeed, as by Proposition 4.1.1, Pedersen commitment is additively homomorphic in both inputs and it is exactly this property to still allow easy verification of the validity of a confidential transaction[4]. What this means is that considering a transaction with 2 inputs and 2 outputs, the following holds: $v_{i1} + v_{i2} - v_{o1} - v_{o2} = 0 \leftrightarrow C_{i1} + C_{i2} - C_{o1} - C_{o2} = 0$.
Then it also affects the way blinding factors are set.

5.2.1 Commitment to value 0 & network verification

As mentioned, the nodes of the network would verify the validity of a transaction and would eventually propagate it by checking whether the commitments to inputs and outputs sum to 0.
On the other hand the additively homomorphic property ensures that having

[3] Recall that $\sum_i TxO_i + fee = \sum_i TxIn_i$.
[4] Despite inputs/outputs amounts not being public.

a sum of commitments to a total value of 0 is just the same as having a single commitment to $v = 0$ (although such a commitment will never appear). Given that, we anticipate here a result that would be fundamental when we will deal with ring signatures in confidential transactions.
A commitment to value $v = 0$ gives the opportunity to create a digital signature with that commitment as though it was a public key (and $v = 0$ is special in this sense, you cannot do it with commitment to $v \neq 0$). Indeed, given $C = rG + \cancel{vH}_{v=0} = rG$ you can produce a digital signature with C as verification public key and r as the corresponding private key. Instead, if $v \neq 0$, you would be stuck not knowing the elliptic curve discrete logarithm of C with respect to G because of the addition of vH (i.e. basically because of the NUMS hypothesis on the generator points): $C = rG + vH \rightarrow \underbrace{?}_{unknown} = \log_G C$.
At first sight the previous paragraph can be puzzling, but recall that[5] it is only possible to create a digital signature with a point which is a multiple of G. By definition, a signature with private key q can be verified with public key qG; if $v \neq 0$, it is impossible to find q such that $qG = rG + vH$ as it would require knowing the ratio (i.e. the elliptic curve discrete logarithm) between G and H.
Eventually, this means that a way a Pedersen commitment can be proven to be a commitment to $v = 0$ consists in signing a transaction with the commitment as public key (the message signed is the hash of this commitment public key so as to bind the signature to the commitment), the blinding factor as private key.

5.2.2 Blinding factors setting

Being the Pedersen commitment homomorphic in both inputs by Definition 4.1.4, this has a consequence on the way blinding factors are managed: in order for all the commitments in a confidential transaction to sum to 0, not only committed values, but also blinding factors need to sum to 0.
Consider as an example the usual transaction with 2 inputs and 2 outputs and let Alice be the sender and Bob the receiver. Suppose moreover that Bob will receive a transaction output only, the other coming back to Alice as change.
In such a situation, blinding factors from the inputs are already set as they are associated to commitments to the transaction outputs referenced by the actual transaction inputs. Let's say they are r_A, r_B. Thus, Alice is left with one degree of freedom in choosing r_C, r_D (the ones associated to the outputs)

[5] As successfully explained in `https://bitcoin.stackexchange.com/questions/54042/how-does-a-range-proof-bound-lower-at-0-and-not-1`.

correctly. For instance she could set r_C as $r_C = r_A + r_B - r_D$ (or she could do the same with r_D). Then she will send r_C to Bob[6], which is the blinding factor associated to the commitment to the transaction output he will obtain, but without r_A, r_B, r_D being disclosed.
At first sight, such a construction seems to have a security flaw. Indeed, in the example above Alice knows the blinding factor associated to the transaction output sent to the receiver (and in a more general example he would know all of the blinding factors associated to transaction outputs), which would give her the possibility to spend this output. Though observe that here we are just describing the skeleton of a confidential transaction, while not specifying the conditions for spending as they are specific to the different protocols exploiting confidential transactions.
In [12, 22] the issue is addressed by allowing all the commitments to inputs and outputs in a transaction to sum to a non-zero value kG, k being the so-called *excess value* (another private key, but chosen by the recipient). This in turn still allows the verification of the validity of the transaction. Indeed, the total amount v of the transaction is still null, which according to section 5.2.1 enables to provide a signature with the commitment as public key. Therefore, each Mimblewimble transaction will include a digital signature provided with the excess blinding factor as private key.
On the other hand, this would prevent Alice to be able to spend Bob's output(s).

5.3 Zero-Knowledge range proofs

The tremendous potential of homomorphic encryption does not come without flaws when applied to Bitcoin. Since the mathematics underlying commitments occurs over a finite field, addition is modular and wraps around: $a + b = c \mod n \rightarrow a + b = c + kn$.
More precisely, the elliptic curve group $\mathbb{G}$ is cyclic having prime order n (which is a 256-bits prime number) which implies that the outlined scheme based on verifying whether Pedersen commitments to transaction amounts sum to 0 is insecure without additional measures. What can happen is that addition of large values can overflow (indeed a small amount can be in the same equivalent class, modulo the field order, of a very large positive amount) or that negative values can succeed in providing a valid transaction. As a consequence, overflow can basically allow to print an unlimited amount of coins illegally and in such a way that it would be impossible to discover. On the other hand if negative amount were valid, it would be possible to create

[6]The safe transfer of blinding factors will require a section on its own.

coins from nothing.
As a simple example, yet far from real parameters, overflow can work this way. Consider to have a curve with prime order $n = 13$ and a standard 2 inputs - 2 outputs transaction.

Inputs	Outputs
$C(1, r_A)$	$C(8, r_C)$
$C(1, r_B)$	$C(7, r_D)$

Table 5.1: Modular addition: example of wrapping

In such a situation, matching of input and output commitments yields 0 ($1 + 1 - 8 - 7 = -13 \mod 13 = 0$), thus the network would validate it. Meanwhile the majority of the coins (13 of 15) has been created illegally.
For what concerns the negative-amount-like behaviour, the following could happen:

Inputs	Outputs
$C(1, r_A)$	$C(5, r_C)$
$C(1, r_B)$	$C(-3, r_D)$

Table 5.2: Modular addition: negative-amounts

Again, the transaction is still well balanced, there's a creation of coins from nothing and no easy detection (even if v is negative, vH is a usual elliptic curve point).
The introduction of Zero-Knowledge range proofs is thus required to prevent wrapping: these configure basically as additional pieces of data which prove each commitment being genuine. In particular, range proofs are a cryptographic tool proving that each committed output is within a certain range ensuring that no overflow is possible and amounts are non-negative (e.g. $[0, 2^{32})$ satoshi). Moreover, they are Zero-Knowledge proofs of knowledge and so they prove the committed amount is in range without disclosing neither the amount nor the blinding factor.
Two main approaches to range-proof construction are available in literature. This work concentrates on the first solution to the problem, which exploits a particular variant of ring signatures, *Borromean ring signatures* ([17]). In recent times, a new and more efficient solution, *Bulletproofs* ([8]), has come out.

5.3.1 Enforce zero-knowledgeness: ring signatures

A basic example can motivate the need for providing such a proof in Zero-Knowledge.

Suppose Alice, the prover, wants to prove to Bob that C is a commitment to the value $v = 1$, without telling him the blinding factor r, but the value being known.
What Bob (the verifier) can do is to compute $C' = C - 1H$[7] and ask Alice to provide a signature (with respect to G) with public key C'. If Alice is able to provide a valid one, then C has to be a commitment to $v = 1$. Indeed from section 5.2.1, we know that it is possible to provide a digital signature with the commitment as public key only with commitments to $v = 0$. If C is a commitment to $v = 1$, then $C' = C - 1H = rG + \cancel{1H} - \cancel{1H} = rG$ (from which $C = rG + 1H$) and Alice knows the blinding factor r to sign with C' as it is the one she has set for the commitment C.
However, the exposition of the amount would be detrimental to the fulfillment of value privacy, which is the primary purpose of confidential transactions. From here the need to enforce zero-knowledgeness by avoiding giving away the amount. It turns out that this issue can be efficiently approached through ring signatures and next sections will explain how.

5.3.2 Role of ring signatures in confidential transactions

In section 4.3 we have described what ring signatures were born for, namely hiding signer's identity in a group of potential signers. However, this is not the only application for which ring signatures can be useful. For instance, in confidential transactions the use of ring signatures is somehow specialized due to their intrinsic nature.
Rather than hiding the transaction creator among a group by signing the transaction with a ring signature, in confidential transactions ring signatures will be used to prove that single bits (or digits, depending on the considered encoding) of the encrypted output amount are in a certain range (e.g. {0,1}) without giving away the actual number. In particular, the signer will produce a "ring" of commitments for each digit of the encrypted output amount. In turn, from the number of overall commitments (after the encoding of the encrypted amount is made public) one can deduce the length in digits of the encrypted amount the whole commitment commits to. For instance, considering a binary representation, it would possible to encode a 3-bits amount with 6 Pedersen commitments (2 per bit) and in turn a 3-bits number can be at most in [0,8). Thus if valid it will prove to be a commitment to a value in that range. More technical details will come in next sections.
To have an idea of how this can work let us see how the example of section

[7] Recall that C is public and this would just be a point addition operation on the curve.

5.3.1 can be specialized with ring signatures.
This time the proof will not give away the amount. In particular, Alice can prove to Bob that C is *either* a commitment to $v = 0$ or to $v = 1$, thus it is in range $\{0, 1\}$.
While Alice provides a ring signature over $\{C, C'\}$, Bob can compute again $C' = C - 1H$. If compared with the example in section 5.3.1 where the committed value was known, this time it should be less clear the reason why Bob would compute the same quantity as before. However a single ring with two commitments only should suggest a binary encoding for the committed amount, thus motivating the choice. Then,

- if C is a commitment to $v = 1$, Alice does not know its discrete logarithm (recall what we have said in section 5.2.1), but C' becomes a commitment to $v = 0$ of which she knows the discrete logarithm and makes her able to provide a valid signature over the ring;
- on the contrary, if C is a commitment to 0, she knows its discrete logarithm, she doesnt for C';
- if C is a commitment to any other amount, none of them commits to $v = 0$, thus preventing Alice to be able to sign.

Observe that this can work for any pair of numbers or it can even work to process larger rings.

5.3.3 AOS ring signatures

Introduced in [3] by Abe, Ohkubo and Suzuki (from which they derive the name), they are a particular variant of ring signatures able to gain consistent reductions in size and verification time with respect to earlier ring signatures schemes. It is worth to notice that when used with discrete-log type keys only (because they allow to use both public keys for integer factoring based schemes and the ones for discrete-log based schemes), they yield much shorter signatures with respect to previous ring signature schemes.
The design of the ring is basically the same described in section 4.3, where a ring of r verification public keys requires knowledge of *one* of the corresponding secret private keys. Moreover, given that all of the verification public keys play the same role in verifying the signature, the specific signing key remains secret.
They provide "OR proofs": given a ring with r public keys, the ambiguous signer proves to know $\{q_0$ OR q_1 OR $\ldots$ OR $q_r\}$.
Before entering in the details of the full signature algorithm, it is necessary

to describe the way a signer can produce a valid signature in a ring, yet not a ring signature. Let the signer be endowed with his private-public key pair $\{q, Q\}$. This single signature generation involves:

- picking up a random nonce $k \rightarrow K = kG$;
- given the message m to be signed, computing $hash(K||m) \rightarrow e = hash(K||m)$;
- signing as follows: $s = k + eq$;
- publishing the signature as (s, e, m).

In turn, its verification (given s, e, m, Q as inputs) would imply computing $sG - eQ$. Indeed, $sG - eQ = sG - eqG = (s - eq)G = kG = K$; then with K at disposal the verifier could possibly reconstruct $hash(K||m)$ and verify whether it equals e (given as input). If so, the signature is valid.
Thus, it can be noticed that it is nothing else than one of the variants of a Schnorr signature (according to [28]); the transition to a valid *ring signature* would involve chaining these single signatures together and the way this is done is explained through the full signature algorithm.
The complete signature algorithm is the following. Let m be the message to be signed, i be the the index corresponding to the verification public key Q_i in the ring and i^* the index corresponding to the unique private key. It outputs the signature σ for (m, Q_i).

Algorithm 5.1 AOS ring signature: signature algorithm

procedure AOS-SIGN(m, q_{i^*}, Q_i: $0 \leq i \leq r-1$)
  Initialization:
  $k_{i^*} \xleftarrow{\$} \{1, ..., n-1\}$ ▷ n: elliptic curve group order
  $K_{i^*} \leftarrow k_{i^*}G$
  Forward sequence:
  for $i \leftarrow i^*+1, \ldots, r-1, 0, \ldots, i^*-1$ **do**
    $e_i \leftarrow hash(K_{i-1}||m||i)$
    $s_i \xleftarrow{\$} \{1, ..., n-1\}$
    $K_i \leftarrow s_iG - e_iQ_i$
  end for
  Forming the ring:
  $e_{i^*} \leftarrow hash(K_{i^*-1}||m||i^*)$
  $s_{i^*} \leftarrow k_{i^*} + e_{i^*}q_{i^*}$
  return $(e_0, s_0, \ldots, s_{r-1}) =: \sigma$
end procedure

The key points of the algorithm are the following. At first, all of the s-values except for the one which effectively closes the ring are forged.
Then, in principle the signature would include the complete set of e-values and s-values $(e_0, s_0, e_1, s_1, \ldots, e_{r-1}, s_{r-1})$. However, as e-values can be deterministically retrieved once the first e-value and all of the s-values are published, it suffices to publish $(e_0, s_0, s_1, \ldots, s_{r-1})$, an $(r+1)$-dimensional tuple of 32-bytes numbers.

Instead, the verification algorithm is the following. It takes as input the message m, the whole published signature σ and the set of verification public keys Q_i, $0 \leq i \leq r-1$. It clearly shows that the signature is valid if the computed e_0 coincides with the e_0 provided as first entry of the tuple σ.

Algorithm 5.2 AOS ring signature: verification algorithm

1: **procedure** AOS-VERIFY$(m, \sigma, Q_i : 0 \leq i \leq r-1)$
2: **for** $i \leftarrow 0, \ldots, r-1$ **do**
3: $K_i \leftarrow s_i G - e_i Q_i$
4: $e_{i+1\%r} \leftarrow hash(K_i || m || i+1\%r)$
5: **end for**
6: **if** $e_0 = 0$ or $e_0 \geq n$ **then** ▷ n: elliptic curve group order
7: **return** "false"
8: **end if**
9: **if** $e_0 = \sigma[0]$ **then**
10: **return** "true"
11: **end if**
12: **end procedure**

The algorithm shows that the procedure obviously starts at the node for which the e-value is known. A valid signature will just ensure that one of the private keys corresponding to the set of public keys Q_i has signed without giving any clue on which effectively did.

5.3.4 Borromean ring signatures

Introduced in [17], Borromean ring signatures are a generalization of the AOS ring signature scheme. They generalize the previous construction as they overcome the simple "OR proofs" and they can represent proofs of knowledge of more general functions (any monotone boolean function is fine) of the signing keys.
While the AOS construction chained signatures together to form a ring, here

entire rings of signatures are chained together providing a single and more compact structure of the one that would arise from having r (r being now the number of rings) separate AOS structures. Indeed, the use case of the Borromean ring signatures in confidential transactions is the one in which r separate AOS-like structures and consequently r rings of public keys are needed.[8] Though we do not enter now in the details of the benefits of Borromean ring signatures, it is worth to notice that they provide a significant space saving for what concerns signature size.
The ambiguous signer can now prove to know one of $\{q_{0,0}$ OR $q_{0,1}$ OR $\dots\}$ AND one of $\{q_{1,0}$ OR $q_{1,1}$ OR $\dots\}$ AND $\dots$ AND one of $\{q_{r-1,0}$ OR $q_{r-1,1}$ OR $\dots\}$, where the first index represents the ring, the second one represents each public key in the ring (the number of public keys in each ring can in principle differ from ring to ring, but in the described construction it will be the same).
Thus, a Borromean ring signature configures as a signature on a message m that can be produced by a signer that knows *all* of the private keys of all the rings of verification public keys (one per ring) or by parties knowing all of them.
Moreover, it is the way in which Borromean ring signature is designed to make it achieve its compactness, which in turn affects the signature size (a more compact structure in this case means a structure which requires less public keys for verification). If we represent each ring by means of a graphical structure, we can think of it as a graph with nodes, each referring to a public key and labelled by its e-value. The AND/OR gate described above is obtained without requiring multiple graphs, but by means of a graph structure made of r rings, with the optimization of pinning a node (e.g. node 0) as a shared node chaining all the rings. The shared node has an ingoing and an outgoing edge per ring, which should be pretty clear given that it closes all of the rings. Eventually, multiple s-values are associated to the shared node and they either require a real private key or force a random e-value by picking up a random s-value (in analogy with the description provided in section 5.3.3).
A graphical representation can help in figuring out the overall structure. Figure 5.1 considers the example of a structure made of 4 rings all closing in e_0. Dashed edges represent those for which s-values are connected to known private keys.

[8] We will enter in the details later in this section. However, we have already given a hint in section 5.3.2: basically a ring per digit of the encrypted output amount is needed.

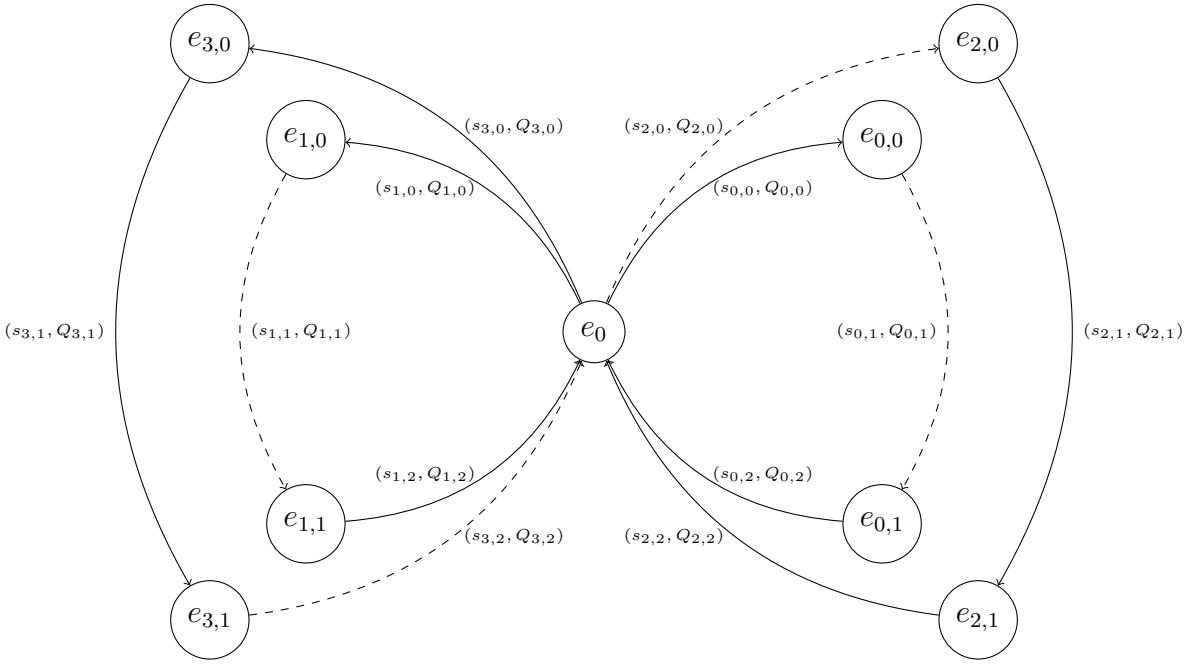

Figure 5.1: Graphical structure behind Borromean ring signatures

Source: adapted from [17]

The complete signature algorithm is the following. Let $Q_{i,j}$, $0 \leq i \leq r-1$, $0 \leq j \leq m_i - 1$ be the collection of verification public keys. In particular, i indexes each ring (these are r rings to be chained together), while j indexes each public key (m_i public keys per ring in total). Thus, $Q_{i,j}$ represents the j^{th} public key in the i^{th} ring.
Moreover, let m be the message to be signed and $\{q_i\}_{i=0,\dots,r-1}$ be the private key[9] corresponding to $\{Q_{i,j_{i^*}}\}_{i=0,\dots,r-1}$, j_{i^*} being the index of the signing private key of the i^{th} ring. Thus, $Q_{i,j_{i^*}}$ represents the public key in the i^{th} ring whose corresponding private key is the signing one (i.e. the one that closes the ring).

[9]There's no need to reference the position in the ring with a second index as there is a known private key per ring only.

Algorithm 5.3 Borromean ring signature: signature algorithm

procedure BORROMEAN-SIGN($m, q_i : 0 \leq i \leq r-1, Q_{i,j} : 0 \leq i \leq r-1, 0 \leq j \leq m_i - 1$)
  $M \leftarrow hash(m||Q_{0,0}||Q_{1,0}|| \ldots ||Q_{r-1,m_{r-1}-1})$
  for $i \leftarrow 0, \ldots, r-1$ **do**
    $k_i \xleftarrow{\$} \{1, ..., n-1\}$ ▷ n: elliptic curve group order
    $j_{i^*} \xleftarrow{\$} \{0, ..., m_i - 1\}$
    $R_{i,j_{i^*}} \leftarrow k_i G$
    if $((j_{i^*} + 1) \ \% \ m_i) \neq 0$ **then**
      for $j \leftarrow (j_{i^*} + 1) \ \% \ m_i, \ldots, m_i - 1$ **do**
        $s_{i,j} \xleftarrow{\$} \{1, ..., n-1\}$
        $e_{i,j} \leftarrow int(hash(M||R_{i,j-1}||i||j))$
        assert $e_{i,j} \neq 0$ and $e_{i,j} < n$
        $R_{i,j} \leftarrow s_{i,j}G - e_{i,j}Q_{i,j}$
      end for
    end if
  end for
  $e_0 \leftarrow hash(M||R_{0,m_0-1}|| \ldots ||R_{r-1,m_{r-1}-1})$
  for $i \leftarrow 0, \ldots, r-1$ **do**
    $e_{i,0} \leftarrow int(hash(M||e_0||i||0))$
    assert $e_{i,0} \neq 0$ and $e_{i,0} < n$
    for $j \leftarrow 1, \ldots, j_{i^*}$ **do**
      $s_{i,j-1} \xleftarrow{\$} \{1, ..., n-1\}$
      $e_{i,j} \leftarrow int(hash(M||s_{i,j-1}G - e_{i,j-1}Q_{i,j-1}||i||j))$
      assert $e_{i,j} \neq 0$ and $e_{i,j} < n$
    end for
    $s_{i,j_{i^*}} \leftarrow k_i + q_i e_{i,j_{i^*}}$
  end for
  return $(e_0, s_{i,j} : 0 \leq i \leq r-1,\ 0 \leq j \leq m_i - 1) =: \sigma$
end procedure

The algorithm produces a valid signature for the whole structure, namely it provides a signature of knowledge of the r private keys $\{q_i\}_{i=0,\ldots,r-1}$ corresponding to public keys $\{Q_{i,j_{i^*}}\}_{i=0,\ldots,r-1}$.
For what concerns the computation of the s-values and e-values and focusing on single rings, it can be noticed that in analogy with the AOS construction each edge i from a node labelled with its e-value is characterized by a s_i value which is either random or equal to $s_i = k_i + q_i e_i$ and each node with a single incoming and outcoming edge (all but the shared one) has an associated

e-value of the type $e_i = hash(m||s_{i-1}G - e_{i-1}Q_{i-1})$.
On the other hand, the unique node with multiple ingoing and outgoing edges computes the e-value as $e_0 = hash(m||s_{0,m_0-1}G - e_{0,m_0-1}Q_{0,m_0-1}|| \dots ||s_{r-1,m_{r-1}-1}G - e_{r-1,m_{r-1}-1}Q_{r-1,m_{r-1}-1})$ as it closes r rings.
Some other remarks would help in explaining the whole algorithm. First of all, notice the concatenation of both ring number and position in the ring into the computation of e-values which guarantees the impossibility of moving rings around without breaking the signature. This of course does not apply to the shared node (node 0) as it is common to all rings.
For what concerns the message, it is unique for the whole structure, not ring-specific. This shouldn't be surprising if one recalls that the message to be signed is the hash of the commitment. Thus, being the whole chain of rings to commit to a single commitment, this implies the message being unique.[10]

Verification happens according to Algorithm 5.4. Indexes obviously run as before and the algorithm takes as inputs the message m, the whole signature σ and the set of verification public keys $Q_{i,j}$, $0 \leq i \leq r-1$, $0 \leq j \leq m_i - 1$. Observe that it is much simpler than the signature algorithm. It avoids the two phases of the signature procedure as it does not depend on which signing keys per ring are known (of course because a verifier does not know them). Before entering in the details of how Borromean ring signatures are effectively deployed, let us evaluate their benefits in the comparison with the equivalent structure made of AOS ring signatures. Indeed, the construction above should have pointed out that a Borromean ring is structurally equivalent to a number of AOS rings chained in a single shared node. Consider r rings and N verification public keys per ring.

	Signature size
r AOS ring signatures	$r \cdot N + r$ (32-bytes numbers)
Borromean ring signature	$r \cdot N + 1$ (32-bytes numbers)
Δ	**r - 1** (32-bytes numbers)

Table 5.3: Borromean ring signature: signature size

The Table 5.3 summarizes signature sizes and computes the saving obtained via Borromean ring signatures. First of all, observe that verifying r AOS ring signatures requires $r \cdot N$ s-values (N per ring) and r e-values (1 per ring) being published. A Borromean ring signature requires instead a unique e-value (the shared one), from which one can easily derive the overall saving.

[10]In the AOS ring signature construction each ring commits to a single commitment, thus the message becomes ring-specific.

Indeed, it is worth to notice that as the number of rings increases the saving becomes significant. For the small structure in Figure 5.1 with 4 rings and 3 verification public keys each we get already a $\sim 20\%$ shorter signature.

Algorithm 5.4 Borromean ring signature: verification algorithm

procedure BORROMEAN-VERIFY($m, \sigma, Q_{i,j} : 0 \leq i \leq r-1,\ 0 \leq j \leq m_i - 1$)
  $M \leftarrow hash(m||Q_{0,0}||Q_{1,0}|| \dots ||Q_{r-1,m_{r-1}-1})$
  for $i \leftarrow 0, \dots, r-1$ **do**
    $e_{i,0} \leftarrow int(hash(M||e_0||i||0))$
    if $e_{i,0} = 0$ or $e_{i,0} \geq n$ **then** ▷ n: elliptic curve group order
      return "false"
    end if
    for $j \leftarrow 0, \dots, m_i - 1$ **do**
      $R_{i,j+1} \leftarrow s_{i,j}G - e_{i,j}Q_{i,j}$
      if $j \neq m_i - 1$ **then**
        $e_{i,j+1} \leftarrow int(hash(M||R_{i,j+1}||i||j+1))$
        if $e_{i,j+1} = 0$ or $e_{i,j+1} \geq n$ **then**
          return "false"
        end if
      end if
    end for
  end for
  $e_0^{'} \leftarrow hash(M||R_{0,m_0-1}|| \dots ||R_{r-1,m_{r-1}-1})$
  return $e_0^{'} = e_0$
end procedure

This section ends with the explanation of how Borromean ring signatures effectively provide Zero-Knowledge range proofs. Their role was previewed in section 5.3.2.
We have already defined range proofs as additional pieces of data attached to each transaction output proving that the committed output amount lies in a predetermined range of positive values. The first point that needs clarification is the definition of what a proper range would be. This depends on the kind of range proof exploited (in terms of bits of randomness one wants to achieve). For instance, a standard 32-bit range proof limits to a range in between 1 and $2^{32} - 1$ satoshis (42.94967295 BTC). However this in principle does not prevent larger outputs to be created, both because any value in between 2 and 64 bits of randomness can be chosen (provided the relative range

is made public) and because confidential transactions even support scaling the amount by a power of 10. This latter feature means that provided it is specified that the proof does not deal with satoshi units, but rather with multiples of them, it is possible to process amounts in higher ranges through small proofs.[11] For instance referring to the above 32-bit proof, dealing with 10^6 satoshis as unit, it would be possible to embed all possible amounts ranging in between 0.01 and 42949672.95 BTC.
It is also clear that it is even possible to use large ranges for small proofs, but this would make the proof slower to create and verify and it would even imply higher fees.
Then it comes the time for the sender to effectively build a range proof per transaction output. As first step, this implies considering the committed output amount (in a given encoding) and ring-signing (with a Borromean-style ring signature) over each digit. The encoding chosen by confidential transactions for the output amount is a base-4 encoding and the reason behind such a choice is that it minimizes the overall number of commitments sent.
We start by building the range proof step by step, then we will justify the base-4 representation. For simplicity of notation we consider the 32-bit range proof described above. The procedure is the following:

- consider the *output amount* in its *base-4 expansion*;[12]

$$v = v_0 \cdot 4^0 + v_1 \cdot 4^1 + v_2 \cdot 4^2 + \cdots + v_{15} \cdot 4^{15}.$$

- *Ring-sign over each digit*, which implies:
  - committing through a Pedersen commitment to each digit value, i.e. building $C_i = r_i G + v_i 4^i H$, $i = 0, \ldots, 15$, making sure that the sum of commitments is exactly C, the commitment to the value associated to the transaction output;
  - "arranging" 16 rings of signatures (one per digit) with 4 verification public keys per ring, each committing to one digit value v_i in $0, \ldots, 3$;
  - providing (for each digit) a Borromean ring signature over the ring

    $$\{r_i G + v_i 4^i H, r_i G + v_i 4^i H - 4^i H, r_i G + v_i 4^i H - 2 \cdot 4^i H, r_i G + v_i 4^i H - 3 \cdot 4^i H\}.$$

    Thus on the one hand each ring proves each digit to commit to a value $v_i \cdot 4^i$, $i = 0, \ldots, 3$ (without knowing which one), from which it should

[11] As explained in `https://bitcoin.stackexchange.com/questions/46865/confusion-of-confidential-transactions`.
[12] A 32-bits number can be represented with 16 digits in base-4.

be clear that the resulting Borromean ring signature proves the whole commitment to range in $[1, \sum_{i=0}^{15} v_i \cdot 4^i]$ satoshi. On the other hand, if $v_i \notin \{0,1,2,3\}$ none of the commitments would be a commitment to $v_i = 0$, thus preventing the ring to be signable according to the arguments given in section 5.2.1 and to the example presented in section 5.3.2.

- *Publish* the *range proof* associated to the transaction output.
  Notice that according to the outlined scheme, it consists of the set of commitments to each digit value v_i plus the whole Borromean signature:

$$RP_v = (C_0, \ldots, C_{15}, \underbrace{e_0, s_{0,0}, \ldots, s_{0,3}, \ldots, s_{15,0}, \ldots, s_{15,3}}_{signature}),$$

  where as usual the indexes of the s-values respectively refer to ring number and position in each ring.

For the sake of clarity we can show a concrete example of the whole procedure. Consider a transaction output of 2,010 BTC (i.e. 201000000_{10} satoshi $\leftrightarrow$ 0023332300101000_4 satoshi).
Then, commit to each digit value of the encoded amount through a Pedersen commitment $C_i = r_i G + v_i 4^i H$. For instance, considering the third digit from the left (which is a 2), it would mean building the following:

$$C_{13} = r_{13}G + 2 \cdot 4^{13}H = r_{13}G + 134217728H.$$

Then arrange the following set of public keys (for the considered digit):

$$\begin{aligned}
C_{13,0} &= C_{13} - 0H \\
C_{13,1} &= C_{13} - 1 \cdot 4^{13}H = C_{13} - 67108864H \\
C_{13,2} &= C_{13} - 2 \cdot 4^{13}H = C_{13} - 134217728H \\
C_{13,3} &= C_{13} - 3 \cdot 4^{13}H = C_{13} - 201326592H
\end{aligned}$$

Eventually provide a Borromean ring signature over this ring, which in the background means to perform a real signature for $C_{13,2} = r_{13}G$.
For what concerns the range proof size, consisting of the shared e-value, 4 s-values per ring and a commitment per ring it is about 2.6 KB per transaction output. Given these premises, it can also be proven that the base-4 representation minimizes the total size of the range proof.
Indeed, given that y is the number of bits-per-digit in each encoding[13] and

[13]Base 2 $\leftrightarrow$ 1 bit per digit; base 4 $\leftrightarrow$ 2 bits per digit; base 8 $\leftrightarrow$ 3 bits per digit etc.

given that we are considering 32-bit range proofs, it can be proven that the total number of commitments plus s-values (the e-value is unique independently of the encoding) is given by $N(y) = \frac{32}{y} \cdot (1 + 2^y)$.

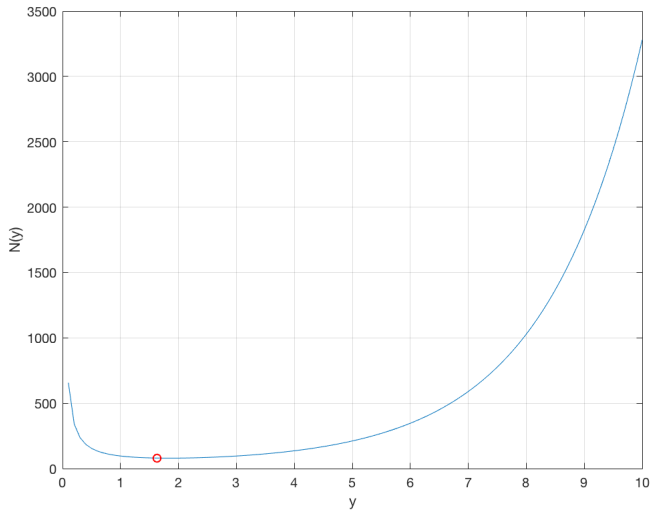

Figure 5.2: Total number of commitments plus s-values

As Figure 5.2 shows, $N(y)$ takes its minimum around $y = 2$, meaning that the minimization occurs when considering the base-4 encoding.

5.4 Sender/receiver communication

What is left out are the details about the communication between sender and receiver for what concerns setting of the blinding factors and transmission of the committed amounts. Indeed, from the point of view of the network range proofs represent proofs that output amounts are valid, but amounts (together with blinding factors) stay secret as they are blinded by Pedersen commitments. However, both amounts and blinding factors need to be known by participants in the transaction. This opens the issue of their safe transfer from sender to receiver.
It turns out that confidential transactions succeed in providing a space-saving solution to embed these secret data in the transaction without adding more data rather than commitments and range proofs. Indeed, *amounts* are embedded in range proofs without taking up more space; *blinding factors* can be retrieved by sender and receiver only as part of a deterministic generation process which enables the transfer to happen non-interactively.
In addition to this, it is even possible to transfer arbitrary user-selected data

without further overload.
Thus, we will see how the ECDH primitive presented in section 4.4 is exploited to serve the purpose this section is all about. First of all we recall that an instance of ECDH is needed to define a shared key between sender and receiver.
A peculiarity of confidential transactions is that each address contains a *scanning public key* for ECDH purpose only. More precisely, the sender ECDH public key is published as part of the output (it coincides with Q_A in the scheme of section 4.4), while each confidential transaction address contains the ECDH public key of the receiver (Q_B in section 4.4). Eventually q_A and q_B play the role of ECDH ephemeral private keys (respectively associated to the sender scanning key and the receiver one). Given this, it is clear that both sender and receiver can compute the same secret key as it was described in section 4.4.
The shared key is then used by both sender and receiver to seed a RFC6979 [24] pseudo-random number generator (prng) to deduce the same blinding factors and random s-values which enter in the Borromean ring signature algorithms. Indeed without entering in the details of the RFC6979 standard, this is used to deterministically and safely generate the nonce k to be used in ECDSA signatures and outside ECDSA can basically work as a cryptographically secure pseudo-random number generator taking as input a seed. Moreover, various rounds of RFC6979 can be run in sequence, each time using the previous output as seed.
Making this process deterministic, provides some sort of advantages. Among these there is certainly the possibility to distinguish where randomness resides and exploit the feature accordingly. For the *sender*, this configures in the possibility to send the committed transaction output amount (and possibly other kinds of arbitrary selected data) over predetermined and forged signature values (we will describe how in a while); for the *receiver*, this means to possibly "rewind" the proof and in turn generate the same blinding factors of the sender, read the transaction output amount and extract further messages the sender could have sent.
We will enter now in all of these details.

- Setting and transmission of the blinding factors:
  the 32-bit range proof of the previous section includes a commitment per digit of the committed output amount, the blinding factors of which should be known by the receiver (their sum being nothing else than the blinding factor of the commitment to the output amount). Indeed, 16 blinding factors (one per ring) should be shared by sender and receiver.
  At this extent the 32-byte shared key derived from the ECDH instance is

used as first seed to a RFC6979 prng, each successive output is used as next seed to deduce all of the blinding factors $r_0, r_1, \ldots, r_{15}$ (though r_{15} is obtained by difference so as to to match the sum of the blinding factors from inputs).

- Generation of forged signatures and nonces:
  as we have described, in the Borromean ring signature algorithm a single s-value per ring is obtained from a known private key, all the others are forged, but indistinguishable from random. In the example of the 32-bits range proof discussed in the previous section, this means that $\frac{3}{4}$ of the s-values are forged and again a RFC6979 prng is used to generate them in such a way that the receiver would be able to easily retrieve all of them by seeding the prng with the same seed the sender has used.
  For what concerns nonces k, one per ring is required to be picked uniformly at random. Observe that k_i, $i = 0, \ldots, r-1$ enters in the computation of the real s-value only, meaning that even for the remaining $\frac{1}{4}$ of the s-values generation of random values is needed (and the same solution is adopted).

- Transmission of committed amount:
  the transaction output amount is embedded in the already provided construction in such a way that it is easily available to the receiver and does not occupy more space.
  Given that the published range proof has to be indistinguishable from random for anybody but the participants in the transaction, after locating where randomness resides (s-values that can be retrieved via RFC6979 procedure are the forged ones) the idea is to replace this randomness in the range proof with an encrypted message that the receiver only can decode. In particular, the amount is embedded into one of the forged signatures (usually the last one, unless it is the real one) by XORing[14] it with the forged s-value for that signature. Indeed, the XOR does not affect the pseudo-randomness of the original s-value. Moreover, it could be useful to stress once more the fact that only forged signatures are tampered with in this way; the real s-value is necessary to close the ring and validate the signature, so it mustn't be manipulated.

[14] The XOR is the *bitwise addition* without carry. Among its properties:

- $\mathrm{A} \oplus 0 = \mathrm{A}$;
- $\mathrm{A} \oplus \mathrm{A} = 0$;
- $(\mathrm{A} \oplus \mathrm{B}) \oplus \mathrm{C} = \mathrm{A} \oplus (\mathrm{B} \oplus \mathrm{C})$;
- $(\mathrm{B} \oplus \mathrm{A}) \oplus \mathrm{A} = \mathrm{B} \oplus 0 = \mathrm{B}$.

Referring to forged signatures only (and consequently forged s-values only), let s_{common} be the s-value deterministically generated via the RFC6979 procedure and $s_{published}$ be the s-value actually published by the sender in the range proof. What this means is that $s_{published}$ is given by

$$s_{published} = s_{common} \oplus amount,$$

and implies that:

$$s_{common} \oplus s_{published} = s_{common} \oplus s_{common} \oplus amount = amount.$$

Thus, the receiver by looking at the last s-value (unless it is the case that the last s-value is not forged) and computing $s_{common} \oplus s_{published}$ sees the amount if the last signature has been tampered with it, 0 if not.

- Transmission of other user-defined data:
  the amount takes 32 bytes only, all the rest less the space corresponding to real s-values (which is however almost 80% of the size of the range proof) can be exploited in a similar way for the transmission of arbitrary data. These data rather than the amount would be XORed in with the pseudo-random s-values.
  Thus, at least one (if no arbitrary data is sent) of the forged s-values published as part of the range proof is the result of a XORing.

- Proof's "rewind" by the receiver:
  given the shared key, the receiver can run the same RFC6979 procedure of the sender and in turn deduce same blinding factors and s-values. In turn he can perform the XOR operation as it was described before.

5.5 Benefits and downsides

Confidential transactions can be possibly constructed without adding any additional cryptographic assumptions with respect to the ones underlying the main protocol: this means that they just require the hardness of the ECDLP. This is quite relevant as it is not that common to add privacy features without relying on harder assumptions.
However, what effectively makes confidential transactions (at least in the form presented in this work) not ready yet for integration in the main protocol is the excessive overload on transaction size. The presented solution (which builds range proofs through Borromean ring signatures) requires a proof which is about 2.6 KB per transaction output (with 32 bits of precision), which means a total size of about 5.4 KB (5 KB of which deriving from

the range proof) for a typical transaction with two outputs (the transaction size even grows linearly with the number of outputs); too much with respect to a standard Bitcoin transaction.

Moreover, confidential transactions are actually implemented in the Elements sidechain.[15] De facto they introduce a different transaction format with respect to the one presented in Chapter 2. Each output includes Pedersen commitment to the output amount, associated range proof, ECDH public key of the sender and locking script. Despite not having presented the details, in section 5.4 it was mentioned that they even introduce a new type of address (longer than a standard one) called *confidential transaction address* which includes the ECDH public key of the receiver. The locking script includes the confidential transaction address.
A very powerful comprehensive schema can be seen in Figure 5.3 and is taken from [11]. It represents a confidential transaction with two inputs and two outputs.

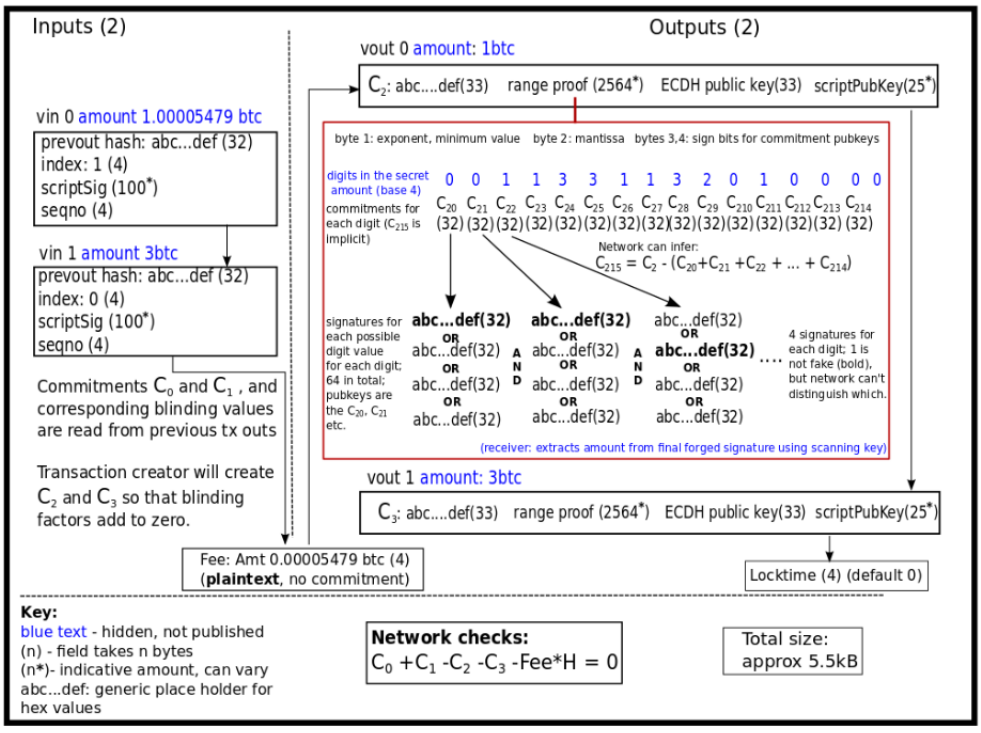

Figure 5.3: Confidential transaction format

Source: [11]

[15] See `https://github.com/ElementsProject/elements`.

Chapter 6

Conclusions

This thesis has tried to motivate the importance for Bitcoin to adopt technologies enhancing privacy in the incoming years. This would not merely improve the privacy of people transacting (which is indeed fundamental), but even strengthen its ability to serve as money. Indeed, though it cannot be recognized as a good unit of account, Bitcoin is a good store of value (it is durable, it can be reliably saved, stored with low costs and easily retrieved) and an excellent medium of exchange (it is easily portable, divisible, swappable, resistant to counterfeiting). Its greatest lack is that it is not that fungible and it is the case that fungibility is strictly linked to privacy.
At the same time, this work should have outlined the reasons why no privacy-based solutions have been soft-forked yet in Bitcoin at the time of writing; this is not certainly for a lack of proposals, quite the opposite. Developers have worked in this direction since long time, but cryptographic, privacy-based solutions are costly and require commitments, thus opening other issues.
Moreover, through confidential transactions [14] we have explored some nice features of homomorphic encryption applied to commitment schemes, the interesting field of Zero-Knowledge proofs, a fancy variant of digital signature scheme and a clever solution for the communication of transaction amount, blinding factors and other user-selected metadata between participants in the transaction.
Confidential transactions basically hides each output amount through a Pedersen commitment to the amount and add a range proof ensuring that the amount does not overflow. The solution we have described builds each range proof through a Borromean ring signature. Among the strengths of confidential transactions, the fact that these can be possibly constructed without new cryptographic assumptions with respect to the main protocol, but relying on the hardness of the ECDLP (differently from some alternative schemes like Zcash's ones) and the substantial savings in terms of size and verification

time with respect to previous solutions (which have made them sources of inspiration for privacy-based alt-coins like Monero, that has effectively implemented confidential transactions through ring signatures, RingCT [19]). On the other hand, the solution suffers from the size of the range proof attached to each transaction output (and thus of the entire transaction) being too large.
These weaknesses have prevented confidential transactions to be soft-forked in Bitcoin up to now.
More recently, however, a new and more efficient solution to range proof construction [8] has been proposed. Its name is Bulletproofs and it is likely worth studying: it could be the solution being effectively soft-forked in the future (it even naturally marries some older proposals) and effectively bringing consistent privacy in the protocol. Indeed, Bulletproofs is still well-suited for constructing efficient range proofs on committed value, but it also adds various optimizations.
At first, it provides aggregation of range proofs: it would be possible to prove that m commitments lie in a given range by just providing additional $O(\log m)$ group elements with respect to a single proof, making it growing logarithmically with the number of transaction outputs. This would be already useful for confidential transactions by themselves as standard Bitcoin transactions have generally at least two outputs and it would even make it efficiently combine with Coinjoin [13]. Additionally it could simultaneously double the range proof precision at marginal additional cost.
Then, it would allow batched verification of multiple Bulletproofs.
All of these enhancements with a total transaction size not so bigger than a standard transaction according to [8, 16].
Moreover, confidential transactions are even beneficial for a newborn and promising cryptosystem called Mimblewimble [12, 22]. Mimblewimble is still a transaction output based system (thus a Bitcoin-like blockchain system) which however implements confidential transactions from the beginning. At the time of writing it is already being built through Bulletproofs, thus inheriting its benefits. Moreover, Mimblewimble removes the need for the unlocking script because it allows to prove a transaction to commit to a Pedersen commitment to 0 just signing the transaction through the difference of the commitments to outputs and inputs. Other than this, it can benefit from transaction aggregation and enables the construction of a simplified blockchain where spent transactions can be pruned, thus improving scalability.

Appendix A

Abstract algebra fundamentals

This appendix is aimed at providing some fundamental notions of algebra of sets and number theory at the basis of the considered public-key cryptosystem. Definitions are mainly taken from [20] and adapted when needed.

A.1 Groups

We start from the definition of a *group*.

Definition A.1.1. *A group is a set of elements $\mathbb{G}$ together with an operation $\circ$ which combines two elements of $\mathbb{G}$. A group satisfies the following properties:*

- *The group operation $\circ$ is closed:* $\forall a, b \in \mathbb{G} \rightarrow a \circ b \in \mathbb{G}$.
- *The group operation $\circ$ is associative:* $\forall a, b, c \in \mathbb{G} \rightarrow (a \circ b) \circ c = a \circ (b \circ c)$.
- *Identity:* $\exists e \in \mathbb{G} \mid \forall a \in \mathbb{G},\ e \circ a = a \circ e = a$.
- *Invertibility:* $\forall a \in \mathbb{G},\ \exists b \in \mathbb{G} \mid a \circ b = b \circ a = e$. *This element is called inverse of a and it is commonly denoted either as a^{-1} or $-a$, depending on the notation (multiplicative or additive).*
- *A group $\mathbb{G}$ is abelian (or commutative) if, furthermore,* $\forall a, b \in \mathbb{G} \rightarrow a \circ b = b \circ a$.

Depending on whether we consider additive or multiplicative notation, the operation $\circ$ stands respectively for addition or multiplication.

Remark A.1.1. *The group operation $\circ$ is called group law of $\mathbb{G}$.*

Remark A.1.2. *The number of elements in a group $\mathbb{G}$ is called group order (or cardinality). We denote it by $|\mathbb{G}|$.*

Example A.1.1. *$(\mathbb{Z}, +)$ is a group. Particularly, it forms an abelian group where $e = 0$ is the identity element, $b = -a$ is the inverse of an element $a \in \mathbb{Z}$.*

Example A.1.2. *$(\mathbb{Z}\backslash\{0\}, \cdot)$ is **not** a group. Particularly, $\nexists b = a^{-1}$ for an element $a \in \mathbb{Z}$ with the exception of the elements -1 and 1.*

Example A.1.3. *$(\mathbb{Z}_m, +)$, where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ and the operation is the addition modulo m, form a group (of order m) with the identity element $e = 0$. Every element a has an inverse $b = -a$ such that $a + (-a) = 0$ mod m.*

Remark A.1.3. *This last example points out a straightforward, yet important, fact. By definition, the inverse must belong to the group $\rightarrow b = m - a$ is the inverse of any group element a.*

Remark A.1.4. *Observe that $(\mathbb{Z}_m, \cdot)$ is **not** a group. Most elements a do not have an inverse such that $aa^{-1} = 1$ mod m.*

Actually, in cryptography it turns out that the groups playing a significant role are those with a finite number of elements. We briefly focus now on one of them, $(\mathbb{Z}_m^*, \cdot)$, the multiplicative group of $\mathbb{Z}_m$.
Let's start with some definitions.

Definition A.1.2. *Given $x, y \in \mathbb{Z}$, $gcd(x, y)$ is the greatest common divisor of x, y.*

Remark A.1.5. *If $gcd(x, y) = 1$, we say that x, y are relatively prime.*

Lemma A.1.1. *$\forall x, y \in \mathbb{Z}$, $\exists a, b \in \mathbb{Z}$ s.t. $ax + by = gcd(x, y)$. a,b, can be efficiently found through the extended Euclidean algorithm (see [20] for details).*

Given the definition of inverse element of a group seen above, we introduce the following:

Lemma A.1.2. *x in $(\mathbb{Z}_m, \cdot)$ has an inverse $\longleftrightarrow$ $gcd(x, m) = 1$.*

Proof. ($\longrightarrow$) Suppose by contradiction that $gcd(x, m) > 1$. Then, $\forall a$: $gcd(ax, m) > 1 \rightarrow ax \neq 1$ in $\mathbb{Z}_m$, which (according to Definition A.1.1) contradicts the hypothesis.
($\longleftarrow$) $\exists a, b$: $ax + \cancel{bm} = 1$ ($bm = 0$ mod m, thus $bm = 0$ in $\mathbb{Z}_m$) $\rightarrow ax = 1$ in $\mathbb{Z}_m \rightarrow x$ is invertible in $\mathbb{Z}_m$, the inverse being $x^{-1} = a$. □

Proposition A.1.1. $(\mathbb{Z}_m^*, \cdot) = \{x \in \mathbb{Z}_m : gcd(x, m) = 1\}$ *forms an abelian group. The identity element is* $e = 1$.

Proof. The proof is straightforward and comes from the verification of the group properties described above. □

Remark A.1.6. *In particular, if* m *is prime, then* $\mathbb{Z}_m^* = \mathbb{Z}_m \backslash \{0\}$.

A.1.1 Cyclic groups

Next step before coming to field structures is to introduce the notion of *cyclic group*, necessary in turn to introduce the Generalized Discrete Logarithm Problem, which is at the basis of ECC.
Let's first start with some preliminary definitions.

Definition A.1.3. *A group* $(\mathbb{G}, \circ)$ *is finite if it has a finite number of elements.*

Definition A.1.4. *The order* $ord(x)$ *of an element* x *of a group* $(\mathbb{G}, \circ)$ *is the smallest positive integer* k *such that* $x^k = 1$, $e = 1$ *being the identity element of* $\mathbb{G}$.

Example A.1.4. *In* $(\mathbb{Z}_7^*, \cdot)$, $ord(x = 2) = 3$. *Indeed* $2^1 = 2$ *mod* 7, $2^2 = 4$ *mod* 7, $2^3 = 1$ *mod* 7.
It is even interesting to see that by keeping on computing powers of $x = 2$, *those will keep on running through the above sequence. Indeed,* $2^4 = 2$ *mod* 7, $2^5 = 4$ *mod* 7, $2^6 = 1$ *mod* 7.

Based on this,

Definition A.1.5. *A group* $(\mathbb{G}, \circ)$ *which contains an element* x *with maximum order* $ord(x) = |\mathbb{G}|$ *is said to be cyclic. Elements with maximum order are called generators and are denoted by* g.

Remark A.1.7. *The reason for which* g *is called generator is that it generates (or spans) the entire group (all the group elements can be recovered by raising* g *to the powers* $1, \ldots, |\mathbb{G}|$*). In the previous example,* $g = 3$ *is a generator.*

Remark A.1.8. *It is also clear that not every element is a generator of the cyclic group (*$x = 2$ *being an example).*

Then we introduce three more theorems defining fundamental properties of cyclic groups. The first one is the following, by Euler.

Theorem A.1.1. *For every prime p, $(\mathbb{Z}_p^*, \cdot)$ is an abelian finite cyclic group.*

Theorem A.1.2. *Let $\mathbb{G}$ be a finite cyclic group. Then $\forall x \in \mathbb{G}$, $ord(x)$ divides $|\mathbb{G}|$.*

Consequently, in a cyclic group there exist only element orders dividing exactly the cardinality of the group.

Remark A.1.9. *In the already analyzed $(\mathbb{Z}_7^*, \cdot)$, the only possible element orders are $ord(x) = 1, 2, 3$ being $|\mathbb{G}| = 6$.*

Theorem A.1.3. *Let $\mathbb{G}$ be a finite cyclic group. Then, if $|\mathbb{G}|$ is prime, all elements $x \neq 1 \in \mathbb{G}$ are generators.*

Proof. By Theorem A.1.2, being the group cardinality a prime, the only possible element orders are $ord(x) = 1$ or $ord(x) = |\mathbb{G}|$. As $ord(x) = 1 \longleftrightarrow x = 1$, then any $x = g \neq 1 \in \mathbb{G}$ is a generator. □

The last needed definition concerning group is the one of *subgroups*, basically non-empty subsets $\mathbb{H}$ of a (cyclic) group $\mathbb{G}$ being themselves groups.

Theorem A.1.4. *Let $(\mathbb{G}, \circ)$ be a cyclic group. Then every element $x \in \mathbb{G}$ with $ord(x) = s$ is the generator of a cyclic subgroup with s elements.*

A.2 Fields

We can eventually introduce the definition of *field*.

Definition A.2.1. *A field $K \neq 0$ is a set of elements with the following properties:*

- *$(K, +)$ is an abelian (additive) group, with identity element $e = 0$.*
- *$(K \backslash \{0\}, \cdot)$ is an abelian (multiplicative) group, with identity element $e = 1$.*
- *The distributivity law holds, i.e., $\forall\, a, b, c \in K$: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.*

Example A.2.1. *The set $(\mathbb{R}, +, \cdot)$ of real numbers is a field with identity element $e = 0$ for the additive group and identity element $e = 1$ for the multiplicative group. $\forall a\ \exists b = -a$ additive inverse, $\forall a \neq 0\ \exists b = \frac{1}{a}$ multiplicative inverse.*

Example A.2.2. *For every prime p, the set $(\mathbb{Z}_p, +, \cdot)$ is a field.*

Remark A.2.1. *Example A.2.2 shouldn't be much of a surprise, given the premises. Indeed, we have already seen* $(\mathbb{Z}_p, +)$ *is an additive group;* $(\mathbb{Z}_p \backslash \{0\}, \cdot)$ *in principle (p generic) wouldn't be a multiplicative group, but p prime implies* $(\mathbb{Z}_p \backslash \{0\}, \cdot) \equiv (\mathbb{Z}_p^*, \cdot)$*, which we have already seen being a multiplicative group.*

Remark A.2.2. *Example A.2.2 is also the most common representative of prime finite field, which cryptography is most concerned on.*

A.2.1 Finite fields

Definition A.2.2. *A finite field is a field with a finite number of elements (also called Galois field).*

Theorem A.2.1. *A finite field of order q, denoted as* $\mathbb{F}_q$*, only exists if* $q = p^k$*, p prime number, k positive integer.*

In particular, prime finite fields (Example A.2.2 being the most representative example) takes a fundamental role in dealing with the DLP, argument of the next section.

A.3 Discrete Logarithm Problem

We conclude the appendix with the Discrete Logarithm Problem, in its various formulations.
Consider the prime finite field $\mathbb{F}_p = (\mathbb{Z}_p, +, \cdot)$, p prime.
It turns out that the DLP can be directly explained through cyclic groups (from which our previous concern in their description). We present it in its multiplicative form (thus DLP over $(\mathbb{Z}_p^*, \cdot)$), the additive one being simple to recover from.

Definition A.3.1. *Given the finite cyclic group* $(\mathbb{Z}_p^*, \cdot)$*, a generator* $g \in \mathbb{Z}_p^*$ *and another element* $y \in \mathbb{Z}_p^*$*, the DLP is the problem of determining the integer* $1 \leq k \leq p-1$ *such that* $g^k = y \mod p$*.*

Remark A.3.1. *Being g a generator of a finite cyclic group and y another group element, k must necessarily exist due to Remark A.1.7.*

Remark A.3.2. $k = \log_g y \mod p$*, from which its name.*

Remark A.3.3. *The obvious transposition to additive groups: determining k such that* $kg = y$*.*

Here it comes, instead, the reason of the introduction of subgroups.

Remark A.3.4. *It is often desirable to have the DLP in groups where* $|\mathbb{G}|$ *is prime to prevent the so called Pohlig-Hellman attack*[1]*; being* $|\mathbb{Z}_p^*| = p-1$ *(not prime), subgroups of* $\mathbb{Z}_p^*$ *with* $|\cdot| = n$*,* $n < p$*,* n *prime are usually exploited.*

A.3.1 Generalized Discrete Logarithm Problem

It is possible to generalize and present the DLP over an arbitrary cyclic group, which is at the basis of ECC.

Definition A.3.2. *Given a finite cyclic group* $(\mathbb{G}, \circ)$ *such that* $|\mathbb{G}| = n$*, a generator* $g \in \mathbb{G}$ *and* $y \in \mathbb{G}$*, the GDLP is the problem of determining the integer* k*,* $1 \leq k \leq n$ *such that* $y = \underbrace{g \circ g \circ \cdots \circ g}_{k \quad times} = g^k$*.*

Remark A.3.5. *The same considerations presented before hold here.*

[1]For security considerations we refer to [20]; the only thing we point out is that the cited one is not the only possible attack (*brute-force search*, *Baby-step giant-step*, *Pollard* ρ *method* being other possibilities) and that the best known algorithm runs in time $O(|\mathbb{G}|^{\frac{1}{2}})$.

Bibliography

[1] Bitcoin developer guide. https://bitcoin.org/en/developer-reference.

[2] Sec 1: Elliptic curve cryptography. http://www.secg.org/sec1-v2.pdf, 2009.

[3] Abe, M., Ohkubo, M., and Suzuki, K. 1-out-of-n signatures from a variety of keys. https://www.iacr.org/archive/asiacrypt2002/25010414/25010414.ps, 2002.

[4] Ammous, S. *The Bitcoin Standard*. Wiley, 2018.

[5] Andreev, O. Blockchains in a quantum future - protecting against post-quantum attacks on cryptography. https://blog.chain.com/preparing-for-a-quantum-future-45535b316314, 2017.

[6] Antonopoulos, A. M. *Mastering Bitcoin*. O'Reilly, 2017.

[7] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timn, J., and Wuille, P. Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf, 2014.

[8] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. Bulletproofs: Short proofs for confidential transactions and more. https://eprint.iacr.org/2017/1066.pdf, 2017.

[9] Franco, P. *Understanding Bitcoin*. Wiley, 2014.

[10] Gibson, A. From zero (knowledge) to bulletproofs. https://github.com/AdamISZ/from0k2bp/blob/master/from0k2bp.pdf.

[11] Gibson, A. An investigation into confidential transactions. https://github.com/AdamISZ/ConfidentialTransactionsDoc/blob/master/essayonCT.pdf.

[12] Jedusor, T. E. Mimblewimble. https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt, 2016.

[13] Maxwell, G. Coinjoin: Bitcoin privacy for the real world. https://bitcointalk.org/index.php?topic=279249.0, 2013.

[14] Maxwell, G. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt, 2015.

[15] Maxwell, G. Improve transaction privacy / fungibility in bitcoin core and the bitcoin system [meta tracking issues]. https://github.com/bitcoin/bitcoin/issues/6568, 2015.

[16] Maxwell, G. [bitcoin-dev] updates on confidential transactions efficiency. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-November/015283.html, 2017.

[17] Maxwell, G., and Poelstra, A. Borromean ring signatures. https://github.com/ElementsProject/borromean-signatures-writeup.

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

[19] Noether, S., Mackenzie, A., and the Monero Research Lab. Ring confidential transactions. *ledgerjournal.org* (2015). https://eprint.iacr.org/2015/1098.

[20] Paar, C., and Pelzl, J. *Understanding Cryptography.* Springer, 2009.

[21] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf, 1991.

[22] Poelstra, A. Mimblewimble. https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[23] Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2016.

[24] Pornin, T. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). https://rfc-editor.org/rfc/rfc6979.txt, Aug. 2013.

[25] Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret. https://www.iacr.org/archive/asiacrypt2001/22480554.pdf, 2001.

[26] Rosenberg, M. Confidential transactions from basic principles. http://cryptoservices.github.io/cryptography/2017/07/21/Sigs.html, 2017.

[27] Song, J. Programming bitcoin. https://github.com/jimmysong/programmingbitcoin.

[28] Wuille, P. Schnorr's bip. https://github.com/sipa/bips/blob/bip-schnorr/bip-schnorr.mediawiki, 2018.