Bitcoin and Blockchain Security

Ghassan Karame
Elli Androulaki

ARTECH
HOUSE
BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library.

ISBN 13: 978-1-63081-013-9

685 Canton Street
Norwood, MA 02062

Contents

Preface **xi**

Acknowledgments **xiii**

Chapter 1 Introduction **1**
1.1 Book Structure 5
1.1.1 Chapter 2 5
1.1.2 Chapter 3 6
1.1.3 Chapter 4 6
1.1.4 Chapter 5 7
1.1.5 Chapter 6 7
1.1.6 Chapter 7 8
1.1.7 Chapter 8 8
1.1.8 Chapter 9 8
1.1.9 Chapter 10 9

Chapter 2 Background on Digital Payments **11**
2.1 Payment Systems Architecture 11
2.2 Security and Privacy in Payments 13
2.2.1 Security 14
2.2.2 Privacy 15
2.2.3 Combining Security and Privacy 16
2.3 Security in Payment Systems prior to Bitcoin 17
2.3.1 Common Payment System Characteristics 17

2.3.2 Privacy-preserving Payments Due to the Research Community 20
2.3.3 Deployed Payment Systems 26
2.4 Summary 29

Chapter 3 Bitcoin Protocol Specification **33**
3.1 Overview of Bitcoin 33
3.2 Building Blocks and Cryptographic Tools 35
3.2.1 Cryptographic Hash Functions 35
3.2.2 Merkle Trees 35
3.2.3 ECDSA 36
3.3 Bitcoin Data Types 36
3.3.1 Scripts 37
3.3.2 Addresses 38
3.3.3 Transactions 38
3.3.4 Blocks 43
3.4 Bitcoin Architecture 47
3.4.1 Node Types 48
3.4.2 Peer-to-Peer Overlay Network 49
3.5 Scalability Measures in Bitcoin 53
3.5.1 Request Management System 53
3.5.2 Static Time-outs 55
3.5.3 Recording Transaction Advertisements 55
3.5.4 Internal Reputation Management System 56

Chapter 4 Security of Transactions in Bitcoin **59**
4.1 Security of Confirmed Transactions 59
4.1.1 Transaction Verification 60
4.1.2 Eclipse Attacks in Bitcoin 61
4.1.3 Denying the Delivery of Transactions 63
4.1.4 Transaction Confirmation 65
4.2 Security of Zero-Confirmation Transactions 69
4.2.1 (In-)Security of Zero-Confirmation Transactions 69
4.2.2 Possible Countermeasures 74
4.3 Bitcoin Forks 79
4.3.1 Exploiting Forks to Double-Spend 79
4.3.2 Fork Resolution 80

Chapter 5 Privacy in Bitcoin **85**

5.1 User Privacy in Bitcoin 86
5.1.1 Protocol-Based Privacy Quantification in Bitcoin 87
5.1.2 Exploiting Existing Bitcoin Client Implementations 89
5.1.3 Summing Up: Behavior-Based Analysis 90
5.1.4 Coin Tainting 91
5.1.5 Risks of Holding Tainted Bitcoins 93
5.2 Network-Layer Attacks 93
5.2.1 Refresher on Bitcoin P2P Network Setup 94
5.2.2 Privacy Leakage over the Bitcoin Network 94
5.3 Enhancing Privacy in Bitcoin 97
5.3.1 Mixing Services 98
5.3.2 CoinJoin 99
5.3.3 Privacy-Preserving Bitcoin Protocol Enhancements 100
5.3.4 Extending ZeroCoin: EZC and ZeroCash 107
5.4 Summary 120

Chapter 6 Security and Privacy of Lightweight Clients **125**
6.1 Simple Payment Verification 125
6.1.1 Overview 125
6.1.2 Specification of SPV Mode 126
6.1.3 Security Provisions of SPV mode 127
6.2 Privacy Provisions of Lightweight Clients 128
6.2.1 Bloom Filters 128
6.2.2 Privacy Provisions 129
6.2.3 Leakage Due to the Network Layer 130
6.2.4 Leakage Due to the Insertion of Both Public Keys and Addresses in the Bloom filter 130
6.2.5 Leakage under a Single Bloom Filter 131
6.2.6 Leakage under Multiple Bloom Filters 134
6.2.7 Summary 138
6.2.8 Countermeasure of Gervais et al. 139

Chapter 7 Bitcoin's Ecosystem **143**
7.1 Payment Processors 144
7.2 Bitcoin Exchanges 146
7.3 Bitcoin Wallets 146
7.3.1 Securing Bitcoin Wallets 148
7.4 Mining Pools 151
7.4.1 Impact of Mining Pools on De-centralization 152

7.5 Betting Platforms 154
7.6 Protocol Maintenance and Modifications 155
7.6.1 Bitcoin Improvement Proposals 156
7.6.2 The Need for Transparent Decision Making 156
7.7 Concluding Remarks 157

Chapter 8 Applications and Extensions of Bitcoin **163**
8.1 Extensions of Bitcoin 163
8.1.1 Litecoin 164
8.1.2 Dogecoin 164
8.1.3 Namecoin 165
8.1.4 Digital Assets 165
8.2 Applications of Bitcoin's Blockchain 166
8.2.1 Robust Decentralized Storage 166
8.2.2 Permacoin 169
8.2.3 Decentralized Identity Management 171
8.2.4 Time-Dependent Source of Randomness 171
8.2.5 Smart Contracts 172
8.3 Concluding Remarks 175

Chapter 9 Blockchain Beyond Bitcoin **179**
9.1 Sidechains 180
9.2 Ethereum 181
9.2.1 Accounts 182
9.2.2 Transactions and Messages 182
9.2.3 State and Transaction Execution 183
9.2.4 Blocks 183
9.2.5 Mining and Blockchain 184
9.3 Open Blockchain 185
9.3.1 Membership Services 186
9.3.2 Transactions Life-cycle 190
9.3.3 Possible Extensions 192
9.4 Ripple 193
9.4.1 Overview of Ripple 194
9.5 Comparison between Bitcoin, Ripple, Ethereum, and Open Blockchain 196
9.5.1 Security 197
9.5.2 Consensus Speed 198
9.5.3 Privacy and Anonymity 198

9.5.4 Clients, Protocol Update, and Maintenance 199
9.5.5 Decentralized Deployment 199

Chapter 10 Concluding Remarks **205**

About the Authors **213**

Index **215**

Preface

We were first introduced to Bitcoin in October 2011. At that time, both Elli and I were conducting our post-doctoral research at ETH Zurich. We were reading several media articles mentioning Bitcoin, and were rather curious about the underlying system. A specific article caught our attention at that time: *Bitcoins were accepted as a form of payment in a fast-food restaurant in New York*. We were not surprised by the fact that people were using Bitcoin for real payments; it is true that we did not really believe in Bitcoin at that time. We believed that Bitcoin was an interesting protocol allowing computer geeks to make money by running a program on their PC. Our surprise was mainly that Bitcoin—in which a transaction takes almost an hour to be confirmed—was used to handle fast payments! We decided to immediately write a paper to warn the community from such usage of Bitcoin; in our paper, we showed analytically and experimentally that double-spending in Bitcoin can be easily realized in the network on unconfirmed transactions. At that time, we bought 10 Bitcoins with 5 Swiss Francs and I remember thinking: "These Bitcoins are really expensive" (I wish I knew better.) Our paper was published at ACM CCS 2012, which is one of the most prestigious computer security conferences in the world. We additionally proposed some countermeasure to allow fast payments with minimal risk of double-spending; our countermeasure was eventually integrated in Bitcoin XT.

From that point on, we delved into researching Bitcoin. This resulted in a number of papers that appeared at top security and privacy conferences; the first few lines in our introductions would evolve from "Bitcoin is receiving considerable attention in the community" to something that turned out to be a big surprise to us as well: "Bitcoin has received more adoption than any other digital currency proposed to date."

Five years after our first research paper on Bitcoin (during which we published eight research papers on Bitcoin at top security venues), we decided that it was time to share our Bitcoin experience, and the various lessons that we learned with a broader audience.

This book is mostly intended for computer scientist/engineers and security experts. If you are interested in Bitcoin, and you do have general computer science knowledge, this book will teach you all that you need to know about the security and privacy provisions of Bitcoin.

Dr. Ghassan Karame

Chapter 1

Introduction

With the publication of the Bitcoin white paper in 2008, and the subsequent delivery of a first prototype implementation of Bitcoin 2 months later, the individual or group behind the alias "Satoshi Nakamoto" was able to forge a new class of decentralized currency. Unlike previous electronic cash proposals, this proposal was rather straightforward, explained in a concise white paper comprising 8.5 single-column pages, and relying on basic cryptographic constructs, such as hash functions and digital signatures. The release of the proof of concept implementation of Bitcoin shortly after the dissemination of the white paper was extremely timely and important for the subsequent growth of Bitcoin. The working implementation confirmed that, unlike previous proposals, the system is clearly feasible/workable, and scales to a large number of nodes. Open-sourcing the implementation was also an excellent call for developers to maintain and support the growth of the system.

The design of Bitcoin offered the world a promise for a low-cost decentralized and anonymous currency. The core idea of Bitcoin is simple. The system allows two or more parties to exchange financial transactions without passing through intermediaries (such as banks or payment processors). These transactions are validated collectively in a peer-to-peer network by all users. This not only eliminates the need for centralized control (e.g., by banks), but also reduces the cost of making transactions (at the national and international levels). The premise of ease of use and anonymity were also appealing features of the original design; Bitcoin does not require users to register their identity/credentials nor does it require them to fill out endless forms in order to set up an account. More importantly, Bitcoin users could operate using pseudonyms—without ever revealing their true identity.

Bitcoin's design relies on a clever and well-incentivized cooperation between users in the network. Namely, peers in the network need to receive and validate

all broadcasted transactions—regardless of their respective geographical location. Peers confirm transactions in blocks, by solving a computational puzzle. The puzzle's difficulty is dynamically adjusted based on the computing power in the network, and those peers who succeed in solving the puzzle (and therefore confirm transactions) are financially rewarded. Such reliance on computational puzzles is an effective mechanism to provide a decentralized time-stamping service in the network, and an effective deterrent of Sybil attacks, whereby users create several fake identities in the hope of increasing their advantage in the open network. Namely, the vote or impact that these users exhibit in the network does not depend on the number of their accounts, but is tightly coupled with their available computing power.

This clever design was enough to attract enough traction and participation in Bitcoin across the community.

- Open-sourcing Bitcoin's code solicits the participation of skilled developers who are interested in attaining immediate impact in the community. Their contribution to the Bitcoin code will be reflected in official Bitcoin client releases, which will impact the experience of all Bitcoin users.
- Users were asked to collaboratively contribute in confirming financial transactions; besides involving active user participation in regulating the Bitcoin ecosystem, several users saw in Bitcoin a novel way to invest their computing power and collect immediate financial returns.
- Bitcoin drew a considerable number of consumers who were seeking refuge in the anonymity prospects of the emerging digital currency. This probably explains why Bitcoin's first adopters were believed to be involved in illegal activities and controversial businesses.

These facts ensured a rapid growth of the Bitcoin community in spite of the suspicious disappearance of Bitcoin's founder Satoshi Nakamoto shortly after its release. A number of reports claim that this disappearance was an outcome of the increasing adoption of the system. However, until the time of writing, there are no facts that substantiate these claims.

The rapid growth of the system was however only skeptically received by the financial sector and by the research community. Financial market makers were skeptical about the sustainability of Bitcoin, given the absence of regulations and legislations. As well, researchers criticized the lack of governance in Bitcoin, the underlying economic model, and the security and privacy provisions of the system. The latter point received considerable attention in various academic computer science communities; the literature features a considerable number of reported

attacks, such as double-spending attacks, Eclipse attacks, selfish mining attacks, as well as thorough analyses criticizing the lack of privacy provisions in the system.

Nevertheless, in spite of the ongoing research criticism, and the considerable number of reported attacks on the system, Bitcoin grew to witness a wider adoption and attention than any other digital currency proposed to date. At the time of writing, Bitcoin holds the largest market share among all existing digital currencies, with a market cap of a few billion USD. There are also numerous businesses, exchange platforms, and banks that are currently built around the Bitcoin ecosystem.

One of the (many) reasons that led to the sustainability of the Bitcoin system was the ability of the developers to assimilate research results from the security community and integrate them swiftly within the development of released client implementations. This strategy has probably saved the Bitcoin community from a wide range of attacks and threats that would have definitely crippled Bitcoin's growth. This also led to an implicit partnership between various researchers working on analyzing and securing Bitcoin and the Bitcoin development community. The immediate outcome is that several of the countermeasures proposed by the research community have been effectively integrated in official Bitcoin client releases.

Nevertheless, several security challenges remain ahead for Bitcoin, and there seems to be a sharp disagreement in the community and among core Bitcoin developers on the necessary strategies to sustain the growth of the system. These debates were mostly fueled by discussions on expanding Bitcoin's block sizes. More specifically, a subset of the core developers were in favor of increasing the block size beyond the default cap of 1 MB in order to better cope with the growth of the network, while the rest of developers opposed such a move in the fear of changing/worsening the current network dynamics. This large debate resulted in the exit of developers who were favoring the increase of the maximum block size—a move that many see as the start of the decline of the emerging currency.

In this book, we are not concerned with contributing to the debate on the best strategies to sustain the growth of the system, nor do we plan to take part in favoring any of the existing Bitcoin forks (Bitcoin core, Bitcoin classic, Bitcoin XT), nor do we aim at suggesting/motivating any particular scalability changes to the core Bitcoin system. We definitely do not wish to contribute to the speculations about the future of the currency. Our view (which several other researchers in the community also share) is that the Bitcoin experiment has clearly succeeded. We base this view on the fact that no other proposal for digital currency—besides Bitcoin—has withstood the test of time; Bitcoin has sustained more than 9 years of operation. Namely, no other digital currency proposal—besides Bitcoin—has witnessed such a massive adoption by users/vendors/businesses.

This massive adoption of Bitcoin has truly fueled innovation, and there are currently more than 500 alternate blockchains—most of which are simple variants of Bitcoin. Bitcoin unveiled a key-enabling technology and a hidden potential within the system, the *blockchain*. Indeed, the blockchain allows transactions, and any other data, to be securely stored and verified without the need of any centralized authority. Note that the community has been in search of a scalable distributed consensus protocol for a considerable amount of time.

In this book, we overview, detail, and analyze the security and privacy provisions of Bitcoin and its underlying blockchain—effectively capturing 8 years of thorough research on these subjects. Our contributions go beyond the mere analysis of reported vulnerabilities of Bitcoin; namely, we describe and evaluate a number of countermeasures to deter threats on the system—some of which have already been incorporated in the system. Recall that Bitcoin has been forked multiple times in order to fine-tune the consensus (i.e., the block generation time and the hash function), and the network parameters (e.g., the size of blocks). For instance, Litecoin and Dogecoin—Bitcoin's most prominent forks—reduce the block generation time from 10 to 2.5 and 1 minute, respectively. As such, the results reported in this book are not only restricted to Bitcoin, but apply equally to a number of altcoins that are basically clones/forks of the Bitcoin source code. As far as we are aware, this book emerges as the most comprehensive and detailed analysis of the security and privacy provisions of Bitcoin and of its related clones/variants.

This book takes a holistic approach in covering the security and privacy throughout the entire life cycle of coin expenditure in the system—effectively covering the security of transaction confirmation in the system, the fairness of the mining process, the privacy of users, the security of Bitcoin wallets, network attacks, the security and privacy of lightweight clients, among others. More importantly, the book aims to answer the following important questions:

- What are the actual assumptions governing the security of Bitcoin? Is Bitcoin truly secure if 50% of the mining computing power is honest?
- To which extent do the scalability measures adopted in Bitcoin threaten the underlying security of the system?
- To which extent does Bitcoin offer privacy to its users? How can one quantify the user privacy offered by Bitcoin?
- Are lightweight clients secure? To what extent do lightweight clients threaten the privacy of users?
- What are the proper means to secure Bitcoin wallets?

- Who effectively controls Bitcoin?
- How do the security and privacy provisions of other blockchain technologies compare to Bitcoin?
- What are the security lessons learned after 8 years of massive research into Bitcoin?

Thoroughly reporting on security and privacy vulnerabilities of systems can be often confused with criticism. The aim of this book is solely to provide our readers with the first in-depth analysis of the Bitcoin system with the goal of laying down the basic foundations for constructing next generation secure blockchain currencies and technologies. Based on recent incidents and observations, we additionally show that the vital operations and decisions that Bitcoin is currently undertaking are not decentralized. More specifically, we show that a limited set of entities currently control the services, decision making, mining, and incident resolution processes in Bitcoin. We also show that third-party entities can unilaterally decide to "devalue" any specific set of Bitcoin addresses pertaining to any entity participating in the system. In the following section, we present a detailed outlook of the contents of this book.

1.1 BOOK STRUCTURE

The remainder of this book is organized as follows.

1.1.1 Chapter 2

In Chapter 2, we start with an overview of the predecessors of Bitcoin and their associated crypto-based payment schemes, with a particular focus on their security, privacy provisions, and implementation deficiencies. We also define the notions of payment security and privacy as considered in existing payment systems. As such, this chapter provides the necessary background knowledge for readers to assess cryptocurrencies that emerged prior to Bitcoin and understand the various gaps that could not be captured by previous proposals—these were mainly the gaps that Bitcoin promises to fill.

1.1.2 Chapter 3

In Chapter 3, we detail the operation of Bitcoin and summarize the main scalability measures integrated in the system. We explain the cryptographic building blocks that Bitcoin leverages and detail the various data structures used in the Bitcoin system. We also describe the different roles that participants can assume in the Bitcoin ecosystem. As such, this chapter lays down those foundations of the Bitcoin protocol that are essential for the readers to dive into the security and privacy provisions of the system in the following chapters.

1.1.3 Chapter 4

In Chapter 4, we thoroughly analyze the security provisions of Bitcoin in light of recent published attacks, and we discuss possible countermeasures. For instance, we show that the initial measures adopted in Bitcoin to handle fast payments are not enough to deter double-spending attacks, and discuss a first workable countermeasure against double-spending that is currently integrated in Bitcoin. Fast payments refer to payments where the time between the exchange of currency and goods is short (in the order of a minute). While the Bitcoin proof-of-work (PoW) based time-stamping mechanism is essential for the detection of double-spending attacks (i.e, in which an adversary attempts to use some of his or her coins for two or more payments), it requires tens of minutes to verify a transaction and is therefore inappropriate for fast payments. Clearly, there is only limited value in verifying the payment after the user has obtained the goods (and, e.g., left the store) or services (e.g., access to online content).

We also show that an adversary can deny the delivery of blocks and transactions to victim Bitcoin nodes for a considerable amount of time. We show that this can be achieved by exploiting Bitcoin bandwidth optimization techniques and the measures that are in place to tolerate network delays and congestion. The minimal requirement for this attack to succeed in practice is simply that the attacker can establish at least one connection to the victim. An even more powerful attack resulting in almost indefinite delays at the victim node only requires that the attacker can fill the victim's remaining open connection slots—without necessarily causing any network partitioning in the Bitcoin network.

These results therefore motivate the need for a careful design of the scalability mechanisms adopted in Bitcoin. While existing mechanisms limit the amount of propagated information in the system to the minimum necessary, we show that these techniques come at odds with security and reduce the ability of the network to, for

example, detect double-spending attacks, resolve, or prevent blockchain forks. For instance, these findings suggest that an adversary who commands more than 33% of the computing power in the network can control the fate and security of all Bitcoin transactions. In this respect, we describe a modification of the block request process in Bitcoin to deter this misbehavior.

1.1.4 Chapter 5

In Chapter 5, we address user privacy in Bitcoin. Namely, in spite of the reliance on pseudonyms, the public time-stamping mechanism of Bitcoin raises serious concerns with respect to the privacy of users. In fact, given that Bitcoin transactions basically consist of a chain of digital signatures, the expenditure of individual coins can be publicly tracked.

In this chapter, we evaluate the privacy that is provided by Bitcoin. This is achieved (1) by investigating the behavior of Bitcoin client and exploiting its properties, and (2) by evaluating the privacy provisions in light of recent reported attacks on the system. Motivated by these attacks, we also discuss a number of possible measures that can be used to enhance the privacy of users in Bitcoin. Here, we cover system-based solutions, such as CoinJoin and mixers, as well as cryptographic-based solutions that enable privacy-preserving payments atop Bitcoin—such as ZeroCoin, Extended ZeroCoin, and ZeroCash.

1.1.5 Chapter 6

In Chapter 6, we analyze the security and privacy of lightweight Bitcoin clients. These clients support a simplified payment verification (SPV) mode where only a small part of the blockchain is downloaded—thus enabling the usage of Bitcoin on constrained devices (e.g., smartphones, cheap virtual private servers). SPV clients were proposed by Nakamoto in the original white paper and were later extended to rely on Bloom filters in order to receive transactions that are relevant to their local wallet. These Bloom filters embed all the addresses used by the SPV clients, and are outsourced to more powerful Bitcoin nodes; these nodes will then forward to the SPV clients those transactions relevant to their wallets. Besides analyzing the security of existing SPV implementations, we also explore their privacy provisions due to the use of Bloom filters. We show that the current integration of Bloom filters within Bitcoin leaks considerable information about the addresses of Bitcoin users. This analysis is not only restricted to Bitcoin, but equally applies to other digital currencies that rely on similar SPV implementations. Our findings therefore

motivate a careful assessment of the current implementation of SPV clients prior to any large-scale deployment.

1.1.6 Chapter 7

In Chapter 7, we analyze the current Bitcoin ecosystem. Although Bitcoin does not truly solve all of the challenges faced by previously proposed digital currencies, Bitcoin grew to witness a wider adoption and attention than any other digital currency proposed to date. At the time of writing, Bitcoin holds the largest market share among all existing digital currencies.

In this chapter, we overview the main operation of Bitcoin and describe a number of businesses, exchange platforms, and wallets that are currently built around the Bitcoin ecosystem. We also analyze the limits of decentralization in the Bitcoin ecosystem. Namely, based on recent incidents and observations, we show that the vital operations and decisions that Bitcoin is currently undertaking are not decentralized. More specifically, we show that a limited set of entities currently control the services, decision making, mining, and the incident resolution processes in Bitcoin. We also discuss the security of online wallets and outline a number of innovative techniques to ensure the protection of private keys against compromise and/or loss.

1.1.7 Chapter 8

In Chapter 8, we overview a number of interesting applications built atop Bitcoin's blockchain. Namely, we describe Namecoin, the first clone of Bitcoin, which implements a decentralized Domain Name Service for registering Web addresses that end in ".bit," and which is resilient to censorship. We then overview Litecoin and Dogecoin, two of the most known altcoins derived from Bitcoin. We also discuss other applications of the Bitcoin blockchain, such as decentralized and authenticated storage and smart contracts. We additionally show how Bitcoin can be used to instantiate a decentralized time-dependent randomness generator. Finally, we discuss current efforts to repurpose the proof-of-work of Bitcoin toward useful computations, among other proposals by digital assets and sidechains to extend the basic functionality of Bitcoin.

1.1.8 Chapter 9

In Chapter 9, we overview a number of interesting blockchain proposals that are currently competing with Bitcoin. These proposals have been mainly motivated by

the success of Bitcoin and attempt to solve some of the caveats encountered in the Bitcoin system.

Namely, we describe Ripple, Ethereum, and the IBM Open BlockChain technologies. We compare these blockchains to Bitcoin with respect to their security and privacy provisions.

1.1.9 Chapter 10

Finally, in Chapter 10, we summarize the main lessons learned from the previous chapters. Namely, we summarize the security and privacy provisions of Bitcoin, and its underlying blockchain—effectively capturing 8 years of thorough research on these subjects. In addition to discussing existing vulnerabilities of Bitcoin and its various related altcoins, we also summarize possible countermeasures to deter threats and information leakage within the system.

As far as we are aware, this book offers the most comprehensive and detailed analysis of the security and privacy provisions of Bitcoin and of its related clones/variants. We hope that the contents of the book provide the necessary tools and building blocks for the design of secure next-generation blockchain technologies.

Chapter 2

Background on Digital Payments

In this chapter, we provide an overview of the predecessors of Bitcoin and their associated crypto-based payment schemes. In particular, we define the notions of payment security and privacy as established in already existing payment systems. Next, we provide an overview of alternatives to banking-based payment technologies that preceded Bitcoin, with a particular focus on their security, privacy provisions, and implementation deficiencies (if any).

More specifically, in Sections 2.1 and 2.2, we present a generic payment model, detailing its architecture and security and privacy requirements. In Section 2.3, we list a number of desirable properties of payment systems and their impact on security and performance. We also investigate prominent deployed payment schemes prior to Bitcoin that seek to achieve those properties.

2.1 PAYMENT SYSTEMS ARCHITECTURE

As the name suggests, payment systems facilitate the exchange of money between two entities—a payer and a payee. Apart from the payer and payee, a payment system traditionally involves two more entities; one entity that manages assets and/or funds on behalf of the payer, known as the *issuing* bank (or *issuer*), and another entity that maintains an account for the payee, known as *acquiring* bank, or *acquirer*.[1] For simplicity, we will use the terms payer/payee interchangeably to refer to the buyer/merchant, and we refer to all other parties as *users*.

1 In practice, the *acquirer* and *issuer* can represent the same physical entity (e.g., bank).

In what follows, we adapt the classification of payment systems from [1]. Namely, we distinguish between *cash-like* payments, where payers need to withdraw their funds before using them in payments and *check-like* payments, in which the payers do not need to engage in a withdrawal operation prior to committing to a payment (and the money withdrawal takes place later in time). Figures 2.1 and 2.2 depict the respective architectures of cash-like and check-like systems.

The operations of a typical cash-like system are depicted in Figure 2.1. In a cash-like system, the payer's account is charged before the actual payment takes place. That is, the payer first contacts the *issuer* to withdraw some funds from his or her account. The payer can obtain his or her funds in various forms (e.g., in a credited smart-card, electronic cash). The payer and payee subsequently interact for the requested payment amount to be deducted from the payee's funds. The *acquirer* is made aware of the payment through a special *deposit* operation, where the payee deposits the payments that he or she has received.

The interactions of users and banks within check-like system are depicted in Figure 2.2. As opposed to cash-like systems, in check-like payments, the account of the payer is charged after the payment actually takes place (or concurrently with the payment). The latter case captures a credit card payment. Typically, in a check-like system, a payment request is initiated by the payer who sends the payee a *check* paying the latter. The payee forwards the request to the *acquirer* that notifies the *issuer*. The issuer evaluates the payment request and if it deems it valid, it settles the payment with the *acquirer*. Depending on the protocol, the issuing bank may send a message to the payer requesting a final approval of the payment or a notification that the payment was successfully processed (if the payment request already contains enough information).

Another popular means of classifying payment schemes is to categorize them into *interactive* and *noninteractive* based on whether they require the active participation of both parties.

Extensions of such architectures incorporate mediators that perform the payments on behalf of the users following the user requests. Naturally, in mediator-based payment systems, payers do not directly communicate with their bank account. Instead, they manage their funds through accounts opened with a third-party that is further responsible to send user-authenticated payment requests as defined by the protocol. Mobile phone-enabled payments can be considered as prominent example of such payments. Another variant of such architectures involves systems like PayPal [2], where users open an account to which they transfer money from their bank account. Payments can be executed in this way by any user who owns a PayPal account.

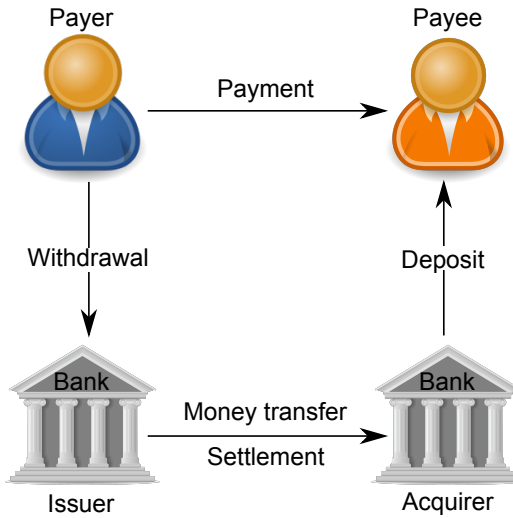

Figure 2.1 Cash-like payment system architecture.

Depending on the type of interaction, such mediators can also play the role of payment escrows; these are entities that can monitor a particular transaction and ensure the proper exchange of money and goods before the payments are confirmed. Though such a functionality has been popular during the last few years, older systems such as PayPal can be considered as pioneering examples of this category.

2.2 SECURITY AND PRIVACY IN PAYMENTS

In this section, we refer to well-established security and privacy requirements among payment systems. In particular, we define these properties in the context of payment systems and overview the challenges associated with each of them.

Strictly speaking, security in information systems is defined by the combination of information integrity, availability, and confidentiality. In some other contexts, security can be obtained using the combination of authentication, authorization, and identification. As we explain in Section 2.2.1, payment systems require security properties that belong to both security descriptions.

Privacy defines the right of each individual to determine the degree to which it will interact and share information with its environment. Privacy is a fundamental

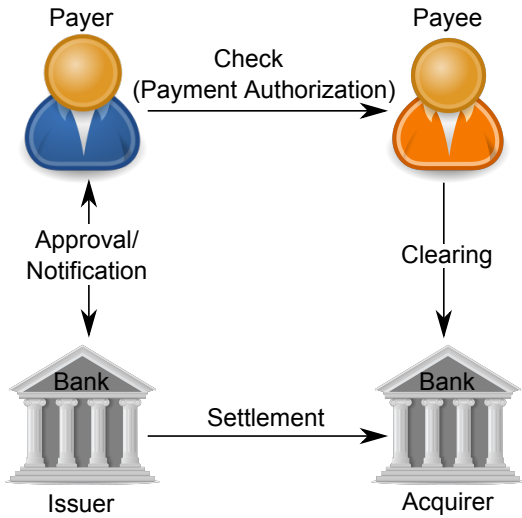

Figure 2.2 Check-like payment system architecture.

right of individuals and strongly relates to data confidentiality. Namely, providing privacy guarantees in payment systems is of crucial importance, and has been the clear focus of researchers and system developers.

Given this, we discuss privacy/confidentiality of payment systems in Section 2.2.2. In Section 2.2.1, we focus on other security properties of such systems, such as integrity and availability.

2.2.1 Security

We start our quest by considering the breakdown of security in integrity, availability, and confidentiality, and continue with other security concepts that are specific to payment systems, such as fairness and resistance to impersonation attacks, among others.

Integrity is defined as the assurance that information is not altered or modified except by properly authorized individuals. Thus, for integrity purposes, maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle becomes crucial. In the context of payment systems, integrity is important when it comes to the integrity of payment records and payment requests. That is, no unauthorized party should be able to alter the contents of a particular payment

request without invalidating the payment request itself or being detected. Common measures to enable data integrity verification against intentional or accidental data modifications include the use of cryptographic techniques, such as cryptographic checksums. Other measures involve controlling the physical environment of networked terminals and servers, restricting access to data, and maintaining rigorous authentication practices. Data integrity can also be threatened by hardware failures or environmental hazards, such as heat, dust, and electrical surges.

Availability ensures that the payment system can serve authorized user requests, and fulfill its purpose whenever the service is supposed to be active. Ensuring this property expands to many aspects of the payment system. Moreover, providing adequate communication bandwidth and preventing the occurrence of bottlenecks are equally important when designing secure payment systems.

Apart from the basic security requirements, security in payment systems has been commonly bound to the *fairness* of the underlying payment mechanism, as well as to the *resistance to impersonation attacks* and *accountability*.

Fairness requires that a particular individual cannot commit to more payments than its assumed account balance. For example, users should not be able to use their credit card to pay more than their credit limit, or use their debit card to pay more than their debit account balance. This property is also commonly known as *balance* [3] in the literature. On the other hand, resistance to impersonation attacks requires that no one should be able to impersonate other users or perform payments on behalf of other users without their consent.

Nonrepudiation is another aspect of the same property that requires that a user should not be able to deny a transaction that he or she carried/authorized.

Finally, accountability requires that a user who *misbehaves* (i.e., behaves against the regulation of the account holder or the law) can eventually be accountable for his or her acts. This concept of security is also strongly associated with *nonrepudiation*.

Common mechanisms and concepts to provide integrity, confidentiality, and (implicitly) availability in payment systems enforce strong authentication and proper authorization. Authentication is the method or mechanism for ascertaining that somebody really is who they claim to be. Authorization, on the other hand, refers to rules that determine who is allowed to do what in a system.

2.2.2 Privacy

As mentioned earlier, privacy is the right of each individual to determine the degree to which they will interact and exchange information with their environment. As

such, privacy is strongly associated with data confidentiality, which mandates that data should not be accessible by unauthorized individuals.

Popular privacy concepts in the context of payment systems consist of *transaction anonymity* and *transaction unlinkability*. Assuming a set of user identities that make use of a payment system, transaction anonymity requires that one cannot link a particular transaction to a specific identity more than to any other identity that is part of the system. On the other hand, transaction unlinkability requires that two transactions of the same individual cannot be linked as such. Depending on the system, privacy of clients committing to transactions can be considered from the perspective of the banks, and/or users' transaction partners.

Given that the transactions of a particular individual contain sensitive information about consumers, privacy has been the subject of investigation in payment systems for a considerable amount of time. As we shall see later, cryptographic primitives, such as anonymous credentials and anonymous electronic cash, are means that were proposed almost a decade ago to protect the fundamental rights of individuals.

2.2.3 Combining Security and Privacy

During the design of security and privacy mechanisms for payment systems, banks were assumed to be rational entities aiming to maximize their profit. Rational entities refer to entities that would only deviate from the protocol (i.e., act maliciously) if such a misbehavior would increase their advantage in the system. Thus, banks would behave as the protocol suggests as long as their activity is traceable, but could be tempted to attack the privacy of their clients if such an activity could not be traced back to them. Using such information about their clients, banks can profile their clients, and/or sell their data to third parties, and so forth. This was very accurately reflected by the degree to which security and privacy mechanisms were adopted by banks.

Security emerges as one of the most critical properties for payment systems. Unless users are sure that they are not in danger of losing their funds, they would not leverage any payment service of a particular bank. As a consequence, most banks have invested considerable resources in devising secure mechanisms for performing payments in the off-line and online realms.

However, this model did not work the same way for privacy mechanisms. While banks have been (and still are) investing funds in *anonymizing* their client data (e.g., when sending them to third parties for processing) they have been neglecting mechanisms to make client payments privacy-preserving with respect to

them. Notice that privacy-preserving payments would prevent banks from profiling their clients in terms of payments and increases their risk in services such as granting a loan. This is the main reason why privacy-preserving mechanisms associated with bank payments have not yet been adopted.

2.3 SECURITY IN PAYMENT SYSTEMS PRIOR TO BITCOIN

In this section, we elaborate on prominent research and industrial payment systems (and their security properties) that started prior to Bitcoin. In particular, in Section 2.3.1, we elaborate on the security vulnerabilities featured in these payment systems. By extracting appropriate lessons from these vulnerabilities, we overview research results in the field of digital payment systems and show how these systems resist or deal with the aforementioned threats in Sections 2.3.2 and 2.3.3. In our overview, we focus on the systems that are of historical importance or became popular throughout their deployment.

2.3.1 Common Payment System Characteristics

Bitcoin is a payment system that satisfies the following properties:

- Liveliness of payments
- No need for payment mediator
- Decentralization of trust in payment processing
- Support for micropayments
- No need for (expensive) specialized hardware

We elaborate in this section on each of these properties separately by extracting lessons from the security vulnerabilities witnessed by previously deployed payment systems.

2.3.1.1 Liveliness of the Interaction with the Banks

One can classify payment systems in two broad categories as described in [1]: *online payments* and *off-line payments*, depending on whether there is a need to contact a single (trusted) entity for the payment to take place (in addition to the transaction

participants). Clearly, online payment systems require that a contact with a third-party, such as with an authentication or authorization server (e.g., a server of the bank confirming a transaction), is necessary whenever a payment is to take place. In contrast, in *off-line* systems, payments can be executed by requiring only the active participation of the payer and the payee. Credit card payments are examples of online payments, while prepaid card-based payments are prominent examples of off-line payments.

From a security perspective, off-line systems are vulnerable to *double-spending* attacks, where an adversary attempts to spend more than the available (off-line) balance (pretending that another off-line payment did not take place). Such attacks against fairness have been identified in the literature and resulted in the incorporation of a number measures to prevent/detect such threats. Among these measures, hardware-based approaches are the most widely used in order to deter double-spending in off-line payments (e.g., by utilizing smart cards). Another method to offer double-spending resistant off-line payments is to rely on nominal checks that have already been preapproved by a third-party to be received by the specific merchant [1]. Finally, additional methods are based on detection of double-spending acts or on tracing and punishing the double-spenders [3].

2.3.1.2 Mediator-Based Services

Mediator-based systems refer to systems where payments are not performed directly by the payer and its counterparties, but through a mediator that offers a payment escrow service to the payer. In such systems, users maintain accounts that they credit/debit directly from their bank accounts. Subsequently, users can make payments to any other user who maintains an account with the same service. PayPal [2] is a representative example of such systems. In many cases, apart from executing the payment itself, these services act as escrows of the fairness of the transaction that takes place.

Though such systems do not tend to suffer from double-spending attacks, they place complete trust in the mediator. That is, if the mediator (service) is compromised or acts maliciously, the security and privacy of transactions can be set at risk. Although timed in a period after Bitcoin was introduced, MtGox [4] is a prominent example of such service. Depending on the service and the trust the user has to put in it (e.g., account details, strong identification information), more funds can be withdrawn by the user's account to perform payments on his or her behalf without the latter's actual consent.

2.3.1.3 Decentralization of Trust

Payment systems can be centralized; that is, they can have one designated entity that is authorized to approve or reject payments on behalf of a particular individual. Payments executed through banks belong to this category.

A payment system is considered to be decentralized if a limited number of specially designated entities participate in the processing of transactions. Moreover, we say that a system is open if any entity can participate in the procedure of confirming or rejecting a transaction. Bitcoin is a prominent example of this category of payment systems.

In the latter two cases, specially crafted consensus protocols are in place to guarantee that the transaction processing is atomic (i.e., a transaction is either approved or confirmed or rejected). The same problem is not present in centralized payment systems, where a single entity receives and processes the user payment requests.

Though centralized systems do not suffer from double-spending attacks, they put complete trust on the single entity that processes payments. That is, if this service starts acting maliciously (e.g., is compromised), the security of the system and privacy of transactions are at serious risk.

2.3.1.4 Support for Micropayments

Micropayments, such as MiniPay [5], are payments of low-value that in many scenarios occur repeatedly and fast. Due to their low value, micropayment systems suffer from two fundamental problems.

First, the payment processing cost in these systems may be higher than the actual payment value if payments are performed similarly to conventional payments. Naturally, this can be against the benefit of the party that processes transactions or require a transaction fee to the payer that is disproportional to the payment amount. M-Pesa [6] is a prominent example of this case.

Second, for the common case of micropayments that occur frequently, payment processing should be considerably faster than conventional payments. Therefore, such systems may deviate in terms of performance and security requirements from systems serving conventional payments and could require different system designs.

2.3.1.5 Need for Special Hardware/Software

As mentioned in Section 2.3.1.1, to support certain security properties, tamper-resistant hardware is needed on the payer or payee side or on both parts. Additionally, recent schemes to support privacy-preserving transactions require sophisticated cryptographic operations to be implemented and performed on the payer and payee side. We classify the specifications of existing payment systems in the following categories:

- Specification requiring *asymmetric cryptographic* operations (e.g., some public key infrastructure to be in place).
- Specification requiring only *symmetric cryptographic* operations, where only symmetric keys are needed and the associated symmetric operations to be implemented.
- Specification requiring secure *hash functions*, where there is only a need for an implementation of a hash function.

Clearly, the more complex the cryptographic operations utilized within a system, the more complicated is the hardware needed to accelerate the computations taking place within it, and the more necessary and expensive that hardware becomes. Although the cost of hardware is not a primary issue in payment systems, it tends to considerably influence the adoption of a given payment system.

2.3.2 Privacy-preserving Payments Due to the Research Community

As mentioned previously, privacy-preserving payments attracted the attention of the research community in the early years of applied cryptography. In the following, we introduce *digital* or *electronic cash*, which aimed to replace conventional *cash*, and proceed with primitives to enable privacy-preserving *checks* and *credit card-based payments*. Finally, we provide a look at other use cases related to privacy-preserving payments, such as stock market and taxation.

2.3.2.1 Digital Cash

In a first attempt to offer privacy-preserving payments that would reflect the real-world needs, in their paper "Revokable and Versatile Electronic Money" [7], Jakobsson and Yung first presented the necessity for revokable anonymity in payments. To achieve this, users maintain their funds in bank accounts that carry their name

or identity. To make payments, users withdraw anonymous coins from these accounts using a three-party protocol that takes place between the user, the bank, and a trusted "ombudsman" in charge of revocation. The user chooses a coin sequence number, and using a blind signature system,[2] the bank, and ombudsman generate a bank signature on information related to the coin. Clearly, the ombudsman, can potentially revoke the anonymity of a user. Although satisfying the need for revokable anonymity in funds management, this scheme does not protect account activity information from the bank, since accounts themselves were not anonymous.

Digital cash or electronic cash was introduced by Chaum [8] as a cryptographic primitive to offer privacy-preserving cash-like payments. In its first version, digital cash is offered as a substitute for money on the Internet that cannot be faked, copied, or spent more than once, as long as the online participation of a bank can be guaranteed. In addition, it is known to provide absolute anonymity, namely no one, not even the bank itself, can relate a particular ecash coin (ecoin) with its owner. Consumers can indeed buy anonymous ecoins from a bank/mint and use them in their online transactions without being traced. Camenish, Lysyanskaya, and others [9–11] enhanced the work in [8] with accountability features, while offering the possibility of off-line transactions with resistance to double-spending. In [11], an ecash-based electronic payment system was introduced by taking into consideration real world system threats.

More concretely, an ecash (EC) [9, 12] system consists of three types of players: the *bank*, *users*, and *merchants*.

To open a bank account, users engage in a registration process through which they generate cryptographic keys to be identified within the system (EC.GenKey). When they need to perform a payment, the users contact the banks and withdraw funds in the form of ecash coins (EC.Withdraw). In most schemes, the users spend their coins among other users without the need to contact the bank at the time (EC.Spend). Users are not required to reveal their identities through EC.Spend and may participate in transactions through one-time pseudonyms [3, 13]. However, they need to use the secret keys they used during EC.Withdraw for the payment protocol not to fail. After the payment completes, merchants need to deposit their coins in the bank to allow double-spending detection to take place (EC.Deposit). These keys (and thus the identity of a payer) can be revealed (through EC.Identify) only if the payer attempts to double-spend a coin (confirmed through EC.VerifyGuilt). Depending on the scheme, it is only the identity of

2 Blind signatures refer to a cryptographic primitive that allows an entity to digitally sign a message without knowing or being able to read the message that it signs.

the double-spender or the entire set of his or her transactions that are revealed (EC.Trace).

A summary of the input and output specifications of the basic operations is listed below.

- $(pk_{\mathsf{B}}, \mathrm{sk}_{\mathsf{B}}) \leftarrow \mathsf{EC.BGenKey}(1^k, params)$ and $(pk_u, \mathrm{sk}u) \leftarrow \mathsf{EC.UKeyGen}(1^k, params)$, which are the key generation algorithms for the bank and the users, respectively.
- $\langle W, \top \rangle \leftarrow \mathsf{EC.Withdraw}(pk_{\mathsf{B}}, pk_u, n)\ [u(\mathrm{sk}_u), \mathsf{B}(\mathrm{sk}_{\mathsf{B}})]$. In this interactive procedure, u withdraws a wallet W of n coins from B.
- $\langle W', (S, \pi) \rangle \leftarrow \mathsf{EC.Spend}(pk_{\mathcal{M}}, pk_{\mathsf{B}}, n)\ [u(W), \mathcal{M}(\mathrm{sk}_{\mathcal{M}})]$. In this interactive procedure, u spends a digital coin with serial number S from his or her wallet W to $\mathcal{M}$. When the procedure is over, W is reduced to W', $\mathcal{M}$ obtains as output a coin (S, π), where π is a proof of a valid coin with a serial number S.
- $\langle \top/\bot, L' \rangle \leftarrow \mathsf{EC.Deposit}(pk_{\mathcal{M}}, pk_{\mathsf{B}})\ [\mathcal{M}(\mathrm{sk}_{\mathcal{M}}, S, \pi), \mathsf{B}(\mathrm{sk}_{\mathsf{B}}, L)]$. In this interactive procedure, $\mathcal{M}$ deposits a coin (S, π) into its account in the bank. If this procedure is successful, $\mathcal{M}$'s output will be $\top$ and the bank's list L of the spent coins will be updated to L'.
- $(pk_u, \Pi_G) \leftarrow \mathsf{EC.Identify}(params, S, \pi_1, \pi_2)$. When the bank receives the two coins with the same serial number S and validity proofs π_1 and π_2, it executes this procedure to reveal the public key of the violator accompanied with a violation proof Π_G.
- $\top/\bot \leftarrow \mathsf{EC.VerifyGuilt}(params, S, pk_u, \Pi_G)$. This algorithm, given Π_G, publicly verifies the violation of pk_u.
- $\{(S_i, \Pi_i)\}_i \leftarrow \mathsf{EC.Trace}(params, S, pk_u, \Pi_G, D, n)$. This algorithm provides a list of serials S_i of the ecoins a violator pk_u has issued, with the corresponding ownership proofs Π_i.
- $\top/\bot \leftarrow \mathsf{EC.VerifyOwnership}(params, S, \Pi, pk_u, n)$. This algorithm allows to publicly verify the proof Π that a coin with serial number S belongs to a user with public key pk_u.

Camenish et al. presented in [12] a money-laundering prevention version of [9], where anonymity is revoked when the spender spends more coins with the same merchant than their spending limit. In this case ecoins are upgraded to:

$$C = (S, V, \pi),$$

where V is a merchant-related locator, while EC.Identify and EC.VerifyGuilt procedures are upgraded to the DetectViolator and VerifyViolation to support the extended violation definition.

Security Properties

Electronic cash systems are built with the following security properties: correctness, fairness, (conditional) anonymity, and resistance to impersonation attacks. Correctness requires that the electronic cash protocols work as intended if all players are honest. Fairness requires that no collection of users and merchants can ever spend more coins than they withdraw. On the other hand, anonymity of users requires that no entity, even the bank itself when colluding with any collection of (malicious) users and/or merchants can obtain information on a user's spending other than the information intentionally released in the network by the user (and associated side-information).

User anonymity is commonly conditional on honest user behavior. That is, given a violation with respect to a predefined policy, electronic cash system can output proofs of guilt Π_G and the violators' public keys pk_{V} such that EC.VerifyViolation accepts. Such a proof of violation enables systems to support the *traceability of a violator's behavior*. That is, EC.Trace will output the serial numbers of all coins that belong to the violator with public key pk_{V} along with the corresponding proofs of ownership that VerifyOwnership accepts.

Finally, resistance to impersonation attacks requires that an honest user u cannot be accused of conducting a violation that EC.VerifyViolation accepts.

Summary

Although initial schemes offering digital cash were online-based, recent digital cash payment systems are off-line, offering to some extent resistance to double-spending (or misbehavior). In particular, ecash schemes reveal the identity of the double-spender and/or the entirety of their transactions. Except for their initial version, those schemes do not require any trusted third-party (mediator). Finally, despite their privacy guarantees, these systems usually incur complicated cryptographic operations (e.g., blind signature) and are therefore penalizing in terms of performance.

2.3.2.2 Credit Card-Based Payment Protocols

Credit card-based transactions tend to be popular for transactions over the wire, due to their easiness of use and their risk management features. These protocols support delayed payment, provide users with logs of their own transactions, receipts, and the opportunity to challenge and correct erroneous charges. In fact, frequent losses of credit cards, as well as impersonation attacks, justify the fact that banks (that

are no more trusted than the people operating them) maintain a detailed log of user transactions. At the same time, users acting as payees tend to see the identities of the payers, when a credit card is used for payments. The latter constitutes a serious violation of privacy of card holders with regard to banks. To remedy that, anonymous payment cards were introduced as the privacy equivalent of credit or debit cards.

Credit cards providing cardholder anonymity even toward the banks were introduced in 1994 by Low et al. [14] in their paper "Anonymous Credit Cards and its Collusion Analysis" [15]. Here, the authors present a system of anonymous lending accounts whose aim is to allow users to spend credit without revealing their identities to the stores. To achieve this, a user makes use of two banks, a *credit bank* who knows their identity (because they are extending the credit), and a *payment bank* who receives funds from the credit bank on behalf of the user and does not need to know the user's identity. The user is known to the payment bank only as an authentication key or pseudonym for signing instructions. The stores participating in this system make use of *store banks*. To establish credit, the user asks the credit bank to issue funds (up to a given credit limit) to the payment bank. The user can then spend credit at stores by anonymously instructing his or her payment bank to transfer funds to the appropriate store bank.

Although users have anonymous accounts with the payment bank, user privacy is not offered in this system when considering collusion between the payment and store banks. That is, a collaboration between these two banks could link a user's payments to the store. A number of other schemes, such as [16, 17], also proposed to blind credit card information from third parties or merchants, respectively but *not toward banks*.

Androulaki and Bellovin [18] recently introduced a different system for managing anonymous lending accounts. This system eliminates the need for a credit bank that knows the identity of the user. To do this, it makes use of two types of digital cash *wallets* introduced by Camenisch, Hohenberger, and Lysyanskaya [9, 12]. One allows for anonymous accounts that hold a specified amount of funding and reveal the identity of the owners if that threshold is exceeded. The other allows for similar limits, but reveals an entire transaction history of the corresponding user account if the same threshold is exceeded. For normal transactions, these wallets are filled with an appropriate spending limit and payments are made from both to ensure that those who exceed their limit are detected and dealt with appropriately. To handle monthly payment of debt, only the wallet is accessed to prove that a specific individual has used a specific amount of his or her limit.

Summary

In general, proposed privacy-preserving credit card schemes offer anonymity and/or transaction unlinkability. Security is achieved in all cases by requiring the user to provide his or her secret key at each transaction; however, upon compromise of the user-side software, fraud detection can only take place on the user side (not through the bank). Clearly, these systems enable off-line payments and detect/identify double-spending (e.g., in the case of digital cash). Such systems leverage advanced cryptographic primitives, but do not support micropayments.

2.3.2.3 Micropayments

A number of payment schemes addressing the delicate requirements of micropayments (i.e., low processing fee and fast processing time) have been proposed in the literature [19–21]. Micropayment schemes adopt the principle that not all payments need to be processed—but only a representative sample of those payments (due to their low value).

In what follows, we give the example of how MiniPay works. The scheme assumes there is a one-way hash function H, a PKI in place and that users of the system, payers and payees, generate and certify their keys upon opening an account to a bank. To make a payment, a user u (with public key pk_u) simply digitally signs a statement of moving a certain amount of his or her funds to the payee's account, $\mathcal{M}$ (with public key $pk_{\mathcal{M}}$):

$$\sigma_{u \to \mathcal{M}} \leftarrow \mathrm{Sign}(\mathrm{sk}_u; pk_{\mathcal{M}} || val),$$

where sk_u is the secret key that corresponds to pk_u, and *val* refers to the value of the transaction.

Upon receiving the payment, the payee checks if the received payment is depositable; that is, the payee checks if the result of the hash function on the payment satisfies certain conditions, such as those expressed through $Cond$:

$$\mathrm{Cond}(\mathsf{H}(\sigma_{u \to \mathcal{M}})) = \mathrm{true}.$$

The payee eventually deposits the payment to the bank, which moves $x \cdot val$ from the account of u to the one of $\mathcal{M}$, where $\mathrm{x} > 1$ is a system parameter that is there to account for nondepositable payments of u.

Privacy-preserving versions of micropayments have been introduced in the literature [19, 22]; here, users leverage cryptographic primitives equivalent to the

ones used in ecash to hide the identities of transaction participants and make transactions of the payer unlinkable. However, the privacy of the recipient is not guaranteed.

Summary

MiniPay is a scheme that addresses micropayment requirements with some privacy offerings. Payments in MiniPay do not require contacting a bank and can therefore be considered off-line. At the same time, double-spending is treated by leveraging accountability properties of pure digital signatures that users use to authorize their transactions. Even in this case, security of the system would need to rest upon trusted software and hardware that run the payment generation on the client side.

2.3.2.4 Other Privacy-Preserving Payments

In what follows, we provide an overview of privacy-preserving payment systems that are crafted for the context of anonymous taxation of investments, stocks, and bank account balances.

In [23], Xu et al. propose a scheme addressing privacy issues related to stock market accounts, by offering anonymous and taxable stock market trading accounts. This system addresses the problem of tax reporting in an anonymous system. Each user anonymously hold funds (or stocks), and the taxable amounts must be reported to the taxation authority.

Here, investors have normal certificates, which they may use to generate a conditionally anonymous certificate (CAC) that represents their anonymous accounts. These CACs are recognized by the Stock Exchange Center (SEC) and used to verify signed orders indicating stock market-related activities. They are normally unlinkable to their original certificates. However, in cases of misbehavior, a CAC may be opened by an appropriate authority to reveal the identity of the owner. During taxation, the SEC prepares reports for each anonymous account and provides them to the tax authority, to which the user can prove ownership of and pay the appropriate taxes. The system leverages user-generated anonymous credentials from a public credential to validate anonymous transactions.

2.3.3 Deployed Payment Systems

In this section, we provide an overview of payment systems that have been already deployed.

2.3.3.1 Zero-Knowledge Systems

Zero-Knowledge Systems (ZKS) was a company founded in 1997 to offer primarily privacy-preserving services. ZKS is well known for Freedom network, its privacy-preserving network service.

The main differentiator of ZKS was the strong privacy provisions offered within their services. ZKS technologies leveraged privacy-preserving cryptographic primitives at the time introduced by Brands along the lines of the schemes described in Section 2.3.2.

2.3.3.2 PayPal

Established in 1998, PayPal started as a company offering its clients the ability to transfer funds electronically between individuals, organizations, or businesses [2]. Clients of PayPal have to register accounts and once these accounts are connected to their bank/credit/debit account, they are able to send funds to anyone in possession of a PayPal account.

PayPal accounts are traditionally associated to an email address, and knowledge of an email address is the minimum knowledge needed to move money to the account owned by that address. PayPal transactions are pseudonymous and allows users to register several accounts, each linked to a given bank account or email address. Therefore, one could argue that PayPal does not offer transactional privacy as defined in the previous sections.

Currently, a number of online markets, such as eBay and Amazon, accept PayPal payments. In typical cases, PayPal accounts are automatically refilled by their owners's bank account. To secure the PayPal account refill process, PayPal introduced two-factor authentication mechanisms. In particular, users would have to authenticate using their username and password and answer an additional challenge to be able to log in. The challenge was either a secret code sent to the user's mobile phone or a number that the user was supposed to feed to a pre-agreed hardware security module. This two-level authentication would prevent a potential attacker from modifying any transaction: the attacker would have to obtain access to the user's password and would also need to obtain the right answer to the challenge (by compromising another user device). Notice that the PayPal implementation of two-factor authentication does not seem to solve the issue of *man-in-the-middle-attack* since a potential attacker could try to impersonate the PayPal website to the user (and vice versa) just by forwarding responses from one end to the other.

Summary

PayPal is a mediator-based system that enables payments to users who maintain accounts through PayPal. The system does not offer privacy and requires trust to be placed in the mediator—who has to be online for the payment to complete.

2.3.3.3 IBM Micropayments

IBM Micropayments was developed by IBM Research and aims to efficiently support small-value transaction payments over the Internet. IBM Micropayments essentially implement the techniques introduced in MiniPay [5].

In IBM Micropayments, each entity (e.g., financial institution or Internet-service provider) manages their own risk by operating their own billing service. Recently, these entities can also offer billing support as a service to consumers and merchants. IBM Micropayments supports interoperability between different types of billing systems—which led to a widespread adoption of this system.

2.3.3.4 Peppercoin

Similar to IBM Micropayments, Peppercoin [24] is another prominent micropayment system based on a research paper [25] that has found its way into the industrial world. Peppercoin shares similar design principles as Minipay, as discussed in the previous paragraphs.

2.3.3.5 M-Pesa

M-Pesa enables mobile phone-based money transfer and microfinancing. M-Pesa was launched in 2007 by the largest mobile network operators in Kenya and Tanzania and has since expanded to many other countries (such as Afghanistan, South Africa, and India).

M-Pesa allows users to withdraw, transfer, and receive funds, as well as perform purchases of goods/services. In particular, the service allows users to deposit money into an account stored on their cell phones. Payments are performed by requiring users to send (PIN-secured) SMS text messages to merchants. To redeem the received payments, these recipients are required to provide the correct PIN. M-Pesa profits from a small fee for each payment/money deposit that takes place through the service [26].

Given that M-Pesa was mainly introduced in countries where the banking network is poorly connected, the system was mainly designed to act as a branchless

banking service. Namely, M-Pesa customers can deposit and withdraw money from a network of agents that includes resellers and retail outlets that act as their banking agents.

Summary

Both M-Pesa and IBM Micropayments are systems that (functionality-wise) support micropayments. However, despite low operational costs, M-Pesa transaction fees are high when compared to the transaction value. This indicates the need in these countries for cheaper ways of micropayments. Privacy is not considered here, while it assumes that trusted hardware and software, such as mobile phone application, is in place.

2.4 SUMMARY

Table 2.1 provides a summary of the payment systems we discuss and their privacy and security guarantees. It is evident that although the theoretical background to build privacy-preserving systems exists, payment systems that have survived throughout the past few years do not support any privacy notion against either the bank or the provider of the payment service. Another remark is that in all the services provided so far (i.e., before the era of decentralized consensus networks and Bitcoin) the payment provider is always a centralized entity that has to be trusted. Micropayments are not sufficiently well supported by these systems but need to be handled by separate and dedicated payment systems.

Although digital cash proposals achieve strong privacy guarantees, these proposals rely on complex cryptographic primitives—such as zero-knowledge proofs—which are not easily understood by application developers and system integrators. This was one of the main reasons explaining the slow development of practical solutions based on digital cash.

In the following chapters, we start by describing the main operations of Bitcoin. Unlike previous electronic cash proposals, this proposal was rather straightforward, explained in a concise white paper comprising 8.5 single-column pages, and relying on basic cryptographic constructs, such as hash functions and digital signatures. The release of the proof of concept implementation of Bitcoin shortly after the dissemination of the white paper was extremely timely and important for the subsequent growth of Bitcoin. The working implementation confirmed that, unlike previous proposals, the system is clearly feasible/workable, and scales to a large

Table 2.1

Summary of Payment Systems Prior to Bitcoin and Their Classification

System	Online/ Off-line	Double-Spending Countermeasures	Need Mediator	Centralization	HW Requirements
Digital Cash	Both	Detection	No	Yes	Yes
Digital Checks	Off-line	Detection	Yes	Yes	Yes
Micropayments	Off-line	Detection	No	Yes	Yes

System	Online/ Off-line	Privacy Mechanism	Accountability	Crypto
Digital Cash	Both	Yes	Yes	PKI
Digital Checks	Off-line	w.r.t. users	Yes	PKI
Micropayments	Off-line	Yes/No	Yes	PKI

number of nodes. Open sourcing the implementation was also an excellent way for developers to maintain and support the growth of the system.

References

[1] N. Asokan, Philippe A. Janson, Michael Steiner, and Michael Waidner. The state of the art in electronic payment systems. *IEEE Computer*, 1997.

[2] Ed Grabianowski and Stephanie Crawford. How paypal works. 2014.

[3] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer-Verlag, 2001.

[4] MtGox, available from `https://mtgox.com/`.

[5] Amir Herzberg and Hilik Yochai. Minipay: Charging per click on the web. In *Selected Papers from the Sixth International Conference on World Wide Web*, 1997.

[6] Hughes N. and Lonie S. M-PESA: Mobile Money for the Unbanked: Turning Cellphones into 24-Hour Tellers in Kenya, 2007. Innovations: Technology.

[7] Markus Jakobsson and Moti Yung. Revokable and versatile electronic money (extended abstract). In *CCS '96: Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 76–87, New York, 1996. ACM.

[8] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[9] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Advances in Cryptology - EUROCRYPT 2005*, Lecture Notes in Computer Science, pages 302–321. Springer-Verlag, 2005.

[10] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings on Advances in Cryptology - CRYPTO*, 1990.

[11] S. Brands. Electronic Cash on the Internet. In *Proceedings of the Symposium on the Network and Distributed System Security*, 1995.

[12] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *Security and Cryptography for Networks*, 2006.

[13] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks – SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2002.

[14] Steven H. Low, Sanjoy Paul, and Nicholas F. Maxemchuk. Anonymous credit cards. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 108–117, New York, 1994. ACM.

[15] S. Low, N. F. Maxemchuk, and S. Paul. Anonymous credit cards and its collusion analysis. *IEEE Transactions on Networking*, December 1996.

[16] Hugo Krawczyk. Blinding of credit card numbers in the set protocol. In *FC '99: Proceedings of the Third International Conference on Financial Cryptography*, pages 17–28, London, 1999. Springer-Verlag.

[17] M. Bellare, J. Garay, R. Hauser, H. Krawczyk, A. Herzberg, G. Tsudik, E. van Herreweghen, M. Steiner, Gene Tsudik, and M. Waidner. Design, Implementation and Deployment of the iKP Secure Electronic Payment System. *IEEE Journal on Selected Areas in Communications*, 18:611–627, 2000.

[18] Elli Androulaki and Steven Bellovin. An anonymous credit card system. In *TrustBus '09: Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business*, pages 42–51, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. Par: Payment for anonymous routing. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, 2008.

[20] Ronald L. Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *Proceedings of the International Workshop on Security Protocols*, 1997.

[21] Ronald L. Rivest. Peppercoin micropayments. In *Financial Cryptography*, 2004.

[22] Yao Chen, Radu Sion, and Bogdan Carbunar. Xpay: Practical anonymous payments for tor routing and other networked services. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, 2009.

[23] Moti Yung Shouhuai Xu and Gendu Zhang. Scalable, tax evasion-free anonymous investing, 2000. available from `http://citeseer.ist.psu.edu/xu00scalable.html`.

[24] Peppercoin. Peppercoin. available from `https://en.wikipedia.org/wiki/Peppercoin`.

[25] Ronald L. Rivest. Peppercoin micropayments. In *Proceedings Financial Cryptography*, 2014.

[26] M-pesa tariffs. available from `http://www.safaricom.co.ke/personal/m-pesa/tariffs`.

Chapter 3

Bitcoin Protocol Specification

by Arthur Gervais and Ghassan Karame

In this chapter, we detail the operation of Bitcoin and summarize the main scalability measures integrated in the system.

3.1 OVERVIEW OF BITCOIN

Bitcoin operates on top of a loosely connected P2P network, where nodes can join and leave the network at will. Bitcoin nodes are connected to the overlay network over TCP/IP. Initially, peers bootstrap to the network by requesting peer address information from Domain Name System (DNS) seeds that provide a list of current Bitcoin node IP addresses. Newly connected nodes advertise peer IP addresses via Bitcoin *addr* messages. Notice that a default full Bitcoin client establishes a maximum of 125 TCP connections, of which up to 8 are outgoing TCP connections.

In Bitcoin, payments are performed by issuing transactions that transfer Bitcoin coins, referred to as BTCs in the sequel, from the payer to the payee. These entities are called "peers," and are referenced in each transaction by means of pseudonyms denoted by Bitcoin addresses. Each address maps to a unique public/private key pair; these keys are used to transfer the ownership of BTCs among addresses. A Bitcoin address is an identifier of 26 to 35 alphanumeric characters (usually beginning with either 1 or 3).

Each Bitcoin address is computed from an Elliptic Curve Digital Signature Algorithm (ECDSA) public key—for which the address owner knows the corresponding private key—using a transformation based on hash functions. Since hashes

are one-way functions, it is possible to compute an address from a public key, but it is infeasible to retrieve the public key solely from the Bitcoin address.[1] Recall that, using ECDSA signatures, a peer can sign a transaction using his or her private key; any other peer in the network can check the authenticity of this signature by verifying it using the public key of the signer.

A Bitcoin transaction is formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin. Transactions take as input the reference to an output of another transaction that spends the same coins and output the list of addresses that can collect the transferred coins. A transaction output can only be redeemed once, after which the output is no longer available to other transactions. Once ready, the transaction is signed by the user and broadcast in the P2P network. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

The difference between the input and output amounts of a transaction is collected in the form of fees by Bitcoin *miners*. Miners are peers that participate in the generation of Bitcoin blocks. These blocks are generated by solving a hash-based proof-of-work (PoW) scheme; more specifically, miners must find a nonce value that, when hashed with additional fields (e.g., the Merkle hash of all valid transactions, the hash of the previous block), the result is below a given target value. If such a nonce is found, miners then include it in a new block, thus allowing any entity to verify the PoW. Since each block links to the previously generated block, the Bitcoin *blockchain* grows upon the generation of a new block in the network. A Bitcoin block is mined on average every 10 minutes and currently awards 12.5 BTCs to the generating miner. It was shown in [2] that Bitcoin block generation follows a shifted geometric distribution with parameter 0.19. This also suggests that there is considerable variability in the generation times; for example, some blocks were generated after 99 minutes (e.g., block 152,218).

During normal operations, miners typically work on extending the longest blockchain in the network. The longest blockchain is calculated based on the chain featuring the largest number of blocks created with the largest total difficulty from the genesis block. Due to the underlying PoW scheme, different miners can potentially find different blocks nearly at the same time—in which case a fork in the blockchain occurs. Forks are inherently resolved by the Bitcoin system; the longest

1 The actual derivation of a Bitcoin address from a public key entails a series of transformations based on hashes, checksums, etc. For ease of presentation, we omit the details of the actual transformation. More detail on the construction of Bitcoin addresses can be found in [1].

blockchain that is backed by the majority of the computing power in the network will eventually prevail.

3.2 BUILDING BLOCKS AND CRYPTOGRAPHIC TOOLS

The Bitcoin protocol limits its use of cryptographic tools to cryptographic hash functions such as SHA256 and RIPEMD160, Merkle trees, and the Elliptic Curve Digital Signature Algorithm (ECDSA).

3.2.1 Cryptographic Hash Functions

Hash functions map an arbitrary long input byte sequence to a fixed size output—commonly referred to as a digest—effectively fingerprinting the input sequence. Cryptographic hash functions refer to hash functions that exhibit two essential properties: one-wayness, and collision-resistance. Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ refer to a cryptographic hash function. Informally, the one-way property implies that given $H(x)$, it is (computationally) infeasible to derive x. On the other hand, the collision resistance property implies that it is computationally infeasible to find $x \neq y$ such that $H(x) = H(y)$.

The collusion-resistance property of cryptographic hash functions constitutes important security pillars in Bitcoin. For example, the PoW in Bitcoin is mainly based on computing hashes, and the *id* of a transaction corresponds to the hash of the transaction. Hash functions are a base component of different types of data structures used in Bitcoin (e.g., Merkle trees).

3.2.2 Merkle Trees

Merkle trees allow the combination of multiple input sequences in a hash tree converging into the topmost Merkle root hash. This data structure allows the compact representation of a set of transactions, such as when the tree is built up from the transaction hashes (see Figure 3.1). Merkle trees can be used to instantiate cryptographic accumulators, which answer a query whether a given candidate belongs to a set.

A Merkle tree is a binary tree in which the data is stored in the leaves. More specifically, given a tree of height ℓ, a Merkle tree accumulates elements of a set X by assigning these to the leaf nodes (starting from position 0). Let $a_{i,j}$ denote a node in the tree located at the ith level and jth position. Here, the level refers to

the distance (in hops) to the leaf nodes; clearly, leaf nodes are located at distance 0. On the other hand, the position within a level is computed incrementally from left to right starting from position 0; for example, the leftmost node of level 1 is denoted by $a_{1,0}$. In a Merkle tree, the intermediate nodes are computed as the hash of their respective child nodes; namely $a_{i+1,j} = H(a_{i,2j}, a_{i,2j+1})$, where $H(X)$ refers to the cryptographic hash of X. Figure 3.1 depicts an example of a Merkle tree accumulating eight elements. Here, a_{30} is referred to as the Merkle root and commits to all leaf elements $U_0, \ldots, U_7$. To prove the membership of element U_3 (highlighted in Figure 3.1) in the root a_{30}, intermediate nodes a_{02}, a_{10}, and a_{21} (highlighted in ovals in Figure 3.1) are needed. We say that these nodes form the sibling path of U_3. Given n leaves, Merkle trees require $O(n)$ for constructing the tree and $O(\log(n))$ to prove membership of any element in the tree.

Formally, a Merkle tree comprises the following algorithms:

$\delta \leftarrow \texttt{Acc}(X)$**.** This algorithm accumulates the elements of a set X into a digest δ. Here, δ corresponds to the root node (i.e., $\delta = a_{\ell,0}$). This can be used to prove that the exact set X is correctly accumulated in δ.

$\pi_M \leftarrow \texttt{Prove}_M(X, x)$**.** Given a set X and element $x \in X$, this algorithm outputs a proof of membership π_M asserting that $x \in X$. π_M consists of the sibling path of x in the modified Merkle tree and the root $a_{\ell,0}$.

$\texttt{Verify}_\texttt{M}(\delta, x, \pi_M)$**.** Given δ, an element x, its sibling path and the root $a_{\ell,0}$, this algorithm outputs `true` if and only if $\delta = a_{\ell,0}$ where ℓ is the length of the sibling path and the sibling path of x matches the root $a_{\ell,0}$.

3.2.3 ECDSA

Bitcoin currently relies on the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve [3]. ECDSA is a variant of the Digital Signature Algorithm (DSA) that uses elliptic curve cryptography. The required secp256k1 private keys have a length of 256 bits and can be transformed (deterministically) into the corresponding secp256k1 public keys. Additional details on ECDSA can be found in [3].

3.3 BITCOIN DATA TYPES

In this section, we introduce the main Bitcoin specific data types.

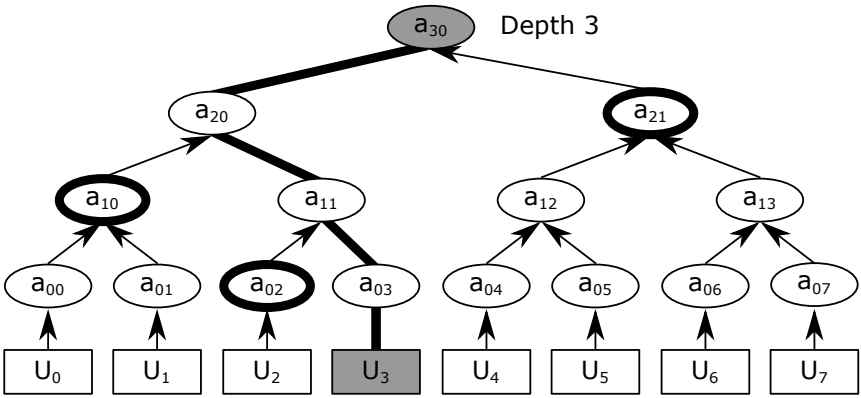

Figure 3.1 A Merkle tree of depth 3, accumulating eight elements $U_0, \ldots, U_7$.

3.3.1 Scripts

Bitcoin introduced a custom non-Turing complete scripting language in an attempt to support different types of transactions and extend the applicability of transactions beyond the simple transfer of funds. Scripts are stack-based, support a number of functions (commonly referred to as opcodes), and either evaluate to true or false. The language supports dozens of different opcodes ranging from simple comparison opcodes to cryptographic hash functions and signature verification. Since scripts are supposed to be executed on any Bitcoin node, they could be abused to conduct denial-of-service attacks and therefore, a considerable number of opcodes have been temporarily disabled. This was one of the main reasons why scripts do not provide rich support when compared to standard programming languages.

An example script program `<signature> <publicKey> OP_CHECKSIG` contains two constants (denoted by <...>) and one opcode (execution goes from the left to the right). Constants are pushed by default onto the stack, and upon execution, the stack would therefore contain `<signature> <publicKey>`. The next step is to execute `OP_CHECKSIG`, which verifies the `<signature>` under the provided `<publicKey>`. If the signature matches the provided public key, `OP_CHECKSIG` returns true, and in return, the script outputs true. Otherwise, the script will output false.

3.3.2 Addresses

An address is a unique identifier that takes the role of the originator and/or the destination of any given transaction. Technically, an address corresponds to the Base58 encoded cryptographic hash of a public key. An address's associated Bitcoin balance can be zero or positive, ranging from the smallest unit of 1 Satoshi (10^{-8} Bitcoin) to 21 million BTCs, the maximum amount of Bitcoins that can be created according to the current consensus protocol. Typically, Bitcoin addresses start with the decimal 1, or 3 (for multisignature addresses), and typically range from 26 to 35 characters. An example of a Bitcoin address is 1L78r1WnEZ73QNmQviecrnyrWrnqRhWNLy.

3.3.3 Transactions

A transaction is a data structure that can take one or more inputs and outputs (see Figure 3.2). An input redeems the BTCs that are referenced in a former transaction output. Transactions therefore effectively form a chain of transactions, and BTCs are technically only kept in transactions' outputs, not within addresses.

A transaction output specifies how many BTCs it contains as well as under which conditions a subsequent transaction can redeem the output. The subsequent transaction redeems the output of a former transaction by encoding the necessary spending information in a transaction input. The conditions under which an output can be spent are encoded with the help of scripts, and only the participants that are able to provide the correct input to the script, such that it evaluates to $\mathtt{true}$ upon execution, are allowed to spend the BTCs output by a given Bitcoin transaction.

Figure 3.2 depicts a simplified transaction with one input and two outputs. In this example, the transaction spends w BTCs to address $\mathcal{X}$ and x BTCs to address $\mathcal{Y}$. The outputs that have not yet been spent (i.e., the two outputs of transaction 2 in Figure 3.3), are commonly referred to as *unspent transaction outputs (UTXO)*.

Table 3.1 summarizes the general format of a Bitcoin transaction. Tables 3.2 and 3.3, respectively, describe the transaction input and output format.

3.3.3.1 Supported Transaction Types

Bitcoin supports a number of default transaction types. Typically, only supported transaction types are broadcasted and validated within the network. Transactions that do not match the standard transaction type are generally discarded. Note that because transactions can have multiple outputs, different output types can be combined within a single transaction.

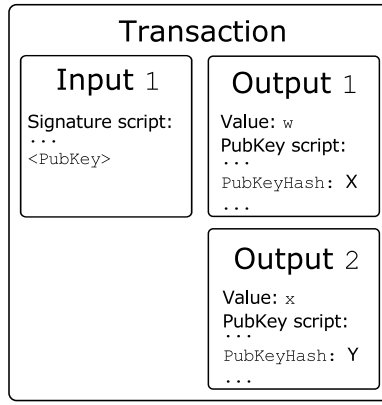

Figure 3.2 An example of a transaction with a single input spending $(w + x)$ BTCs to two output addresses ($\mathcal{X}$ and $\mathcal{Y}$).

Pay To Public Key Hash (P2PKH) A P2PKH transaction output contains the following opcodes:

`OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECK`

The corresponding input that would be eligible to spend the output specifies the required signature and the full public key as follows:

`<Sig> <PubKey>`

Pay To Script Hash (P2SH) A P2SH transaction output can only be redeemed by an input that provides a script that matches to the hash of the corresponding

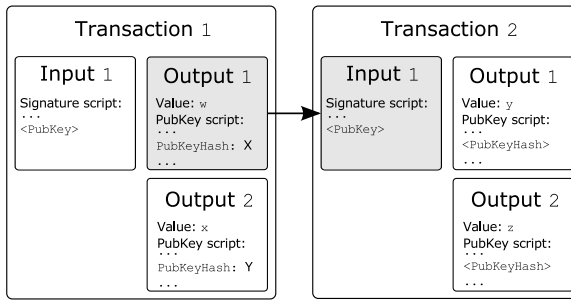

Figure 3.3 The input of transaction 2 points to the output of transaction 1.

Table 3.1

Transaction Format Within a Bitcoin Block

Field	*Description*	*Size*
Version number	Version, currently 1	4 bytes
Input counter	positive integer	1—9 bytes
List of inputs	see Table 3.2	
Output counter	positive integer	1—9 bytes
List of outputs	see Table 3.2	Variable
Locktime	Block height or time when transaction is valid	4 bytes

Table 3.2

Transaction Input Format

Field	*Description*	*Size*
Previous transaction hash	Dependency	32 bytes
Previous transaction output index	Dependency index	4 bytes
Script length		1—9 bytes
ScriptSig (Signature script)	Input script	Variable
Sequence number	0xFFFFFFFF	4 bytes

output. For example, a P2SH output contains the following transaction outputs:

```
OP_HASH160 <Hash160(redeemScript)> OP_EQUALVERIFY
```

The redeeming input consequently needs to provide a *redeemScript*, that hashes to the input's hash. Note that every standard script can be used for this purpose:

```
<sig> <redeemScript>
```

P2SH outputs are currently widely used in multisignature (multisig) transactions.

Multisig A multisignature (or commonly referred to as multisig) transaction requires multiple signatures in order to be redeemable. Multisig transaction outputs are usually denoted as m-of-n, m being the minimum number of signatures that are required for the transaction output to be redeemable, out of the n possible signatures that correspond to the public keys defined in the transaction output. An example transaction output is:

Table 3.3
Transaction Output Format

Field	*Description*	*Size*
Value	Positive integer of Satoshis to be transferred	4 bytes
Script length		1—9 bytes
ScriptSig (Pubkey script)	Output script	Variable

`<m> <A pubkey> [B pubkey] [C pubkey..] <n> OP_CHECKMULTISIG`

while the redeeming input follows this structure:

`OP_0 <A signature> [B signature] [C signature..]`

Note that P2SH allows an entity to create a transaction so that the responsibility for providing the redeem conditions is pushed from the sender to the redeemer of the funds. Consequently, the sender is not required to pay an excess in transaction fees if the redeem script happens to be complex. Multisignature transactions can be realized with either M-of-N output scripts as well as P2SH.

3.3.3.2 Script Execution

We now describe the process of script execution in Bitcoin.

In order to validate a new transaction, the input (signature script) and the output of the former transaction (pubkey script) are concatenated. Once concatenated, the script is executed according to the scripting language. During execution, constants, denoted by <..> are pushed onto the stack, and opcodes execute their respective actions by taking into account the topmost stack value.

In Figure 3.4, we depict an example of the validation of transaction 2 that spends a former output of transaction 1. The output and input script are concatenated (signature script first and then the PubKey script). In a first step, the two constants `<PubKey> <Sig>` are pushed onto the stack. Subsequently, `OP_DUP` duplicates the topmost stack value, `<PubKey>` in this case. The next opcode `OP_HASH160` hashes the `<PubKey>` and saves it as `<PubKeyHash>` on the stack. Again, a constant `<PubKeyHash>` is pushed onto the stack and `OP_EQUALVERIFY` verifies whether the two topmost stack elements are equal. If they are equal, they are removed from the stack and the last opcode `OP_CHECKSIG` verifies whether the public key on the stack (`<PubKey>`) matches the signature (`<Sig>`). If the

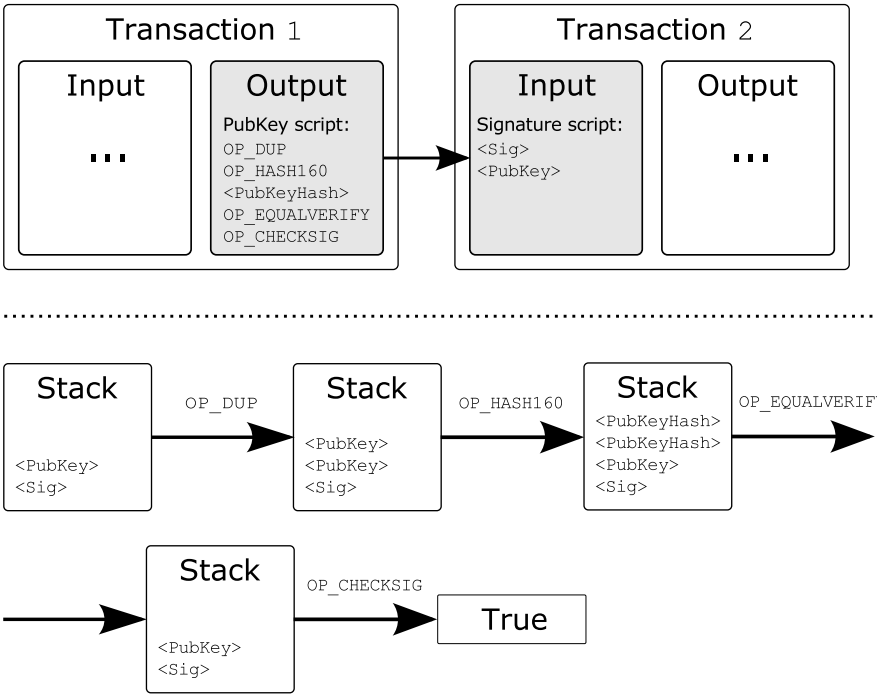

Figure 3.4 Script execution for a P2PKH transaction.

signature is valid, the script returns `true`, meaning that the input of transaction 2 is allowed to spent the output 1 of transaction 1.

3.3.3.3 Transaction Change and Fees

A transaction output is required to be spent in its entirety. Typically, however, it is unlikely that the input coins exactly match the desired output amount. Bitcoin addresses this problem by creating change outputs to where the difference between the input coins and output coins is spent. Change outputs typically correspond to new randomly generated Bitcoin addresses whose private keys are retained by the current owner of the input coins. These addresses are often referred to as *shadow* addresses.

Note that the sum of BTCs issued to all transaction outputs cannot exceed the sum of BTCs of the transaction inputs. The sum of Bitcoins of the transaction outputs, however, can be smaller than the sum of Bitcoins of the transaction inputs. This difference is paid as a fee to the Bitcoin miner that includes the transaction within a block.

3.3.3.4 Locktime

All transactions contain a field *nLockTime* that specifies the earliest time or the earliest block within which the transaction can be confirmed. Once a time-locked transaction is broadcasted in the network, miners can keep it in their list of transactions to be mined at a later stage. If the creator of the time-locked transaction changes his or her mind, they can create a new transaction that uses the same inputs (at least one overlapping input) as the time-locked transaction. The nontime-locked transaction would be confirmed immediately in a block, which would effectively make the time-locked transaction invalid.

The locktime field is 4 bytes long and is interpreted in two ways: (1) if locktime is less than 500 million, it corresponds to a block height (the highest block number of the current main blockchain), (2) if the locktime is greater to or equal than 500 million, then the locktime field is parsed as a UNIX time stamp.

3.3.4 Blocks

A Bitcoin block is a data structure containing (in a simplified view) a header, along with the list of transactions confirmed within the block. Each block header has a specific set of fields that are listed in Table 3.4. In particular, a block header contains a pointer to the former block, effectively creating a blockchain.

3.3.4.1 Blockchain

As mentioned above, each block contains a pointer to the former block; the chain of blocks therefore constructs the blockchain (see Figure 3.5). The blockchain starts with a genesis block that has been generated by the creator of Bitcoin.

The blockchain is extended by appending blocks to the last block that has been added to the blockchain. The process of creating blocks is commonly referred to as mining, and it requires providing a proof-of-work that we detail in the next paragraphs.

Because different miners perform mining concurrently, it can happen that competing blocks are created at the same block height. This event is commonly

Table 3.4
Bitcoin Block Header Format

Field	*Description*	*Size*
Version	Block version number	4 bytes
Hash of previous block	Hash of previous block header	32 bytes
Merkle root hash	Transaction Merkle root hash	32 bytes
Time	Unix time stamp	4 bytes
nBits	Current difficulty of the network	4 bytes
Nonce	Allows miners to search a block	4 bytes

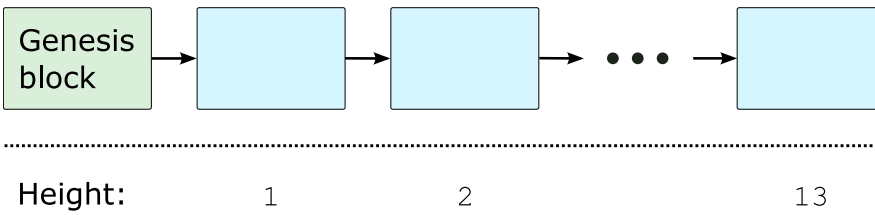

Figure 3.5 Chain of blocks starting from the genesis block forms the blockchain.

referred to as forking and is depicted in Figure 3.6. Eventually, only one blockchain, the longest, can prevail. Blocks that are not part of the longest chain are therefore discarded and are referred to as orphan blocks.

3.3.4.2 Proof-of-Work

In order to achieve consensus among peers in the Bitcoin network, Bitcoin relies on the synchronous communication assumption along with a hash-based PoW concept. Here, peers have to prove that they have expended a certain amount of computations; peers that perform the proof-of-work are commonly referred to as miners. The more hashes a miner can perform, the more likely the miner is able to find a block, thus the ability to find blocks is proportional to the miner's hashing power. Bitcoin's particular proof-of-work mechanism is to require the double SHA256-hash of the block header content. The difficulty of the mining process is adjusted dynamically in order to meet an average block generation time of 10 minutes.

More specifically, to generate a block, miners must find a nonce value that, when hashed with additional fields (i.e., the Merkle hash of all valid and received

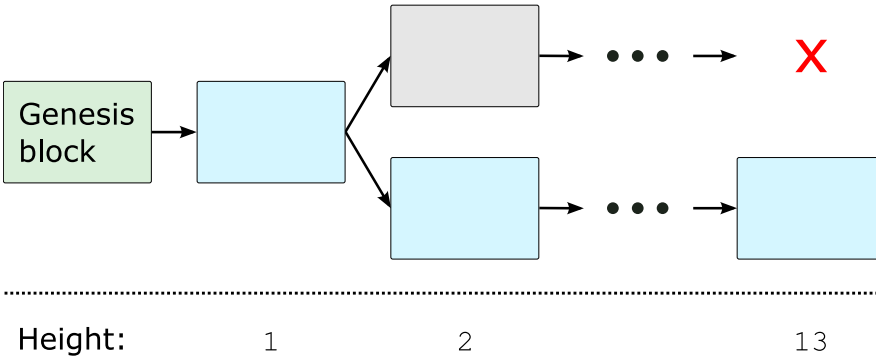

Figure 3.6 An example of a fork in the blockchain; gray blocks are called orphan blocks.

Table 3.5
Example of The Header of Block in 364082 Bitcoin

Hash: 00000000000000000bcd79fd8739a43205f4286e68a4d7bd3a83bcb0c7158d99
Previous block: 00000000000000000009a4f7f94f2e7fc81e64182b0e2540b3cc91c89076f3da5b
Time: 2015-07-06 08:39:20
Difficulty: 49,402,014,931.23
Transactions: 436
Total BTC: 1,081.04970944 BTC
Size: 244.1259765625 KB
Merkle root: 2ade89c464e2f46e393a292e474d391f6055d8f19486a98930775b8926f43934
Nonce: 1386491545

transactions, the hash of the previous block, and a time stamp), the result is below a given target value. If such a nonce is found, miners then include it (as well as the additional fields) in a new block, thus allowing any entity to verify the PoW. Upon successfully generating a block, a miner is granted a number of BTCs (currently 12.5 BTCs). These BTCs are awarded by means of a coinbase transaction that transfers the generated BTCs onto a newly created Bitcoin address controlled by the miner. These mechanisms offer a strong incentive for miners to commit their computational resources in order to support the Bitcoin system and continuously confirm transactions. An example of a Bitcoin block is shown in Tables 3.5 and 3.6.

Note that Bitcoin adopts a limited supply strategy. That is, Bitcoin defines the rate at which currency will be generated. For example, in 2009, each miner

Table 3.6
Block 364082 in the Main Blockchain.[2]

Tx Hash	69f45d539f9d2744116c43a4fd157d54b11f092248db2cfe0db36baccd6d3fe5
Source & Amount	Generation
Recipient & Amount	1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY: 25.06175045 BTC
Tx Hash	...
Source & Amount	...
Recipient & Amount	...
Tx Hash	3ad5d3f813168ac2246c3e80ff6d9279023c30a661a41e1fa522e82dce608d03
Source & Amount	16C1bgMsxPyfJVDPkv367ajXjgpjkiUxUA:0.23689801 BTC
Recipient & Amount	1Gz9aQkk61r5VSD1WvnoyVgSfFafcziD8N:0.23689801 BTC
Tx Hash	69a9421b77e2ec609b98adadd29da98dc1fa4d16e2fc75acd6637ddf7bbc069a
Source & Amount	1FxddVRcF7tttvwc1cyaLUUeosXSoiMVFE:0.74550674 BTC
Recipient & Amount	1CdV9rovEYUJkGEkejWY5MbmqPSTy1E4Rk :0.74550674 BTC
Tx Hash	...
Source & Amount	...
Recipient & Amount	...

was awarded 50 new BTCs upon generating a Bitcoin block. This amount is halved approximately every 4 years until the generation of BTCs in the system depletes.

Once a block is generated, it is broadcast in the entire network. Any entity that receives the block can verify the correctness of the PoW by computing the hash over the announced block fields, checking the correctness of the transactions included within, and verifying that it is below the target difficulty. Note that the Bitcoin network has a global block difficulty, which is updated every 2016 blocks. In essence, the difficulty is adjusted depending on the generation time of the last 2016 blocks in the network. That is, if the last 2016 blocks took more than 14 days of time to compute (i.e., more than 10 minutes on average), then the difficulty is reduced. Otherwise, if they took less than 14 days of computation, then the network difficulty is increased.

To generate a block, miners *work* on constructing a PoW. In particular, given the set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin miners need to find a nonce such that:

$$\texttt{SHAd256}\{\mathsf{Bl}_l \,||\, \mathrm{MR}(\mathrm{TR}_1, \ldots, \mathrm{TR}_\mathrm{n}) \,||\, \mathrm{No}\} \leq \mathit{target}, \tag{3.1}$$

2 The block contains a total of 436 transactions; here, we only show 3 transactions confirmed in the block.

where SHAd256 is the SHA−256 algorithm applied twice, Bl_l denotes the last generated block, $\mathrm{MR}(\overline{x})$ denotes the root of the Merkle tree with elements $\overline{x}$, $\mathrm{TR}_1 \,||\, \ldots \,||\, \mathrm{TR}_n$ is a set of transactions that have been chosen by the miners to be included in the block,[3] No is the 32-bit nonce, and *target* is a 256-bit number. To generate the PoW, each miner chooses a particular subset of the candidate solutions' space and performs a brute-force search. It is apparent that the bigger the *target* is, the easier it is to find a nonce that satisfies the PoW.

The resulting block is forwarded to all peers in the network who can then check its correctness by verifying the hash computation. If the block is deemed to be valid,[4] then the peers append it to their previously accepted blocks. Since each block links to the previously generated block, the Bitcoin blockchain grows upon the generation of a new block in the network. As mentioned earlier, when miners do not share the same view in the network (e.g., due to network partitioning), they might work on different blockchains, thus resulting in forks in the blockchain. Block forks are inherently resolved by the Bitcoin system; the longest blockchain will eventually prevail. Transactions that do not appear in blocks that are part of the main blockchain (i.e., the longest) will be readded to the pool of transactions in the system and reconfirmed in subsequent blocks.

Currently, in the Bitcoin system, a transaction can be redeemed by the payee if it has received at least six confirmations; that is, there are five new blocks that build on the block which confirms it. This mechanism offers an inherent protection against double-spending attacks since it is computationally infeasible for an adversary to change the history of a transaction that has been confirmed by six blocks in the system. This process is exemplified in Figure 3.7.

3.4 BITCOIN ARCHITECTURE

The Bitcoin ecosystem emerged over the need to provide services to different nodes depending on their available resources. We describe in the following the different node types in Bitcoin and how they interoperate in the network.

3 These transactions are chosen from the transactions that have been announced (and not yet confirmed) since Bl_l's generation.

4 That is, the block contains correctly formed transactions that have not been previously spent and have a correct PoW.

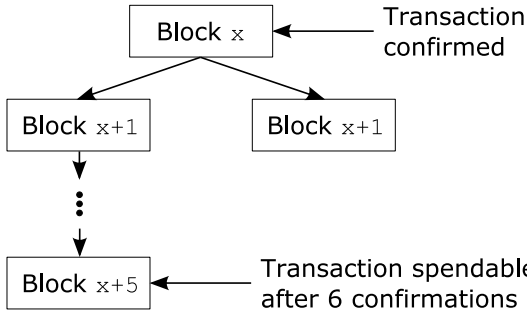

Figure 3.7 Confirming transactions in Bitcoin.

3.4.1 Node Types

Due to the heterogeneity of nodes in the network in the Bitcoin ecosystem, multiple nodes types are supported in the system.

3.4.1.1 Miner

Miners perform the proof-of-work in order to find and broadcast blocks in the Bitcoin network. Their operations consists mainly of quickly retrieving information about the newest blocks and validating transactions that are included in new blocks. Miners typically operate dedicated mining hardware to perform as many hash operations as possible, and can use dedicated communication links to efficiently spread found blocks to the whole network. Note that the term "mining" originates from the traditional process of acquiring scarce/precious material (e.g., gold)—hence the analogy to Bitcoin mining.

As mentioned earlier, every discovered Bitcoin block provides a monetary reward to the miner. Miners typically organize themselves in groups, commonly referred to as mining pools. Because of its higher collective hashing power, a mining pool has a higher probability to find a block, and mining pool members can consequently receive larger payouts than individual miner. We discuss the impact of mining on the security of Bitcoin in Chapter 4.

3.4.1.2 Full Node

We define a full Bitcoin node as a node that (1) maintains the full copy of the blockchain, (2) validates all incoming transactions and blocks, and (3) forwards transactions and blocks to its peers. In addition to providing validation services to the Bitcoin network, a full node might provide an open TCP port (Bitcoin uses the TCP port 8333) to where other Bitcoin peers connect. Throughout the rest of this book, we use the terms "full node" and "regular node" interchangeably.

3.4.1.3 Lightweight Clients

Lightweight clients are clients that do not store, nor maintain the full Bitcoin blockchain, but follow a simple payment verification (SPV) scheme. This latter scheme allows the lightweight client to verify that a transaction has been included in the blockchain by receiving and verifying only the block headers. In addition, lightweight clients receive only transactions that are relevant to their wallets and do not need to perform transaction or block validation. As a result, lightweight clients require significantly less resources to operate than full nodes or miners. In Chapter 6, we discuss the operation of lightweight clients and detail the SPV mode.

3.4.2 Peer-to-Peer Overlay Network

Bitcoin's Peer-to-Peer (P2P) overlay network enables peer discovery and provides broadcast information propagation. The network is sparsely connected by default. A fraction of the Bitcoin peers (e.g., full nodes) provide incoming TCP connections on port 8333 (typically full Bitcoin nodes) to which other peers can connect. SPV clients in particular do not provide direct services to the P2P overlay network and only receive information relevant to their addresses.

3.4.2.1 Peer Discovery

Upon initialization, a Bitcoin node is typically not aware of other full nodes' IP addresses. The client therefore queries DNS servers (called DNS seeds) which provide, for bootstrap purposes, a list of IP addresses of full Bitcoin peers. The DNS seeds are maintained by Bitcoin community members: some provide dynamic DNS seed servers that automatically acquire IP addresses of active nodes by scanning the network; others provide static seeds that are updated manually.

Once a peer has received a list of full Bitcoin node IP addresses, the peer performs up to eight outgoing connection attempts; the eight nodes that the peer

attempts connection with are referred to as entry nodes. In order to diversify the number of peers that a node is aware of, a node can request from its neighbors the IP addresses of other peers that they are aware of using the *addr* P2P network message. Note that a peer replaces dynamically any dropped or failed entry connections. There is otherwise no default connection renewal foreseen by the peers until the node resets. A Bitcoin node can connect to a default number of 125 TCP connections. Note that nodes can change this default number in their clients (and recompile the code) in order to establish more than 125 concurrent TCP connections.

Though the Bitcoin daemon does not explicitly distinguish between clients and servers, Bitcoin peers can be grouped into those which can accept incoming connections (servers) and those which cannot (clients), such as peers behind NAT or firewall.

The Bitcoin protocol implements an IP address propagation mechanism to help peers discover other peers in the P2P network. Each Bitcoin peer maintains a list of IP addresses of other peers in the network and each address is given a time stamp that determines its freshness. Peers can request IP addresses from this list from each other (through *getaddr* messages) and/or advertise to the network IPs known to it (through *addr* messages). Upon receiving an *addr* message from the network, a peer checks if it is well formed (i.e., contains less than 11 IPs), and parses the IPs listed within to decide whether to forward it to its neighbors.[5] More specifically, the peers check if each of the addresses advertised are fresh (using the time stamp field), and if so, they forward it to two neighbors of their choice.[6] Note that by limiting the number of neighbors to which an address is forwarded, the overall messages exchanged among peers in the Bitcoin P2P network is inherently reduced.

Assuming a reachable address, Bitcoin peers choose a subset of their neighbors (also known as responsible nodes) that will receive the address advertisement. This selection is performed by computing the hash of the forwarding address concatenated with a secret salt, current date, and the memory address of the data structure describing the neighbor. Only two neighbors (the first two) are selected in this process.

Once the neighbors are selected, the transmission of the scheduled messages (i.e., the address in this case) is performed in rounds lapsing 100 milliseconds each. In each round, the corresponding *addr* messages are pushed to the randomly

5 The current reference implementation of Bitcoin nodes recognizes three types of addresses: IPv4, IPv6, and OnionCat addresses.

6 Here, we assume that the receiving peer is a reachable address.

selected neighbors. This process of pushing *addr* messages is also known as *trickling*, and the chosen node as *trickle* node.

When a peer receives a *getaddr* message, it sends back a bounded number of addresses that it is storing. The number of addresses sent is set to around 23% of the total number of stored addresses without exceeding the threshold of 2500 addresses. To avoid flooding the network with unnecessary messages, Bitcoin peers maintain a history of the addresses that have been announced for each neighbor. Thus, a peer would only advertise once the same address over a connection. Note that this history is per connection (and not per IP), and is cleared every 24 hours. In addition, there is an upper bound of the total number of addresses that a peer can locally store (currently around 20K); old addresses are replaced by newly received ones whenever this limit is reached.

For each received address, the peer assigns a score as follows: the local interfaces initially are assigned with low scores (i.e., 1), while external IP addresses receive high scores (i.e., 4).

3.4.2.2 Peer Connection

Given an IP address of a Bitcoin node, a peer attempts to connect via a standard TCP connection on TCP port 8333. Once the TCP connection is established following the basic three-way TCP handshake, the connecting peers exchange *version* messages that contain (1) the client version number, (2) current block height, and (3) the current time. Both nodes acknowledge the *version* messages with a *verack* message in order to confirm that the session has been established.

Note that *version* messages advertise those addresses that are locally assigned with the highest scores.

3.4.2.3 Block Synchronisation

The *genesis block* (block 0) is the first block of the Bitcoin blockchain and is hardcoded in each Bitcoin node. Subsequent blocks are synchronized on the go; if a node has been off-line during a substantial amount of time (typically 24 hours), the block synchronization allows the node to catch up to the current tip of the longest blockchain. Recall that only peers that are tightly synchronised with the current blockchain can perform transaction and block validation.

The current Bitcoin implementation (version 0.10 upward) employs a block synchronization with *headers-first*. This process allows nodes to download the headers of all possible chains, determine the best chain based on the length and

the difficulty of the block headers, and subsequently to download in parallel the actual block content once the main blockchain has been determined.

3.4.2.4 Dedicated Relay Networks

As discussed earlier, miners increase their advantage in the network by (1) receiving new blocks as fast as possible and (2) broadcasting the mined blocks as fast as possible to the majority of the other miners. Clearly, the time to download and process new blocks is of crucial importance, since miners risk losing their profits when, for example, mining on outdated blocks.

Since the Bitcoin P2P overlay network is a general-purpose broadcast mechanism connecting full Bitcoin nodes, SPV clients, and miners, this network is not necessarily optimized to efficiently circulate blocks across miners. This motivated the rise of a number of private peering network and alternative relay networks [4] whose sole goal is to connect the miners using high-speed connections in order to ensure a fast broadcast of blocks to miners. These networks [4] aim to provide transaction and block information faster than the official Bitcoin P2P network (e.g., with latencies in the order of 100 - 3000 milliseconds).

Matt Corallo's relay network is one of the most prominent instantiation of an alternative relay network, currently operating five relay nodes—each serving between 20 and 40 clients. Corallo's network performs partial object validation and does not follow the request management system of Bitcoin to ensure a faster spread of information in those networks. *Given that relay networks are operated by a single entity and only support a limited number of nodes, this allows the operator of these relay networks to control the information fed to the biggest mining pools—and thus to control the entire Bitcoin network.*

Note, however, that one can envision alternative relay networks built with different trust models and can incorporate arbitrary policies to counter such centralization; for instance, one can construct a small and trusted decentralized relay network only comprising of representative nodes from each centralized mining pool.

3.4.2.5 Alert Mechanism

Alert messages were introduced in the Bitcoin client after version 0.3.10. These messages serve to alert the Bitcoin users in case of critical incidents. For example, if a severe vulnerability is found in the Bitcoin client, Bitcoin developers can issue an alert message that will be displayed to the user.

Since alert messages are cryptographically signed, they can only be sent by people that possess the appropriate cryptographic key. Currently, this key is shared among the Bitcoin developers. *This gives these entities privileged powers to reach out to users and urge them to adopt a given Bitcoin release.* We point out that the current alert payload format supports a RESERVED string that is not currently being used. It is straightforward to see that this field can be abused in the future to send additional control commands (i.e., Botnet-like commands) to be executed by Bitcoin users [5].

3.5 SCALABILITY MEASURES IN BITCOIN

At the time of writing, almost one transaction per second (tps) [6] is executed in Bitcoin; this results in an average block size of almost 400 KB. The maximum block size is currently limited to 1 MB, which corresponds to less than seven transactions per second. Given the increasing adoption of Bitcoin, the number of transactions, and the block sizes are only expected to increase. For example, if Bitcoin were to handle 1% of the transactional volume of Visa,[7] then Bitcoin needs to scale to accommodate almost 500 tps—requiring a large amount of information to be broadcasted in the network.

Motivated by these factors, the current Bitcoin protocol implements various bandwidth optimizations and measures in order to sustain its scalability and correct operation in spite of ever-increasing use. In what follows, we detail the existing measures taken by Bitcoin developers.

3.5.1 Request Management System

Bitcoin uses an advertisement-based request management system to minimize the information spread in the network.

Namely, to minimize information spread in the network, messages are propagated in the Bitcoin network with the help of an advertisement-based request management system. More specifically, if node $\mathcal{A}$ receives information about a new Bitcoin object (e.g., a transaction or a block) from another node, $\mathcal{A}$ will advertise this object to its other connections (e.g., node $\mathcal{V}$) by sending them an *inv* message. These messages are much smaller in size than the actual objects, because they only contain the hash and the type of object that is advertised. Only if node $\mathcal{V}$ has not previously received the object advertised by the *inv* message, will $\mathcal{V}$ request the

7 Currently, the Visa network is designed to handle peak volumes of 47,000 tps [7].

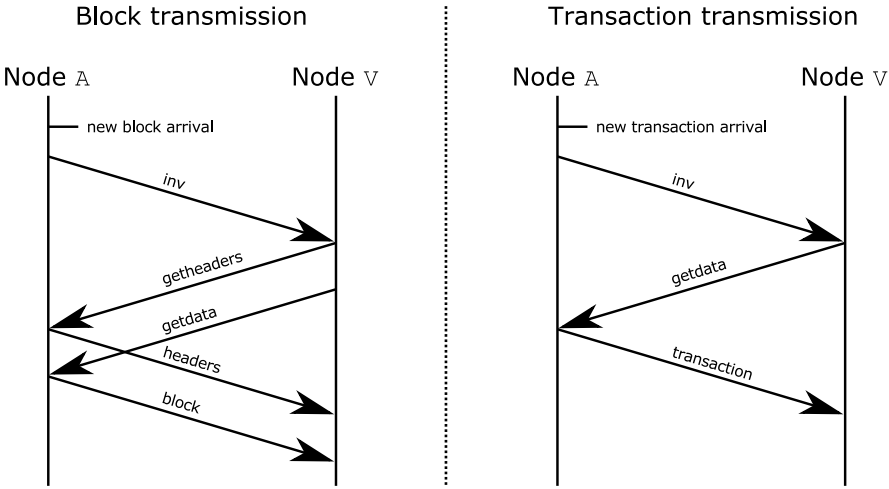

Figure 3.8 Propagation mechanism for blocks and transactions.

object from $\mathcal{A}$ with a *getdata* request. Following the Bitcoin protocol, node $\mathcal{A}$ will subsequently respond with a Bitcoin object (e.g., the contents of a transaction or a block).

By doing so, inventory messages limit the amount of data broadcast in the network. Note that in case the object advertised corresponds to a block, neighbor $\mathcal{V}$ first requests the block header before the actual block data. Here, when a block header is advertised via a *headers* message, the receiving node internally stores the highest block known by the sending peer. The receiving node also validates any received header by verifying its corresponding PoW. Transactions, on the other hand, are propagated directly following the reception of the corresponding transaction *inv* message. This process is summarized in Figure 3.8.

To minimize bandwidth consumption, Bitcoin nodes request a given object only from a single peer, typically the first peer that first advertises the object. Requesting the same object from multiple peers entails downloading the same data several times, and therefore can only increase the bandwidth consumption of the system.

As in the case of address discovery protocols, when a client generates a transaction, the client schedules it for forwarding to all of its neighbors. In particular, the client computes a hash of a value composed of the transaction hash and a secret salt. If the computed hash has the last two bits set to zero, the transaction is forwarded

immediately to all the eight entry nodes. Otherwise, the transaction is queued for announcement as described before for *addr* messages: the neighbor receives the transaction whenever it is selected as a trickle node. To avoid flooding the network with unnecessary messages and messages similar to *addr* messages, a Bitcoin peer maintains history of all forwarded transactions for each connection. If a transaction was already sent over a connection, the transaction will not be resent another time. In addition, a Bitcoin peer keeps all received transactions in a memory pool, such that if the peer received a transaction with the same hash as one in the pool or in a block in the main blockchain, the received transaction will be ignored.

3.5.2 Static Time-outs

Bitcoin relies on static time-outs in order to prevent blocking while tolerating network outages, congestion, and slow connections. Blocking can occur, for example, when a node stops responding during communication.

Given that Bitcoin runs atop an overlay network, communication latency and reliability pose a major challenge to the correct operation of the system.

For example, in Bitcoin version 0.10, the Bitcoin developers introduced a block download time-out of 20 minutes.[8]

Similarly, for transactions, the Bitcoin developers introduced a 2-minute time-out. Note that the choice of the time-out is a nontrivial task and depends on a number of parameters such as bandwidth, object size, latency, processing power, and the Bitcoin version of each node. On the one hand, overly long time-outs might deteriorate the quality of service of the whole network and can be abused to conduct, for example, double-spending attacks [8]. On the other hand, short time-outs might hinder effective communication under varying network conditions or when communicating with slow peers.

3.5.3 Recording Transaction Advertisements

Bitcoin clients keep track of the order of the received transaction advertisements. If a request for a given transaction is not delivered, the next peer in the list is queried.

When a transaction T is advertised via an *inv* message to a given node, the latter keeps track of the order of announcements with a first-in first-out (FIFO) buffer. Each time a peer advertises T, the peer is inserted into the buffer. Transaction T is only requested from one peer at a time. For each entry in the buffer, Bitcoin

8 Available from `https://github.com/bitcoin/bitcoin/pull/5608`.

clients maintain a 2-minute time-out, after which the next entry in the buffer is queried for T.

3.5.4 Internal Reputation Management System

Bitcoin combats the broadcasting of ill-formed blocks and transactions by maintaining an internal reputation management system. Namely, whenever a node receives objects (e.g., blocks, transactions), it checks their correctness before forwarding them to other peers in the network. First, objects are validated based on their respective syntax and size (e.g., oversized objects are discarded). If this verification passes, the contents of the objects are subsequently validated. For transactions, this includes verifying the signature and the input and output coins used in the transaction; similarly, the PoW included in block headers is verified with respect to the current network difficulty.

To prevent any abuse of the Bitcoin overlay network (e.g., denial-of-service attacks), a receiving node locally assigns a penalty to peers who broadcast ill-formed objects. Once a node has reached 100 penalty points, the receiving node disconnects from the misbehaving node for 24 hours. For example, if a node broadcasts invalid alerts, then it will be given 10 penalty points. Nodes that attempt more serious misbehavior, such as inserting invalid transaction signatures, are immediately assigned 100 points, and therefore directly banned. Penalties also apply to ill-formed control messages such as *inv* (inventory) or *addr* commands. Note that locally assigned penalties are not transmitted to other peers.

References

[1] Technical background of version 1 Bitcoin addresses, 2013. available from `https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses`.

[2] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917, New York, 2012. ACM.

[3] D Brown. Sec 2: Recommended elliptic curve domain parameters, 2010. available from `http://www.secg.org/sec2-v2.pdf`.

[4] Matt Corallo. Bitcoin relay network. available from `http://bitcoinrelaynetwork.org/`.

[5] Bitcointalk Forum, available from `https://bitcointalk.org/`.

[6] Bitcoin Wiki, available from `https://en.bitcoin.it/wiki/`.

[7] Stress Test Prepares VisaNet for the Most Wonderful Time of the Year, 2015. available from http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html.

[8] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1):2:1–2:32, May 2015.

Chapter 4

Security of Transactions in Bitcoin

In this chapter, we review and analyze transaction security in the Bitcoin protocol. Here, we distinguish between the security of standard payments (dubbed confirmed transactions) and of fast payments, where the exchange between the payment and the service is short, in the order of a few tens of seconds.

4.1 SECURITY OF CONFIRMED TRANSACTIONS

As mentioned in the previous chapter, transactions are constructed in Bitcoin as follows. The payer chooses the coins that he or she will pay with (the number of coins depends on the payment amount and the coin values owned by the payer), and includes the hashes of the previous transactions where each of the chosen coins was spent as inputs to the transactions. The public key(s) (hence referencing the address(es) of the payee) are then used as outputs of the transaction. Note that the official Bitcoin client has started to support transactions with multiple recipients since December 16, 2010. As shown in Figure 4.1, all inputs and outputs of the transaction are then signed using the private key of the payer; the resulting transaction (including the signature) is then broadcasted in the P2P network.

When the payee receives the transaction, he or she checks the signatures and verifies the correctness of the transaction (see Section 4.1.1). If these verifications are successful, the payee awaits that the network confirms his or her transaction before redeeming the received coins. Transactions are confirmed in Bitcoin using blocks. Bitcoin blocks are computed by miners—peers that "mine" for Bitcoins—and implement a hash-based proof-of-work (PoW) concept. Miners verify the correctness of each transaction they receive from the network and subsequently

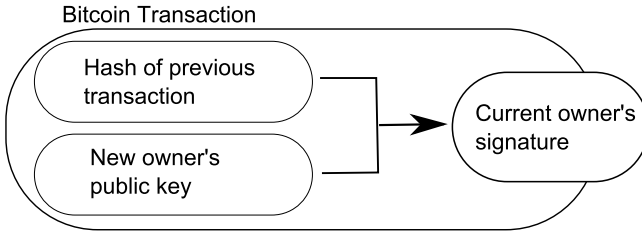

Figure 4.1 A Bitcoin transaction with one input and one output.

include those correct transactions in their newly generated block. Since blocks implement a PoW, transactions that are included in a block are hard to revert; if six Bitcoin blocks build on a block including the payee's transaction, then the payee can redeem the coins received from the payer.

Bitcoin relies on the synchronous communication assumption along with the hash-based PoW in order to ensure the security of transactions in the system. In what follows, we discuss both of these concepts in greater detail.

4.1.1 Transaction Verification

Since each payment references the last transactions where each of the coins has been spent, coin expenditure can be easily traced in the network, as shown in Figure 4.2. Moreover, since all transactions are broadcasted in the entire network, all peers in the network can verify their correctness.

Namely, whenever a peer in the network (including the payee) receives a transaction, it checks its signature, format, the correctness of its fields, and that the sum of the coins referenced by the inputs matches that of the outputs (and the fees[1]). Additionally:

- Each peer verifies that the input coins have not been spent earlier by checking against the history of all executed transactions in the network
- Each peer verifies that the input coins refer to correct transactions that have already been confirmed in the network.

By doing so, Bitcoin prevents the double-spending of coins in the system. This is mainly achieved since the details and order of all transactions are publicly

1 As described later, fees are collected by miners who include the transaction in their newly generated block.

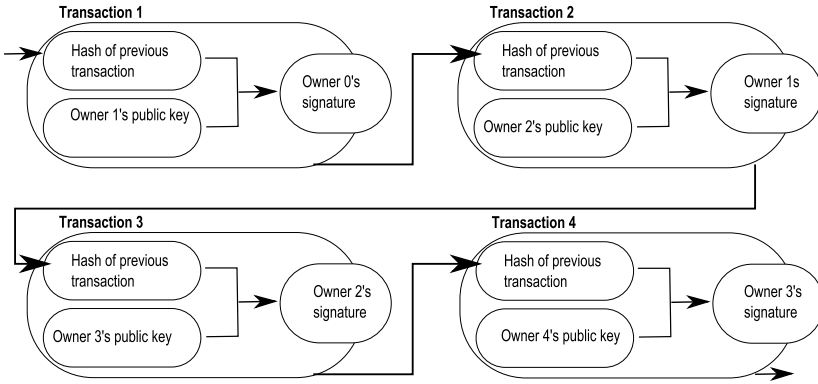

Figure 4.2 Coin expenditure in the Bitcoin network. Here, we show an example comprising of four transactions spending a coin from Owner 0 to Owner 1, Owner 2, Owner 3, and Owner 4.

announced in the system, and since the communication in Bitcoin is synchronous. All verified transactions are included temporarily in the peer's memory pool until they are confirmed by the network (see Section 4.1.4).

A study in [1] has shown that transactions are propagated in the network in few seconds. Recently, several attacks [2, 3] were reported on the delivery of transactions and blocks in Bitcoin. In what follows, we discuss these attacks in greater detail.

4.1.2 Eclipse Attacks in Bitcoin

Recently, Heilman et al. showed how to attack the Bitcoin mining protocol by monopolizing the connections of nodes in the system [2]. For example, the attacker could possess a large number of IP addresses at his or her disposal and controls a large number of machines (e.g., a botnet). Alternatively, the adversary might be an Internet service provider (ISP) or a nation state adversary. On the other hand, the victim node is assumed to have a public IP address; that is, the victim is not located behind network address translators (NATs). Finally, the attack requires the ability of the adversary to cause a restart of the victim's Bitcoin client (e.g., by means of distributed denial-of-service attacks or when the victim upgrades his or her Bitcoin client).

The intuition behind eclipse attacks is straightforward. Eclipsing entails blinding the view of the victim from the blockchain and requires that the adversary is able

to isolate the victim from the rest of the network by monopolizing all of the victim's outgoing and incoming connections.

In [2], this is achieved by exploiting the way in which Bitcoin clients store the IP addresses that are advertised in the network. Namely, in Bitcoin, peers exchange *addr* messages that contain IP addresses and their time stamps. These messages are used by nodes to obtain network information from peers. Public IPs are stored at each node in two tables: *tried* and *new* tables. The *tried* table consists of 64 buckets; each can store 64 unique addresses from peers with whom the node has established communication before. The node also keeps the time stamps of each tried IP address.

When the node connects to a new peer, his or her address and the time stamp are added to the *tried* bucket. If the bucket is full, the new address is inserted at a random location in the bucket and replaces the address that was stored there. Similarly, the *new* table contains 256 buckets, each containing up to 64 addresses.

Since each Bitcoin node can have up to eight outgoing connections, and 117 incoming connections, the node selects the IP addresses to connect to from *tried* with probability ρ. Otherwise, the node selects with probability $1 - \rho$ the address to connect to from the *new* table; note that node selection is biased to IP addresses that have a recent time stamp. Therefore, for the adversary to monopolize the connections of his victim, the adversary has to populate the *tried* table with addresses that are under their control. Heilman et al. suggest populating the *new* table with bogus IP addresses (e.g., nonexisting IP addresses). Later on, when the victim restarts, then it will attempt to connect with addresses from the *tried* and *new* tables; all of the addresses that the victim will connect to are guaranteed then to be under the control of the adversary. Real experiments in the Bitcoin network show that eclipse attacks succeed with a probability of 84%.

4.1.2.1 Implications

The aforementioned attack can have serious implications on the Bitcoin network.

Implication 1: The adversary can increase its advantage in selfish mining (see Section 4.1.4.1), by splitting the mining power of the honest nodes.

Implication 2: The adversary can double-spend transactions even if these transactions are confirmed by six consecutive blocks. For example, the victim can be a merchant and the adversary can simply pay him, eclipse the miners working on confirming this transaction, and then issue a double-spending transaction to uneclipsed miners. Since the blocks performed by eclipsed miners will be eventually obsolete, this attack is likely to succeed.

4.1.2.2 Countermeasures

Heilman et al. suggest a number of countermeasures to thwart this attack:

Countermeasure 1: One possible hardening technique is to ensure that the same address hashes to the same bucket and the same location in the *tried* table. By doing so, one can prevent the adversary from reusing the same address more than once to fill the *tried* table.

Countermeasure 2: Another countermeasure would be to simply avoid any bias in choosing addresses that are recent. Currently, there is a bias in choosing recently time-stamped addresses, which will increase the probability to connect to the adversary's addresses.

Countermeasure 3: Another basic countermeasure would be to ensure that an IP address exists (e.g., by attempting to ping/connect to it) before overwriting an older address in the *tried* and *new*.

Countermeasure 4: One possible countermeasure would be to simply add new buckets, which will harden the realization of such attacks.

Countermeasures 1, 2, and 4 have been integrated in the official Bitcoin client v0.10.1.

Note that in [2], the adversary needs to have almost 5120 IP addresses at his or her disposal to eclipse a victim. Moreover, the adversary would need clients to restart. Recently, Gervais et al. [4] have however shown that even resource-constrained adversaries can perform similar eclipse attacks without requiring any node restart. Namely, the authors show that the adversary can abuse existing scalability measures adopted in Bitcoin in order to deny information about transactions to Bitcoin nodes for a considerable amount of time. In what follows, we sketch out this attack and outline a number of countermeasures.

4.1.3 Denying the Delivery of Transactions

To minimize information spread in the network, transactions are propagated in the Bitcoin network using an advertisement-based request management system (see Chapter 3). Namely, if a peer receives information about a new transaction from another node, then the node will advertise this transaction to its other connections by sending them an *inv* message (see Figure 4.3). This message is much smaller in size than the actual transaction, because it only contains the hash and the type of object that is advertised.

If the neighbor has not previously received the transaction advertised by the *inv* message, he or she will request the transaction using a *getdata* request. Note that

Bitcoin clients keep track of the order of the received transaction advertisements. If a request for a given transaction is not delivered within 2 minutes, the next peer in the list is queried [4].

By doing so, inventory messages limit the amount of data broadcast in the network. However, this process also opens the door for an adversary to considerably delay the delivery of transactions in the network, and therefore to severely affect the ability of the network to verify transactions on time.

This is achieved as follows [4]. The adversary simply sends n back to back *inv* messages for the same transaction T. Note that it is important that the adversary is the first to announce T to the victim, otherwise the latter will request it from another peer. When the victim sends a *getdata* request, the adversary does not reply. After 2 minutes, the time-out will trigger, and the victim will request T from the next node on the list—which also corresponds to the adversary's. Note that the victim will not request T from any other advertiser in the meantime. By doing so, the adversary effectively increases the time-out specific to the advertised transaction by $2n$ minutes, as shown in [4].

This attack has considerable impact on the security of Bitcoin, since it deprives peers from the benefit of immediately receiving and verifying transactions. As we show in Section 4.2.1, this attack can considerably facilitate double-spending in the network.

4.1.3.1 Possible Countermeasures

Little can be done to thwart this attack. For instance, even if nodes limit the number of connections they accept (since the attack requires a direct connection to the victim), it is hard for nodes to ensure that all their current connections are trustworthy.

Moreover, nodes can try to filter the received *inv* messages by IP or can randomly (instead of sequentially) query the next peer after a time-out has occurred. However, an adversary that possesses several nodes at his or her disposal can easily thwart these countermeasures and flood the victim with *inv* messages corresponding to the desired transaction from a large number of nodes. Even if they randomly select the peer to query from the advertiser's list, then the probability of consistently selecting the adversary can be considerable, depending on the number of nodes controlled by the adversary.

This shows the limits of synchrony in Bitcoin and motivates the need for a redesign of Bitcoin's object request management system.

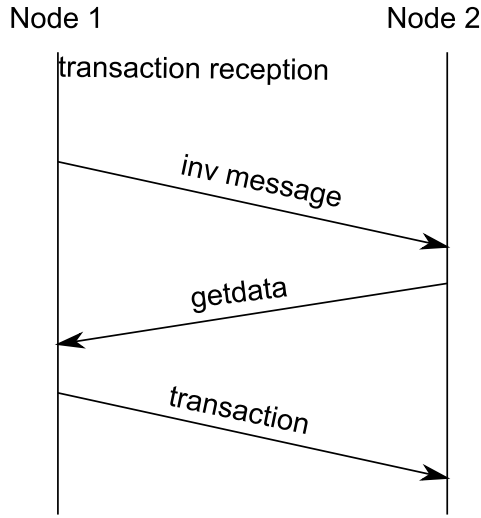

Figure 4.3 Transaction advertisement management system in Bitcoin.

4.1.4 Transaction Confirmation

In Section 4.1.1, we described the transaction verification process in Bitcoin. Note that this process cannot guarantee by itself the security of transactions since a powerful adversary may try to modify the history of transactions that occurred in the system (e.g., in order to double-spend or to increase its advantage in the system).

To this end, Bitcoin relies on the hash-based PoW mechanism in order to (computationally) prevent any entity from modifying the history (and order) of the transactions executed within the system. As mentioned in Chapter 3, to generate a block, miners must find a nonce value that, when hashed with the Merkle hash of all valid and received transactions included in their memory pool, the hash of the previous block, and a time stamp, the result is below a given target difficulty.

Since each block links to the previously generated block, the Bitcoin blockchain grows upon the generation of a new block in the network. In this way, blocks confirm Bitcoin transactions and commit them in the system. Namely, if any entity wants to modify the transactions executed in the system, then it does not only have to redo all the work required to compute the block where that transaction was included, but it has to also recompute all the subsequent blocks in the chain. That is, the older

a Bitcoin transaction is, and thus the deeper it is included in the blockchain, the harder it becomes to modify the transaction.

4.1.4.1 Selfish Mining

The original white paper of Bitcoin [5] claimed that the security of transactions in the system can be guaranteed as long as more than 50% of the network miners are honest. The main intuition here is that, in case of conflict or fork in the blockchain, then honest peers will adopt the longest Bitcoin chain—which is backed up by the majority of the computing power in the system. As long as honest peers control the majority of computing power in the system (i.e., they control more than 50% of the hash rate), then they can sustain the prolongation of the longest chain and ensure that only valid transactions are confirmed in this chain.

On the other hand, an adversary that controls more than 50% of the computing power in the system can, in theory, double-spend transactions, prevent transactions from being confirmed, prevent honest miners from mining valid blocks, and so on. This clearly invalidates the entire security of Bitcoin.

Eyal and Sirer [6] have shown that this limit can be considerably reduced. Namely, the authors showed that selfish miners which command more than 33% of the total computing power of the network can acquire a considerable mining advantage in the network. In [7], Sapirshtein et al. extended these results and provided even lower bounds on the computational power an attacker needs in order to benefit from selfish mining. Namely, in the selfish mining strategy of [6], a selfish miner does not directly announce its newly mined blocks in the network and instead keeps them secret until the remaining network finds new blocks. This strategy aims at wasting the computing power invested by other honest miners in the system; these miners will be investing their computing power toward building a block that is unlikely to be part of the longest chain.

To deter this misbehavior, Eyal and Sirer propose the following countermeasure: when a miner is aware of two competing blocks, the miner should propagate both blocks and select a random block to mine on.

Recent analysis has shown that this countermeasure can be easily circumvented by the adversary. For instance, it was recently shown in [4,8] that a resource-constrained adversary can deny the delivery of blocks in the system for a considerable amount of time. Namely, by exploiting the object request management system of Bitcoin as described in Section 4.1.3, a resource-constrained adversary can prevent the delivery of blocks for at least 20 minutes (since the time-out of block reception in the request management system of Bitcoin is 20 minutes). By doing so,

an adversary can subvert the aforementioned countermeasure indicated by Eyal and Sirer against selfish mining [4].

Several other recent works examined the game theoretic consequences of attacks and cooperation between pools. For instance, Eyal [9] has shown that pools can gain additional advantage in the network by infiltrating into other pools. Namely, by registering with the victim pool, the attacking pool will then receive tasks and transfer them to some of its own miners. Although the attacker mining power is reduced, since some of its miners are used for block withholding, the attacker earns additional revenue by infiltrating into the other pool—which might increase the revenue of the attacker (and decrease the mining difficulty in the Bitcoin protocol).

Even worse, recent results show that by combining the aforementioned mining attacks with network-level attacks, the adversary can considerably increase its advantage in the selfish mining game [4, 10]. For instance, the findings of Gervais et al. [4] suggest that an adversary who performs selfish mining and denies block delivery from other miners can acquire considerable advantage in the network if he or she commands more than 26.5% of the computing power. Moreover, the authors showed that an adversary that commands less than 34% of the computing power can effectively sustain the longest blockchain and therefore control the entire network.

4.1.4.2 Transaction Confirmation Time

In what follows, we analyze the transaction confirmation time in Bitcoin (adapted from [1, 11]). Recall that transactions are confirmed by means of a PoW, as shown in (3.1).

We start by noting the following observations:

1. The probability of success in a single-nonce trial is negligible. Since SHA-256 is a pseudorandom permutation function, each of the 2^{32} nonces has $\frac{target}{2^{256}-1}$ probability of satisfying the PoW.
2. Miners compute their PoW independently; therefore, the success probability of one miner does not depend on the progress of others.
3. Miners frequently restart the generation of their PoW and whenever a new transaction is added to the memory pool of a miner, the Merkle root (included in the block) changes.
4. The time interval, dt, between the announcement of successive transactions is in the order of a few tens of seconds.

Given the first two observations, the probability of a miner of succeeding in a single block generation attempt can be modeled as an independent Bernoulli process with success probability $\varepsilon = \frac{target}{2^{256}-1}$. Based on the last observation, consecutive block generation attempts can be modeled as sequential Bernoulli trials with *replacement*. This claim is justified by the fact that the PoW progress invested by a miner (expressed as a number of hash calculations) prior to a PoW reset is negligible in comparison to $2^{256} - 1$.[2]

Let n_i refer to the number of attempts that a miner m_i performs within a time period δ. The probability p_i of m_i finding at least one correct PoW within these trials is given by $p_i = 1 - (1 - \varepsilon)^{n_i}$. Since ε and n_i are small, p_i can be approximated to $p_i = 1 - (1 - \varepsilon)^{n_i} \approx n_i\varepsilon$. Therefore, the set of trials of m_i within δ constitutes a single Bernoulli process with success probability $n_i\varepsilon$.

Assuming that there are ℓ miners, $\mathsf{m}_i,\ i = 1 \dots \ell$ with success probability $p_i,\ i = 1 \dots \ell$ respectively, the overall probability of success in block generation can be approximated to:

$$\mathsf{pr} \approx 1 - \prod_{i=1}^{\ell}(1 - p_i),\ \text{or}\ \mathsf{pr} = 1 - (1 - p)^{\ell} \approx \ell \cdot p.$$

This is true when $p\ell \ll 1$ and when the miners have equal computing power (i.e., $p_i = p, i = 1 \dots \ell$).

We divide time into equal-sized intervals of size δ; let $t_0 = 0$ denote the time when the last block was generated. Here, each miner can make up to n_i trials for block generation within each interval. Let the random variable X_k denote the event of success in the time interval between t_k and t_{k+1}. That is,

$$X_k = \begin{cases} 1 & \text{if a block is created between } t_{k-1}, \text{and } t_k, \\ 0 & \text{otherwise.} \end{cases}$$

It is clear that $\mathsf{Prob}(X_k = 1) = \mathsf{pr}$. We denote by $\mathcal{Y}$, the number of attempts performed by miners until a success is achieved. Note that $\mathcal{Y}$'s values are distributed according to the geometric distribution model since:

$$\mathsf{Prob}(\mathcal{Y} = k) = \mathsf{Prob}(X_k = 1) \prod_{i=1}^{k-1} \mathsf{Prob}(X_i = 0) = \mathsf{pr}(1 - \mathsf{pr})^{k-1}.$$

2 This is the case since the PoW progress approximates $2^{35} \ll 2^{256} - 1$ given the computing power of most Bitcoin miners [12, 13].

Assuming a constant rate of trials per time window δ, the number of failures until a success is observed in block generation is proportional to the block generation time $\mathcal{T}$.

$$\mathsf{Prob}(\mathcal{T} = k \cdot \delta) = \mathsf{Prob}(\mathcal{Y} = k) \; = \; \mathsf{pr}(1 - \mathsf{pr})^{k-1}.$$

Given this, we conclude that the distribution of block generation times can be modeled with a shifted geometric distribution with parameter pr [14]. In Figure 4.4, we sketch this distribution using parameters $p = 0.19$, and $\delta = 60$ seconds as advocated in [11]. Figure 4.4 shows that each block confirmation requires on average 10 minutes with a standard deviation of almost 20 minutes.

4.2 SECURITY OF ZERO-CONFIRMATION TRANSACTIONS

In the previous section, we discussed the security of standard transactions in Bitcoin. We showed that transaction confirmation relies on the PoW process, whose solution follows a shifted geometric distribution.

Our findings show that the time required to confirm transactions impedes the operation of many businesses that are characterized by a fast-service time. Namely, a client can wait up to 100 minutes before his or her payment receives the six confirmation required to validate standard payments in Bitcoin. It is also clear that vendors, such as vending machines and takeout stores [15], cannot rely on transaction confirmation when accepting Bitcoin payments. To remedy these problems, Bitcoin encourages vendors to accept fast Bitcoin payments with zero confirmations (i.e., without requiring that these transactions are confirmed in blocks), as soon as the vendor receives a transaction from the network transferring the correct amount of BTCs to one of its addresses [15, 16].

In what follows, we show that zero-confirmation transactions are insecure. We also outline a countermeasure to strengthen the security of zero-confirmation transactions.

4.2.1 (In-)Security of Zero-Confirmation Transactions

In what follows, we show that double-spending attacks are easily realizable on zero-confirmation transactions.

For that purpose, we assume a setting featuring a malicious client $\mathcal{A}$ and a vendor $\mathcal{V}$ connected through a Bitcoin network (see Figure 4.5). We assume that $\mathcal{A}$ wishes to acquire a service from $\mathcal{V}$ without having to spend its BTCs. More

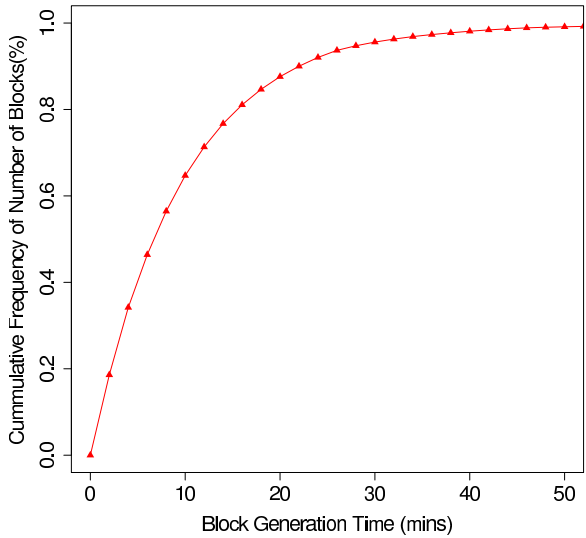

Figure 4.4 Cumulative distribution function (CDF) of block generation times. Approximately 30% of Bitcoin blocks take between 10 and 40 minutes to be generated.

specifically, $\mathcal{A}$ could try to double-spend the coin that he or she already transferred to $\mathcal{V}$. By double-spending, we refer to the case where $\mathcal{A}$ tricks the vendor $\mathcal{V}$ into accepting a transaction $\mathrm{TR}_{\mathcal{V}}$ that $\mathcal{V}$ will not be able to redeem subsequently. In this case, $\mathcal{A}$ creates another transaction $\mathrm{TR}_{\mathcal{A}}$ that has the same inputs as $\mathrm{TR}_{\mathcal{V}}$ (i.e., $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ use the same BTCs) but replaces the recipient address of $\mathrm{TR}_{\mathcal{V}}$—the address of $\mathcal{V}$— with a recipient address that is under the control of $\mathcal{A}$. Figure 4.6 shows an example of transactions $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$.

We assume that $\mathcal{A}$ can only control few peers in the network (that he or she can deploy since Bitcoin does not restrict membership) and does not have access to $\mathcal{V}$'s keys or machine. The remaining peers in the network are assumed to be honest and to correctly follow the Bitcoin protocol. We further assume that $\mathcal{A}$ does not participate in the block generation process.

Given this, we outline the necessary conditions for $\mathcal{A}$'s success in performing a double-spending attack on zero-confirmation transactions.

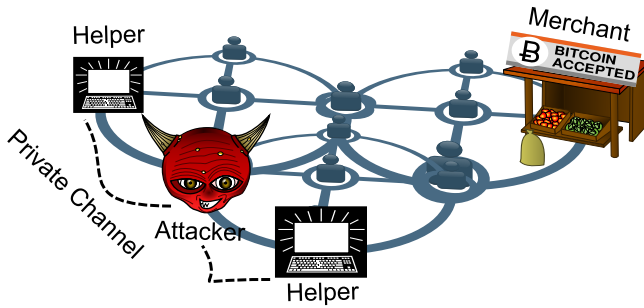

Figure 4.5 Our system model.

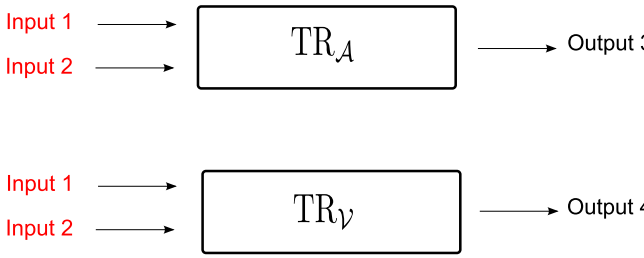

Figure 4.6 Example of two transactions involving double-spending coins labeled Input 1 and Input 2.

Requirement 1: $\mathrm{TR}_{\mathcal{V}}$ *is added to the wallet of* $\mathcal{V}$

If $\mathrm{TR}_{\mathcal{V}}$ is not added to the memory pool of $\mathcal{V}$, then $\mathcal{V}$ cannot check that $\mathrm{TR}_{\mathcal{V}}$ was indeed broadcasted in the network.

Let $t_i^{\mathcal{V}}$ and $t_i^{\mathcal{A}}$ denote the times at which node i receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$, respectively. As such, $t_{\mathcal{V}}^{\mathcal{V}}$ and $t_{\mathcal{V}}^{\mathcal{A}}$ denote the respective times at which $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$. Note that for $\mathrm{TR}_{\mathcal{V}}$ to be included in $\mathcal{V}$'s wallet, then $t_{\mathcal{V}}^{\mathcal{V}} < t_{\mathcal{V}}^{\mathcal{A}}$; otherwise, $\mathcal{V}$ will first add $\mathrm{TR}_{\mathcal{A}}$ to its memory pool and will reject $\mathrm{TR}_{\mathcal{V}}$ as it arrives later.

This requirement can be easily satisfied if $\mathcal{A}$ (or one of its helper nodes) connects directly to $\mathcal{V}$ and sends $\mathrm{TR}_{\mathcal{V}}$ to $\mathcal{V}$ first, before forwarding the double-spending transaction $\mathrm{TR}_{\mathcal{A}}$ to the rest of the network. In this way, $\mathcal{V}$ will immediately receive $\mathrm{TR}_{\mathcal{V}}$ and add it to its memory pool before receiving $\mathrm{TR}_{\mathcal{A}}$.

Requirement 2: $\mathrm{TR}_{\mathcal{A}}$ *is confirmed in the blockchain*

If $\mathrm{TR}_{\mathcal{V}}$ is confirmed first in the blockchain, $\mathrm{TR}_{\mathcal{A}}$ cannot appear in subsequent blocks. That is, $\mathcal{V}$ will not have its BTCs back. Recall that the goal of $\mathcal{A}$ is to acquire a service offered by $\mathcal{V}$ without having to spend his or her BTCs.

As shown experimentally in [1], this requirement can be satisfied by broadcasting $\mathrm{TR}_{\mathcal{A}}$ quickly to as many nodes in the network. Note that if $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ are released by $\mathcal{V}$ and $\mathcal{A}$ at the same time in the network, they are likely to have similar chances of getting confirmed in an upcoming block. This is the case since Bitcoin peers will not accept multiple transactions that share common inputs; they will only accept the version of the transaction that reaches them first, which they will consider for inclusion in their generated blocks and will ignore all subsequent transactions. Given this, a double-spending attack can succeed if $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$ (see Requirement 1), and the majority of the peers in the network receive $\mathrm{TR}_{\mathcal{A}}$ so that $\mathrm{TR}_{\mathcal{A}}$ is more likely to be included in a subsequent block. This can be achieved if $\mathcal{A}$ can rely on the cooperation of one or more helper nodes that help him or her broadcast $\mathrm{TR}_{\mathcal{A}}$ to a large number of nodes. Note that $\mathcal{A}$ (and its helpers) can try to increase the number of their immediate neighbors in order to increase the success probability of their attack.

Requirement 3: $\mathcal{V}$*'s service time is smaller than the time it takes* $\mathcal{V}$ *to detect misbehavior*

Since Bitcoin users are anonymous and users hold many accounts, there is only limited value in $\mathcal{V}$ detecting misbehavior after the user has obtained the service (e.g., left the store). As such, for $\mathcal{V}$ to successfully detect any misbehavior by $\mathcal{A}$, the detection time must be smaller than the service time.

We point out that requirements (1) and (2) are sufficient for the case where the vendor only checks for the reception of the transaction as a proof of payment and does not employ any other double-spending prevention/detection techniques. This is currently the case in most existing Bitcoin client implementations.

This analysis shows that requirements 1, 2, and 3 can be realizable in existing Bitcoin implementations.

This analysis is confirmed by means of experimental results adapted from [1] and summarized in Table 4.1. These results confirm that double-spending attacks succeed with a probability of almost 100% if $\mathcal{A}$ utilizes at least one additional helper node. This clearly shows that zero-confirmation transactions are not secure in Bitcoin and should not be accepted directly by vendors.

Table 4.1
Success Probability in Double-Spending Zero Confirmation Payments in Bitcoin[3]

Location	# Helpers	Success probability
Asia Pacific 1, 125 connections	2	100%
Asia Pacific 2, 125 connections	2	100%
North America 1, 8 connections	1	100%
North America 2, 40 connections	1	90%
Asia Pacific 1, 8 connections	2	100%
Asia Pacific 2, 125 connections	2	100%
North America 1, 40 connections	1	100%

4.2.1.1 Finney Attack

Note that we assume so far that $\mathcal{A}$ does not participate in the mining process. In case $\mathcal{A}$ is a miner or compromises a node that participates in the mining process, then the advantage of $\mathcal{A}$ in mounting double-spending attacks on zero-confirmation transactions can further increase depending on the mining power available to the adversary.

Namely, Finney [17] describes a double-spending attack in Bitcoin where the attacker includes in his or her generated blocks a number of transactions that transfer some coins between his or her own addresses. These blocks are only released in the network after the attacker double-spends the same coins using zero-confirmation payments and acquires a given service.

Clearly, the success probability of this attack depends on the mining power available to the adversary. Given the tremendous computing power that supports the current Bitcoin network,[4] the success probability of an adversary that does not control a considerable fraction of the mining power is only negligible.

3 Here, "Location" denotes the location of $\mathcal{V}$, "connections" denote the number of $\mathcal{V}$'s connections. The success probability is adapted from the findings of [1] and is interpolated by means of experiments using Amazon nodes.

4 The hashing rate in Bitcoin amounted to $0.5 \cdot 10^{15}$ hashes per second in November 2015.

4.2.2 Possible Countermeasures

We start by discussing a number of countermeasures to alleviate this attack. We also present a solution that is integrated in Bitcoin XT and we analyze the limitations of this solution.

Adopting a Listening Period

As advocated in [15], one possible way for $\mathcal{V}$ to detect double-spending attempts is to adopt a listening period of a few seconds before delivering its service to $\mathcal{A}$; during this period, $\mathcal{V}$ monitors all the transactions it receives, and checks if any of them attempt to double-spend the coins that $\mathcal{V}$ previously received from $\mathcal{A}$. This techniques are based on the intuition that since it takes every transaction a few seconds to propagate to every node in the Bitcoin network, then it is highly likely that $\mathcal{V}$ would receive both $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ within the listening period (and before granting service to $\mathcal{A}$).

This detection technique can be circumvented by $\mathcal{A}$ as follows. $\mathcal{A}$ can attempt to delay the transmission of $\mathrm{TR}_{\mathcal{A}}$ such that $t = (t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}})$ exceeds the listening period (requirement (3)) while $\mathrm{TR}_{\mathcal{A}}$ still has a significant chance of being spread in the network. On one hand, as t increases, the probability that all the immediate neighbors of $\mathcal{V}$ in the Bitcoin P2P network receive $\mathrm{TR}_{\mathcal{V}}$ first also increases; when they receive $\mathrm{TR}_{\mathcal{A}}$ later on, $\mathrm{TR}_{\mathcal{A}}$ will not be added to the memory pool of $\mathcal{V}$'s neighbors and as such $\mathrm{TR}_{\mathcal{A}}$ will not be forwarded to $\mathcal{V}$. On the other hand, $\mathcal{A}$ should make sure that $\mathrm{TR}_{\mathcal{A}}$ was received by enough peers so that requirement (2) can be satisfied. To that end, $\mathcal{A}$ can increase the number of helpers it controls.

As shown in Table 4.2 (results adapted from [1]), an adversary can successfully double-spend transactions even if the merchant adopts a listening period of 15 seconds. The detection probability in this case varies between 10% and 80% depending on the topology of the underlying overlay Bitcoin network. Even worse, as shown in Table 4.3, there are cases in which the vendor can never detect a double-spending attack even if he or she adopts an infinite listening period time. These cases correspond to the scenario where all the neighbors of $\mathcal{V}$ have received $\mathrm{TR}_{\mathcal{V}}$ first and therefore they will never forward $\mathrm{TR}_{\mathcal{A}}$ to $\mathcal{V}$.

Inserting Observers in the Network

Note that $\mathcal{V}$ can also rely on additional nodes that it controls within the Bitcoin network—observers—which would directly relay to $\mathcal{V}$ all the transactions that they

Table 4.2

Experimental Detection Probability Using a Listening Period of 15 Seconds [5]

Setting	Detection probability
Europe, 8 connections, 3 helpers	10%
Europe, 8 connections, 3 helpers	10%
South America, 8 connections, 2 helpers	6.66%
Asia Pacific, 8 connections, 2 helpers	20%
North America, 20 connections, 5 helpers	11%
North America, 20 connections, 5 helpers	10%
North America, 20 connections, 1 helper	30%
North America, 20 connections, 4 helpers	63%
North America, 20 connections, 2 helpers	20%
North America, 20 connections, 1 helper	30%
Europe, 20 connections, 3 helpers	45%
Europe, 30 connections, 1 helper	10%
Asia Pacific, 40 connections, 1 helper	10%
Europe, 40 connections, 1 helper	10%
Europe, 40 connections, 2 helpers	20%
South America, 40 connections, 1 helper	40%
Asia Pacific, 80 connections, 1 helper	20%
Europe, 80 connections, 1 helper	26.66%
Asia Pacific, 100 connections, 1 helper	80%

receive. This countermeasure circumvents the limitations of the listening period as it is hard for an adversary to identify all observers in the network and deny them the delivery of $\mathrm{TR}_{\mathcal{A}}$. However, this technique incurs additional costs on merchants who have to invest in additional equipment to deploy observers in various locations around the globe. As shown in Table 4.4 (results adapted from [1]), almost five different observers need to be deployed across the globe to ensure that at least one observer detects double-spending attacks.

5 Here, 'Setting' refers to the location of $\mathcal{V}$, the number of connections of $\mathcal{V}$ at the time of the attack, and the number of helpers employed by $\mathcal{A}$.

Table 4.3
Experimental Instances Where $\mathrm{TR}_{\mathcal{A}}$ Is Not Received by $\mathcal{V}$ [6]

Setting	Success probability
South America, 8 connections, 3 helpers	7.7%
South America, 8 connections, 4 helpers	57%
Asia Pacific, 8 connections, 3 helpers	57%
Asia Pacific, 8 connections, 3 helpers	66%
North America, 20 connections, 3 helpers	47%
Asia Pacific, 60 connections, 1 helper	20%

Refusing Incoming Connections

The success probability of double-spending attacks on zero-confirmation transactions heavily depends on the propagation delay of $\mathrm{TR}_{\mathcal{V}}$ from $\mathcal{A}$ to $\mathcal{V}$. Clearly, a direct communication channel between $\mathcal{A}$ and $\mathcal{V}$ considerably contributes to the success of the attack. For instance, if merchants do not accept incoming connections or are located behind firewalls and NATs, then the success probability of double-spending attacks can be reduced.

Note that it is hard, however, for merchants to ensure that all their current connections are trustworthy. For example, their connections can be compromised by the adversary.

Increasing the Number of Neighbors

We additionally point out that the number of connections established by $\mathcal{V}$ is an important parameter affecting the success of double-spending attacks. That is, the fewer connections of $\mathcal{V}$, the more likely is that all the neighbors of $\mathcal{V}$ receive $\mathrm{TR}_{\mathcal{V}}$ before $\mathrm{TR}_{\mathcal{A}}$ and thus that $\mathcal{V}$ does not receive $\mathrm{TR}_{\mathcal{A}}$ and therefore cannot detect the attack. Similarly, as the number of connections of $\mathcal{V}$ increases, it is more likely that

6 In this case, $\mathcal{V}$ cannot detect double-spending attacks even if it adopts a very large listening period. Here, 'Setting' refers to the location of $\mathcal{V}$, the number of connections of $\mathcal{V}$ at the time of the attack, and the number of helpers employed by $\mathcal{A}$.

Table 4.4
Experimental Detection Probability Using 5 Observers [7]

Setting	% Observed
Europe, 8 connections, 3 helpers	53%
Europe, 8 connections, 3 helpers	47%
South America, 8 connections, 2 helpers	62%
Asia Pacific, 8 connections, 2 helpers	91%
North America, 20 connections, 5 helpers	46%
North America, 20 connections, 5 helpers	74%
North America, 20 connections, 1 helper	78%
North America, 20 connections, 4 helpers	78%
North America, 20 connections, 2 helpers	60%
North America, 20 connections, 1 helper	60%
Europe, 20 connections, 3 helpers	87%
Europe, 30 Connections, 1 helper	42%
Asia Pacific, 40 connections, 1 helper	42%
Europe, 40 connections, 1 helper	36%
Europe, 40 connections, 2 helpers	36%
South America, 40 connections, 1 helper	57%
Asia Pacific, 80 connections, 1 helper	18%
Europe, 80 connections, 1 helper	28%
Asia Pacific, 100 connections, 1 helper	88%

some of these neighbors receive $\mathrm{TR}_{\mathcal{A}}$ before $\mathrm{TR}_{\mathcal{V}}$ and forward it to $\mathcal{V}$, who can immediately detect a double-spending attempt.

Inflicting Penalties on Misbehaving Nodes

Bitcoin recently adopted a penalty system to punish misbehaving nodes; for example, nodes that broadcast ill-formed objects can be temporarily (up to 24 hours) banned from connecting to a peer. One possible way to deter a double-spending

7 Here, 'Setting' refers to the location of $\mathcal{V}$, the number of connections of $\mathcal{V}$ at the time of the attack, and the number of helpers employed by $\mathcal{A}$. '% Observed' refers to the fraction of observers detecting double-spending attacks.

attack would be to rely on the alert mechanism of Bitcoin to alter the network of a misbehaving address that is attempting double-spending attacks.

For example, in our case, nodes can forward both $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ using an alert message to the rest of the network. All Bitcoin nodes can then verify that the address of $\mathcal{A}$ is attempting a double-spend and can decide not to accept any transaction issued by this address. However, the impact of this penalty is limited since $\mathcal{A}$ could double-spend using addresses that contain little (or no) BTCs. In Chapter 5, we show how to link different Bitcoin addresses of an entity in an attempt to inflict a harsh penalty on $\mathcal{A}$.

Not Advertising $\mathrm{TR}_{\mathcal{V}}$

Bamert et al. [18] suggested that $\mathcal{V}$ can effectively avoid isolation by not relaying transaction $\mathrm{TR}_{\mathcal{V}}$. By doing so, all the neighbors of $\mathcal{V}$ will forward $\mathrm{TR}_{\mathcal{A}}$ to $\mathcal{V}$ who will be able to detect the double-spending attack immediately.

This countermeasure can be circumvented if the attacker also similarly does not immediately advertise $\mathrm{TR}_{\mathcal{A}}$ in the network. As soon as the $\mathcal{V}$ advertises $\mathrm{TR}_{\mathcal{V}}$ (and one of the attacker's nodes receives it), $\mathrm{TR}_{\mathcal{A}}$ can be advertised in the network to prevent $\mathcal{V}$ from detecting the attack.

Forward First Double-Spend Attempt

In order to efficiently detect double-spending on zero-confirmation transactions, Karame et al. proposed in [1,11] that Bitcoin peers forward transactions that attempt to double-spend the same coins in the Bitcoin network. Namely, whenever a peer receives a new transaction, it checks whether the transaction uses coins that have not been spent in any other transaction that resides in the blockchain and in their memory pool. If so, then peers follow the current protocol of Bitcoin; peers add the transaction to their memory pool and forward it in the network. If, on the other hand, peers detect that there is another transaction in their memory pool that spends the same coins to different recipients, then the peers forward the transaction to their neighbors (without adding the transaction to their memory pools).

To decrease the number of transactions circulating in the Bitcoin network and to prevent the deterioration of the performance of the network, peers can only forward the first double-spending transaction attempt in the network and drop all subsequent double-spending of the same coin. This variant ensures that all peers in the network can identify and verify the misbehaving address and refuse to receive

any subsequent transaction from this address. This variant detection technique has been integrated in Bitcoin XT [19].

Recently, Gervais et al. [4] showed that the protection of Bitcoin XT is not effective in preventing double-spending attacks of fast payments. They show that $\mathcal{A}$ can deny the delivery of double-spending transactions to the merchant using the attack described in Section 4.1.3, thus effectively preventing a Bitcoin XT node from discovering any double-spending attempt.

4.3 BITCOIN FORKS

We now discuss another important security threat to Bitcoin, forks.

During the normal Bitcoin operation, miners work on extending the longest blockchain in the network. If miners do not share the same view in the network (e.g., due to network partitioning), they might work on different blockchains, thus resulting in forks in the blockchain (see Figure 4.7). Block forks are inherently resolved by the Bitcoin system; the longest blockchain (which is backed up by the majority of the computing power in the network) will eventually prevail. In rare instances, the Bitcoin developers can force one chain to be adopted at the expense of others [20]. Transactions that do not appear in blocks that are part of the main blockchain (i.e., the longest) will be readded to the pool of transactions in the system and reconfirmed in subsequent blocks.

During block forks, the adversary bears little risk in performing double-spending attacks. Indeed, under such settings, the adversary can try to include $\mathrm{TR}_{\mathcal{V}}$ in one chain, and $\mathrm{TR}_{\mathcal{A}}$ in another [17]. In what follows, we discuss in greater detail double-spending attacks in the special case where Bitcoin is subject to blockchain forks [21].

4.3.1 Exploiting Forks to Double-Spend

In what follows, we describe an example of a double-spending attack—that was tested in Bitcoin in [22]—and takes advantage of block forks. This attack leverages an exploit in Bitcoin that arises from the simultaneous adoption of client versions 0.8.1 and 0.8.2 (or beyond) in the network. Starting from version 0.8.2, Bitcoin clients no longer accept transactions that do not follow a given signature encoding. As we show, this incompatibility with prior client versions can potentially lead to a double-spending attack on zero-confirmation payments in Bitcoin. Note that this attack can only work when $\mathcal{V}$ operates on any client version prior to 0.8.2.

Up to version 0.8.1, a transaction signature could contain zero-padded bytes and the signature check would still be valid. However, starting from version 0.8.2, transactions with padding will no longer be accepted to the memory pool of nodes nor will they be relayed to other nodes.[8] This gives a considerable advantage for $\mathcal{A}$ to mount a double-spending attack as follows:

1. $\mathcal{A}$ sends a transaction $\mathrm{TR}_{\mathcal{V}}$ with a zero-padded signature to $\mathcal{V}$.
2. $\mathrm{TR}_{\mathcal{V}}$ will be relayed to the miners. Miners that use any Bitcoin version newer than 0.8.1 will not accept the transaction in their memory pool and thus not include it into a block. Miners with an older Bitcoin version will accept it.
3. $\mathcal{A}$ waits for a short time t (e.g. 1—5 minutes) until he or she acquires service from the merchant.
4. Then, provided that $\mathrm{TR}_{\mathcal{V}}$ was still not included in a Bitcoin block, $\mathcal{A}$ sends another transaction $\mathrm{TR}_{\mathcal{A}}$ that double-spends the inputs of $\mathrm{TR}_{\mathcal{V}}$ to the benefit of a new Bitcoin address that is controlled by $\mathcal{A}$. $\mathrm{TR}_{\mathcal{A}}$ is not padded with additional zeros.
5. If most peers in the network use newer client versions than version 0.8.1, they will accept $\mathrm{TR}_{\mathcal{A}}$ (and will reject $\mathrm{TR}_{\mathcal{V}}$). The higher the fraction of peers that use version 0.8.2 (or beyond), the larger is the likelihood that $\mathrm{TR}_{\mathcal{A}}$ is included in a block and that the attack succeeds.

While block forks might naturally occur from time to time in the network, such forks are unlikely to last for more than few blocks, as the network views tend to naturally converge on the longest blockchain within a few blocks. We argue that new version releases, on the other hand, can cause more serious damage since they might result in long-lasting block forks that can only be stopped by manual intervention. Version releases should therefore be carefully designed for backward-compatibility; otherwise, the Bitcoin system might witness severe misbehavior.

4.3.2 Fork Resolution

As mentioned earlier, blockchain forks are detrimental to the operation of the Bitcoin system. Since one blockchain will eventually prevail (the longest), all transactions that were included in all other chains will be invalidated by the miners in the system.

8 This applies to all Bitcoin versions starting from version 0.8.2 until the time of writing (i.e., version 0.8.5).

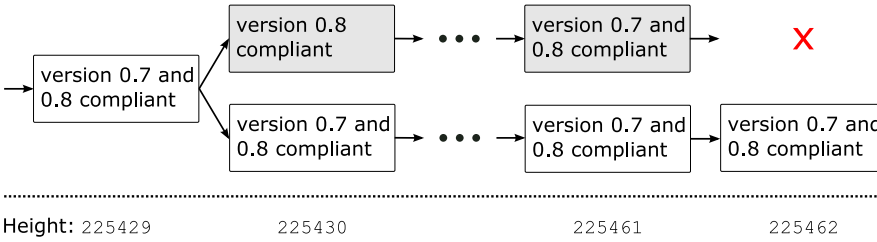

Figure 4.7 Sketch of the Bitcoin blockchain fork that occurred in Bitcoin on 11.03.2013.

Note that Bitcoin does not embed any mechanism to alleviate this problem; instead, if the fork persists for a considerable period of time, Bitcoin developers have to make a decision in favoring one chain at the expense of another (e.g., by sending alert messages and hard-coding the preferred chain in the client code).

As an example, we describe a recent chain fork in March 2013 (adapted from [20]) that solicited intervention from the Bitcoin developers.

Bitcoin client version 0.7 stored the blockchain in the BerkleyDB database, while client version 0.8 switched to the more efficient LevelDB database. Version 0.7 sets the threshold for the maximum number of locks per BerkleyDB update to 10,000; this limit, on the other hand, is set to 40,000 in version 0.8. This discrepancy caused a serious fork in the blockchain starting from block 225,430 on March 11, 2013. This block contained around 1,700 transactions, affected more than 5,000 block index entries, and therefore exceeded the required number of locks for version 0.7 (each block index entry requires around 2 locks in BerkleyDB). As a consequence, this resulted in a severe block fork in the chain; all version 0.7 miners rejected block 225,430 and continued working on a blockchain that did not include it, while miners with version 0.8 accepted that block and added it to their blockchain.

The chain adopted by version 0.8 clients was supported by the majority of the computing power in the network (it exceeded the chain adopted by 0.7 clients by 13 blocks at block 225,451). Nevertheless, the Bitcoin developers decided, 90 minutes after the fork occurred, to force the smallest chain to be the genuine one.

This decision comes at odds with the claim that Bitcoin is a decentralized system and that the majority of the computing power regulates Bitcoin. Less than 10 entities [23] took a decision to outvote the majority of the computing power in the network; this decision has affected the transactions of thousands of users. We also point out that such influential entities also have the power to make more radical decisions (e.g., accepting or rejecting transactions in the system).

References

[1] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917, New York, 2012. ACM.

[2] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 129–144, Berkeley, CA, USA, 2015. USENIX Association.

[3] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 692–705, New York, 2015. ACM.

[4] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. *IACR Cryptology ePrint Archive*, 2015:578, 2015.

[5] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[6] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[7] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.

[8] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. *CoRR*, abs/1402.1718, 2014.

[9] Ittay Eyal. The miner's dilemma. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*, 2015.

[10] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IACR Cryptology ePrint Archive 2015*, 2015.

[11] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1):2:1–2:32, May 2015.

[12] Comparison of Mining Pools, 2013. available from `https://en.bitcoin.it/wiki/Comparison_of_mining_pools`.

[13] Comparison of Mining Hardware, 2013. available from `https://en.bitcoin.it/wiki/Mining_hardware_comparison`.

[14] Proofwiki, 2013. available from `http://www.proofwiki.org/wiki/Definition:Shifted_Geometric_Distribution`.

[15] Myths - Bitcoin, 2013. available from `https://en.bitcoin.it/wiki/Myths#Point_of_sale_with_bitcoins_isn.27t_possible_because_of_the_10_minute_wait_for_confirmation`.

[16] FAQ - Bitcoin, 2013. available from `https://en.bitcoin.it/wiki/FAQ`.

[17] The Finney Attack, 2013. available from `https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack`.

[18] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with bitcoins. In *13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy, September 2013.*, 2013.

[19] Bitcoin XT, 2015. available from https://github.com/bitcoinxt/bitcoinxt.

[20] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is Bitcoin a Decentralized Currency? *IEEE Security and Privacy Magazine, 2014*, May/June issue 2014, 2014.

[21] C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. In *13th IEEE International Conference on Peer-to-Peer Computing*, 2013.

[22] Arthur Gervais, Hubert Ritzdorf, and Ghassan O Karame. Double-spending fast payments in bitcoin due to client versions 0.8. *month*, 6(789):2, 2013.

[23] IRC Bitcoin incident resolution, available from http://bitcoinstats.com/irc/bitcoin-dev/logs/2013/03/11.

Chapter 5

Privacy in Bitcoin

To strengthen the privacy of its users, Bitcoin users participate in transactions using pseudonyms (i.e., Bitcoin addresses). Generally, each user can have hundreds of different Bitcoin addresses that are all stored and transparently managed by its client.

In spite of the reliance on pseudonyms, the public time-stamping mechanism (i.e., the blockchain) of Bitcoin raises serious concerns with respect to the privacy of users. In fact, given that Bitcoin transactions basically consist of a chain of digital signatures, the expenditure of individual coins can be publicly tracked [1–5] in the blockchain. Moreover, information is leaked in the Bitcoin system through the P2P network (the peer's connection and traffic relayed via these connections). For example, a potential attacker could link transactions to their originator IP address [6, 7] by studying connectivity and traffic of the peers.

Given the sharp increase of the user base of Bitcoin, and the growing use of Bitcoin as a currency and payment protocol in various online applications, the need to support privacy in Bitcoin is becoming more prevalent. As a first step in this direction, there have been a number of studies that quantify the privacy provisions of Bitcoin, assuming that the latter is used for everyday payment needs [3]. These studies clearly show the limits of privacy offered by Bitcoin.

Motivated by these findings, the Bitcoin community has focused on enabling privacy-preserving payments within Bitcoin. On the one hand, there has been a considerable number of start-ups that assume the role of payment *mixers*; in other words, these companies perform Bitcoin transactions *on behalf* of users registered to their service, and in this way they obfuscate the payment's origin. While such services have clear potential in solving both sides of the problem (protocol and network), they require trusting a third-party (e.g., the mixing service).

On the other hand, the literature features a considerable number of proposals extending the standard Bitcoin protocol in order to conceal the traceability of payments within Bitcoin [8–10], and/or to hide the payment amounts [9,10] without the need of a trusted third-party.

Throughout this chapter, we assume a realistic threat model as encountered in the current Bitcoin deployment. More specifically, we assume that users of the system act either as payers (coin senders) or as payees (coin recipients). From the system usability perspective, a user may be paying at the same time one or more users, and in some cases is able to merge multiple coins into a single coin of greater value.

From a security perspective, we assume that the adversary is motivated to acquire information about the addresses/transactions pertaining to all or a subset of Bitcoin users. As such, the adversary does not only have access to the public log of transactions, denoted by pubLog, but is also *part of the Bitcoin system* and can perform or receive payments through Bitcoin. Here, the adversary can also access the (public) addresses of some vendors along with (statistical) information such as the pricing of items or the number of their clients within a specified amount of time.

We, however, assume that the adversary is computationally bounded and as such cannot construct ill-formed Bitcoin blocks, double-spend confirmed transactions, forge signatures, and so on.

This chapter is divided into two parts. In Section 5.1, we start by evaluating the privacy provisions of Bitcoin in light of a number of reported attacks on the system. In Section 5.3, we describe and analyze a number of proposals for enabling privacy-preserving payments in Bitcoin.

5.1 USER PRIVACY IN BITCOIN

The literature contains a number of proposals that analyze the privacy offered in Bitcoin [3, 4, 11]. Triggered by the fact that all Bitcoin transactions are posted in a publicly available ledger, the research community investigated the degree to which this public log of transactions would leak information on the profiles of Bitcoin users or enable the tracing of a single person's activities.

As mentioned in previous chapters, to strengthen the privacy of its users, Bitcoin users participate in transactions using pseudonyms (or addresses). However, since transactions basically consist of a chain of digital signatures, the expenditure of individual coins can be publicly tracked [1].

Motivated by these facts, we investigate and quantify in what follows the privacy that is provided by Bitcoin.

5.1.1 Protocol-Based Privacy Quantification in Bitcoin

To quantify privacy in Bitcoin, we observe the public log of Bitcoin, denoted by pubLog, within a period of time Δt. During this period, $\mathrm{n_U}$ users, $\mathrm{U} = \{u_1, u_2, \ldots, u_{\mathrm{n_U}}\}$, participate in pubLog through a set of $\mathrm{n_A}$ addresses: $\mathsf{A} = \{a_1, a_2, \ldots, a_{\mathrm{n_A}}\}$. We assume that within Δt, $\mathrm{n_T}$ transactions have taken place as follows: $\mathrm{T} = \{\tau_1(\mathsf{S}_1 \rightarrow \mathsf{R}_1), \ldots, \tau_{\mathrm{n_T}}(\mathsf{S}_{\mathrm{n_T}} \rightarrow \mathsf{R}_{\mathrm{n_T}})\}$, where $\tau_i(\mathsf{S}_i \rightarrow \mathsf{R}_i)$ denotes a transaction with (unique) ID number i and S_i and R_i denote the sets of senders' addresses and recipients' addresses, respectively.

For the analysis presented here, we consider the privacy measures adopted by existing Bitcoin clients. Namely, we assume that (1) users own many Bitcoin addresses, and (2) users are encouraged to frequently change their addresses (by transferring some of their BTCs to newly created addresses); this conforms with the current practices adopted in Bitcoin. Moreover, conforming with the operation of existing client implementations, a new address—the shadow address [12]—is automatically created and used to collect back the change that results from any transaction issued by the user. Besides the reliance on pseudonyms, shadow addresses constitute the only mechanism adopted by Bitcoin to strengthen the privacy of its users.

In what follows, we introduce a notion to quantify Bitcoin privacy, activity unlinkability, and and we provide metrics to appropriately quantify it in the existing Bitcoin system.

Activity linkability refers to the ability of an adversary $\mathcal{A}$ to link two different addresses (address linkability) or transactions (transaction linkability) that pertain to the same user of the system, and in this sense, activity unlinkability is strongly associated to accountability. That is, the more a third-party (e.g., law enforcement) is able to reconstruct the set of addresses or transactions of an individual, the easier it is to make Bitcoin users accountable for any misbehavior. More specifically, fee-based punishments for double-spending acts could be more effective, for example, by blacklisting or invalidating the BTCs of the addresses that are linked to the double-spender address.

However, activity linkability seems to contradict the privacy requirements of a payment system with public transaction logs as Bitcoin, where it is crucial to maintain the confidentiality of each individual's balance and transactions. Therefore, we see *activity unlinkability* as the privacy-preserving complement of linkability.

We note that since two Bitcoin transactions are not more linkable than the addresses that participate in those transactions, we focus our analysis on unlinkability of addresses. In particular, we define address unlinkability through the following AddUnl game, and we quantify it by assessing the advantage of an adversary $\mathcal{A}$ in winning this game over an adversary who responds to all game challenges with random guesses, $\mathcal{A}^{\mathcal{R}}$. We assume that $\mathcal{A}$ has access to pubLog and that both $\mathcal{A}$ and $\mathcal{A}^{\mathcal{R}}$ have gathered the same a priori knowledge $\mathcal{K}_{\mathcal{A}}$ with respect to correlations of a subset of addresses. $\mathcal{K}_{\mathcal{A}}$ can include any information related to address ownership (e.g., the identity of the owner of the address) the transactional habits of the latter, whether two specific addresses are owned by the same individual, and so on. For simplicity, we assume in the following that $\mathcal{K}_{\mathcal{A}}$ consists of a list of probabilities of correlating every pair of addresses in pubLog; clearly, the correlation probability between addresses for which the adversary has no prior knowledge about, equals the default probability that the two addresses are owned by the same individual (depending on the assumed game). The adversary can gather this a priori knowledge, for example, by interacting with users in the system [5].

We construct the following address unlinkability game in Bitcoin, AddUnl, which consists of an adversary $\mathcal{A}$ and of a challenger $\mathcal{C}$ who knows the correct assignment of addresses to Bitcoin entities. The adversary $\mathcal{A}$ chooses an address a_0, chosen among the addresses that appear in pubLog, but for which the adversary has no prior knowledge (expressed in $\mathcal{K}_{\mathcal{A}}$), and sends it to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ chooses a bit b uniformly at random. If $\mathit{b} = 1$, then $\mathcal{C}$ chooses another address a_1 randomly from pubLog such that a_0, a_1 belong to the same user; otherwise, $\mathcal{C}$ randomly chooses a_1 such that the two addresses are owned by different users. The challenger sends $\langle a_0, a_1 \rangle$ to $\mathcal{A}$, who responds with his or her estimate b' on whether the two addresses belong to the same user. $\mathcal{A}$ wins the game if he or she answers correctly (i.e., $\mathit{b} = \mathit{b}'$). We say that Bitcoin satisfies address unlinkability if for all probabilistic polynomial time (p.p.t.) adversaries $\mathcal{A}$, and $\forall \langle a_0 \rangle$, $\mathcal{A}$ has only at most a negligible advantage over $\mathcal{A}^{\mathcal{R}}$ in winning:

$$\mathrm{Prob}[\mathit{b}' \leftarrow \mathcal{A}(\mathcal{K}_{\mathcal{A}}, a_0, a_1) : \mathit{b} = \mathit{b}'] - \mathrm{Prob}[\mathit{b}' \leftarrow \mathcal{A}^{\mathcal{R}}(\mathcal{K}_{\mathcal{A}}, a_0, a_1) : \mathit{b} = \mathit{b}'] \leq \varepsilon,$$

where ε is negligible with respect to the security parameter κ.

Quantifying Address (Un-)linkability: In what follows, we quantify the unlinkability offered by Bitcoin by measuring the *degree* to which Bitcoin addresses can be *linked* to the same user. To do so, we express the estimate of $\mathcal{A}$ through an $\mathrm{n_A} \times \mathrm{n_A}$ matrix, $\mathrm{E_{link}}$, where $\mathrm{E_{link}}[i, j] = \{p_{i,j}\}_{i,j \in [1, \mathrm{n_A}]}$. That is, for every address a_i, $\mathcal{A}$ assesses the probability $p_{i,j}$ with which a_i is owned by the same user as every

other address a_j in publog. Note that $\mathrm{E}_{\mathrm{link}}$ incorporates $\mathcal{K}_{\mathcal{A}}$, and any additional information that $\mathcal{A}$ could extract from publog (e.g., by means of clustering or statistical analysis). Similar to [13], we quantify the success of $\mathcal{A}$ in the AddUnl game as follows. Let $\mathrm{GT}_{\mathrm{link}}$ denote the genuine address association matrix; that is, $\mathrm{GT}_{\mathrm{link}}[i,j] = 1$, if a_i and a_j are of the same user and $\mathrm{GT}_{\mathrm{link}}[i,j] = 0$, otherwise for all $i,j \in [1,\mathrm{n_A}]$. For each address a_i, we compute the error in $\mathcal{A}$'s estimate which denotes the distance of $\mathrm{E}_{\mathrm{link}}[i,*]$ from the genuine association of a_i with the rest of the addresses in publog $||\mathrm{E}_{\mathrm{link}}[i,*] - \mathrm{GT}_{\mathrm{link}}[i,*]||$, where $||\cdot||$ denotes the L1 norm of the corresponding row-matrix. Thus, the success of $\mathcal{A}$ in AddUnl, $\mathrm{Succ}_{\mathcal{A}}$, can then be assessed through $\mathcal{A}$'s maximum error: $\max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}} (||\mathrm{E}_{\mathrm{link}}[i,*] - \mathrm{GT}_{\mathrm{link}}[i,*]||)$.

Similarly, we represent the estimate of $\mathcal{A}^{\mathcal{R}}$ in the AddUnl game for all possible pairs of addresses by the $\mathrm{n_A} \times \mathrm{n_A}$ matrix $\mathrm{E}^{\mathcal{R}}_{\mathrm{link}}$ that are constructed as follows. $\mathrm{E}^{\mathcal{R}}[i,j] = \pi_{i,j}$ if $\langle a_i, a_j \rangle \in \mathcal{K}_{\mathcal{A}}$, and $\mathrm{E}^{\mathcal{R}}_{\mathrm{link}}[i,j] = \rho + (1-\rho)\frac{1}{2}$ otherwise. Here, $\pi_{i,j}$ represents the probability that addresses a_i a_j correspond to the same user according to $\mathcal{K}_{\mathcal{A}}$, and ρ is the fraction of addresses that cannot be associated to other addresses (i.e., when their owners have only one address). For pairs of addresses that are not included in $\mathcal{K}_{\mathcal{A}}$, this probability equals to $\frac{1}{2}(1+\rho)$; that is, the probability that at least one of the two cases happens: (1) a_0 is the only address of its owner, or (2) $\mathcal{A}^{\mathcal{R}}$ did not succeed in guessing b correctly.

Given this, we measure the degree of address linkability in Bitcoin by evaluating the additional success that $\mathcal{A}$ can achieve from publog when compared to $\mathcal{A}^{\mathcal{R}}$. We call this advantage $\mathrm{Link}^{\mathrm{abs}}_{\mathcal{A}} = \mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}$, and its normalized version $\mathrm{Link}_{\mathcal{A}} = \frac{\mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}$.

Address unlinkability can then be measured by the normalized complement of $\mathrm{Link}^{\mathrm{abs}}_{\mathcal{A}}$, $\mathrm{UnLink}_{\mathcal{A}} = 1 - \frac{\mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}$.

5.1.2 Exploiting Existing Bitcoin Client Implementations

Current Bitcoin client implementations enable $\mathcal{A}$ to link a fraction of Bitcoin addresses that belong to the same user.

Heuristic 1: Multi-input Transactions: Multi-input transactions occur when u wishes to perform a payment and the payment amount exceeds the value of each of the available BTCs in u's wallet. In fact, existing Bitcoin clients choose a set of BTCs from u's wallet (such that their aggregate value matches the payment) and perform the payment through multi-input transactions. It is therefore straightforward to conclude that if these BTCs are owned by different addresses, then the

input addresses belong to the same user [3, 4].

Heuristic 2: Shadow Addresses: As mentioned earlier, the standard Bitcoin client generates a new address, the shadow address [12], on which each sender can collect back the change [3].

This mechanism suggests a distinguisher for shadow addresses. Namely, in the case when a Bitcoin transaction has n output addresses, $\{a_{\mathsf{R}_1}, \cdot, a_{\mathsf{R}_n}\}$, such that only one address is a new address (i.e., an address that has never appeared in pubLog before), and all other addresses correspond to an old address (an address that has appeared previously in pubLog), we can safely assume that the newly appearing address constitutes a shadow address for a_i. Note that the official Bitcoin client started to support transactions with multiple recipients since December 16, 2010.

5.1.3 Summing Up: Behavior-Based Analysis

Besides exploiting current Bitcoin implementations, $\mathcal{A}$ could also make use of behavior-based clustering techniques, such as K-Means (KMC), and the Hierarchical Agglomerative Clustering (HAC) algorithms. Let U be the set of users populating Bitcoin and $(\mathsf{GA}_1, \ldots, \mathsf{GA}_{\mathrm{n_{GA}}})$ denote the GAs that $\mathcal{A}$ has obtained by applying the two aforementioned heuristics on pubLog, respectively. Given this, the goal of $\mathcal{A}$ is to output a group of clusters of addresses $\mathrm{E_{prof}} = \{\mathsf{g}_1, \ldots, \mathsf{g}_{n_\mathrm{U}}\}$ such that $\mathrm{E_{prof}}$ best approximates U. Since each GA is owned by exactly one user, the estimate on the assignment of each GA_i can be modeled by a variable z_i such that $z_i = k$, if and only if, GA_i belongs to g_k.

In fact, HAC assumes that initially each GA represents a separate cluster $(\{z_i = i\}_{i=1}^{\mathrm{n_{GA}}})$ and computes similarity values for each pair of clusters. Clusters with higher similarity value are combined into a single cluster and cluster-to-cluster similarity values are recomputed. The process continues until the number of created clusters equals the number of users $\mathrm{n_U}$.

KMC is then initialized using the output of HAC and assumes that each user is represented by the center of each cluster. The algorithm iterates assignments of GAs to clusters and aims at minimizing the overall distance of GAs to the center of the cluster they have been assigned to. The centers of the clusters and the GA-to-cluster distances are recomputed in each round.

In [3], Androulaki et al. investigated the effectiveness of behavior-based clustering algorithms in profiling Bitcoin users using a Bitcoin simulator. Their results can be summarized as follows:

- The authors show that given 200 simulated user profiles, almost 42% of the users have their profiles captured with 80% accuracy—which clearly results in considerable leakage.
- The profile leakage in Bitcoin is larger when users participate in a large number of transactions and decreases as the number of transactions performed by the user decreases. This is mainly due to the fact that users who participate in more transactions can be more easily profiled when compared to those users who only participate in few transactions.
- The overall number of transactions exchanged in Bitcoin has little impact on the profile leakage of users in the system. Their results show that even when the network features 70% less transactions, then the fraction of captured transactions per user does not decrease significantly irrespective of the activity level of each user.

These results suggest that the privacy provisions of Bitcoin are not strong, which opens the door to the integration of accountability measures in the system. It is straightforward to see that as accountability provisions in Bitcoin become stronger, the privacy provisions will become weaker. Notably, the less untraceable the user activity is within Bitcoin log, and therefore the more accountable the system is, the more individual privacy such as activity unlinkability is compromised.

5.1.4 Coin Tainting

Given that Bitcoin transactions basically consist of a chain of digital signatures, the expenditure of individual coins can be publicly tracked. This enables any entity to taint coins that belong to a specific set of addresses and monitor their expenditure across the network. The literature features a number of proposals that cluster Bitcoin addresses [3] and gather behavioral information about these addresses [4, 11].

Coin tainting *could* be used to achieve a degree of accountability in the Bitcoin network; if an address misbehaves, then Bitcoin users can decide to stop interacting with the address (i.e., not accepting its coins), thus deflating the value of all the coins pertaining to that address. For instance, following a theft of 43,000 BTCs from the Bitcoin trading platform Bitcoinica, the Bitcoin service MtGox traced the stolen BTC and deactivated the accounts that were receiving the tainted coins [14].

These incidents show that powerful entities in Bitcoin can—rightfully or not—devalue the BTCs owned by specific addresses. If these entities were to cooperate with the handful of developers that have privileged rights in the system,

then all Bitcoin users can be warned not to accept BTCs that pertain from a given address (e.g., using alert messages). Even worse, developers can hard-code a list of banned Bitcoin addresses within the official Bitcoin client releases, thus blocking all interactions with a given Bitcoin address without the consent of users.

Furthermore, while coin tainting can be used to punish provably misbehaving addresses, it could also be abused to control the financial flows in the network subject to government pressure, and due to social activism as well. This empowers a few powerful entities that are not necessarily part of the Bitcoin network, such as governments and activists, to regulate the Bitcoin economy. Even if all Bitcoin decisions and operations were completely decentralized—which they are not—coin tainting presents an obstacle to a truly decentralized Bitcoin.

Coin tainting can be especially detrimental if coins are not widely exchanged among Bitcoin addresses. This enables entities to damage only a specific set of addresses without alienating other addresses in the system. Other users are then also likely to boycott the tainted coins.

In [15], Gervais et al. conducted two experiments to analyze the impact of coin tainting on the Bitcoin network. In the first experiment, the authors measured the number of unspent transaction outputs (UTXO) that are affected when tainting a coinbase. Recall that a coinbase is the first transaction in a block, and attributes the block mining reward to a particular address. The authors randomly sampled 100 coinbases from the last 20,000 blocks of the blockchain that by the time of the experiment had the highest block-height at 247,054. The results show that a single coinbase tainting affects a large number of transaction outputs; on average, tainting a single coinbase affects 857,239 UTXO (with a standard deviation of 767,528), accounting for 12.9% of all possible UTXO.

In a second experiment, Gervais et al. analyzed the effect of tainting addresses belonging to a single entity in Bitcoin. Given the absence of data to identify these addresses, the authors relied on the two aforementioned heuristics in order to cluster addresses across Bitcoin entities.

As an application of this analysis, Gervais et al. [15] identified two Bitcoin addresses belonging to Torservers.net (using information available from `blockchain.info`). Given the knowledge of these two addresses, they were able to identify a total of 47 addresses belonging to the operator of Torservers with a total balance of 498.20 BTCs. If an external entity (e.g., a governmental institution) would like to stop the Torservers from receiving Bitcoin donations, it could taint all UTXO of the affected Bitcoin addresses. Recently, Silk Road—one of the most well known underground online black markets—was shut down by the FBI in October 2013. Note that Silk Road was only accessible through the Tor network; the FBI

seized over 27,000 BTCs stored within one or more Bitcoin addresses held by Silk Road.

5.1.5 Risks of Holding Tainted Bitcoins

Based on the aforementioned analysis, entities holding Bitcoins have to take into account various risks that are tightly coupled with coin tainting. For instance, the previous owners of their coins could have acquired these coins by means of illegal activities (e.g., theft). In general, any coin whose expenditure history involves a crime poses considerable risks for its new owners. Recall that coin tainting can be applied to incorporate accountability in the system; for example, it could be used to trace stolen BTCs or launch campaigns not to accept coins issued by suspicious senders.

To be on the safe side, this almost suggests that users should be aware of such risks whenever they receive a transaction in the network. For instance, the receiver has to compute the risk of accepting these coins and whenever needed instruct the sender to use different coins as a payment. Even if the receiver accepts to receive the coins, he or she might be inclined to immediately spend them in order to minimize any risk associated with holding these coins.

The risks associated with holding tainted coins has been thoroughly investigated in [16]. Here, the authors lay down a risk model for holding BTCs and outline a risk prediction approach using public knowledge from the Bitcoin blockchain. The authors observe there will be a certain timespan between the time at which an illegal transaction takes place and the time at which the corresponding coins are tainted/blacklisted in the network. Thus, honest users may accept a dirty BTC despite their willingness to comply with the blacklisting process. Clearly, freshly mined coins are less likely to be involved in suspicious activities when compared to coins who have been switched across several owners. This clearly motivates the need to predict the risk of holding/accepting coins to ensure that they do not lose value due to coin tainting or blacklisting over time.

5.2 NETWORK-LAYER ATTACKS

In this section, we show how an adversary can leverage information from the Bitcoin P2P network in order to profile Bitcoin users. We start with a brief refresher on information exchange in the Bitcoin network.

5.2.1 Refresher on Bitcoin P2P Network Setup

Initially, Bitcoin peers communicated with the rest of the network peers through unencrypted and unauthenticated TCP connections.

Since there was no authentication offered by the connection layer, peers maintain a list of IP addresses associated with their connections (neighbors). To avoid denial-of-service attacks, peers evaluate the behavior of their neighbors in the P2P network by implementing a reputation-based protocol. In particular, whenever a malformed message is received by a node, the peer decreases the reputation value (or increases the *penalty score*) of the associated connection categorized using its IP. As soon as the penalty score of a connection reaches a threshold value (currently 100), the peer rejects connection requests coming from that IP for 24 hours (see Chapter 3 for more detail on this process).

As mentioned in Chapter 3, Bitcoin peers maintain by default eight outgoing connections, also known as *entry nodes*.

5.2.2 Privacy Leakage over the Bitcoin Network

In the following, we present a method to deanonymize Bitcoin users by linking their pseudonyms (addresses) to the IP addresses of the underlying clients. This attack was first introduced in [6] and later expanded in [7].

Note that this attack allows to deanonymize users even when they operate behind network address translators (NATs) or firewalls. More specifically, this technique allows an adversary to distinguish connections and transactions pertaining to different users that are located behind the same NAT.

The main intuition behind this attack is that since entry nodes of any given client are not renewed by default (until the client restarts), each client can be safely and uniquely identified by the set of nodes that he or she connects to.

In terms of required resources, the attach only requires few running instances of the Bitcoin clients (each residing on a different IP) to establish a certain number of connections (following the Bitcoin protocol) and log the incoming transactions [6].

In a specific example offered in [6], an adversary equipped with no more than 50 connections to each Bitcoin server can disclose the sender's IP address for around 11% of all transactions generated in the Bitcoin network. Experimental results have shown that deanonymization rates of up to 60% can also be reached, if the adversary were to mount a small DoS on the network (see [6] for more details). The overall

cost of mounting such an attack on the full Bitcoin network is estimated to be around 1,500 EUR per month [6].

The attack evolves in three steps. First, the attacker attempts to disconnect users from Tor or other anonymizing networks that these clients may be leveraging for connecting to Bitcoin peers. This allows the adversary to use directly the information received by the network (e.g., to figure out the network's topology). Finally, the adversary can use the acquired network knowledge in combination with the mechanism that Bitcoin uses to forward transactions in the network, to deanonymize transactions. In what follows, we detail these steps.

Phase 1: Disconnecting Clients from Tor: The Tor network [17] comprises a set of relays that are publicly available online, and which can be used by any party to send a message while avoiding traffic analysis attacks. To establish a connection to a service or a node through Tor, a user chooses a chain of three Tor relays, through which the messages to the target service or node will be routed. The final node in the chain, also known as *Tor Exit node*, appears to the service as the originator of this connection.

To prevent Bitcoin users from making use of Tor when transacting with Bitcoin, the adversary could exploit the Bitcoin built-in DoS protection. Recall that in Bitcoin, whenever a peer receives a malformed message, it increases the penalty score of the IP address from which the message came and bans that IP for 24 hours when that score reaches 100. To exploit this, the adversary can simply try to connect to various Bitcoin nodes, using Tor, and send malformed messages, such that all Tor exit nodes are banned from the majority of the Bitcoin nodes. Alternatively, the adversary could simply spoof the IP of the exit node and issue malformed messages from that IP that would result in a 24-hour ban of the exit node.

Phase 2: Inferring Network Topology: This phase assumes that the use of Tor has been temporarily deactivated using the strategy described in the previous paragraphs.

In this phase, the adversary targets Bitcoin clients that do not accept incoming connections and only exhibit the minimum (i.e., eight) outgoing connections to the rest of the network. The goal of the adversary is to learn the eight entry nodes of each targeted Bitcoin client.

The attack unfolds as follows. Whenever a client C establishes a connection to one of its entry nodes, it engages in the address discovery protocol described in Chapter 3 and advertises its external addresses that have the highest local scores IP_{C}. If the adversary is already connected to one of those entry nodes, the address

$\mathrm{IP_C}$ will be forwarded to them with some probability (which depends on the number of the attacker's connections).

This suggests that the attacker can shortlist the entry nodes of the target address $\mathrm{IP_C}$ as follows:

- The attacker connects to a large number of Bitcoin server nodes, say $\mathrm{N}_{\mathcal{A}}$ which is assumed to be close to the set of all Bitcoin server nodes $\mathrm{N_S}$.
- The attacker logs the messages received from all connected servers, and for each advertised address, say $\mathrm{IP_C}$, the attacker logs the set of servers $\mathrm{N}_{\mathrm{IP_C}}$ that forwarded it to the attacker's machines.
- The attacker designate $\mathrm{N}_{\mathrm{IP_C}}$ as the entry node subset associated to address $\mathrm{IP_C}$.

Note that address $\mathrm{N}_{\mathrm{IP_C}}$ which is announced to the adversary by a node does not have to necessarily correspond to $\mathrm{N}_{\mathrm{IP_C}}$'s entry node. At the same time, as the client does not simultaneously connect to all of its entry nodes, time intervals among the announcement of the same address by its entry nodes may mislead the attacker to a misconception of the network topology.

Assuming that the adversary knows the target address $\mathrm{IP_C}$ before this address reconnects to the network,[1] one can leverage the antiflooding mechanism that Bitcoin has set in place in order to avoid advertising the same address multiple times [6].

Namely, the proposal in [6] ensures that the adversary advertises $\mathrm{IP_C}$ enough before the $\mathrm{IP_C}$ reconnects such that when $\mathrm{IP_C}$ reconnects, the probability that its advertizement is sent to the adversary's machines via a non-entry node is small.

Phase 3: Deanonymizing Bitcoin Transactions: After preventing nodes from using Tor, and after short-listing certain servers as entry nodes for each victim address, deanonymization evolves as follows:

1. The attacker obtains the list $\mathrm{N_S}$ of Bitcoin servers assuming that it is regularly refreshed. Here, the adversary first collects the entire list of peers by querying all his neighbors/known peers with a *getaddr* message. Given this, the attacker collects the list of advertised addresses and adds to the list of Bitcoin servers $\mathrm{N_S}$ every listed address that is online and publicly reachable. This can be easily ascertained by the adversary by trying to establish a TCP connection and exchange *version* messages.

1 Note that this is a common scenario given that plenty of clients use the same machine to perform their Bitcoin transactions, or sit behind $\mathrm{IP_C}$ a NAT.

2. The attacker composes the list C of Bitcoin clients to be deanonymized. Here, the attacker selects a set IP_{C} of nodes that he or she wants to consider in the deanonymization attack. At this point, the attack is agnostic to how the attacker constructs C. For example, the attacker might randomly select IPs advertised throughout the network or obtain C as a set of the IPs used by a user retrieved by an out-of-band channel.

3. The attacker retrieves the entry nodes $\mathrm{N}_{\mathrm{IP}_{\mathsf{C}}}$ of each client $\mathrm{IP}_{\mathsf{C}} \in \mathsf{C}$ when IP_{C} connects to the network, as described above.

4. The attacker keeps monitoring the traffic from servers in $\mathrm{N}_{\mathrm{IP}_{\mathsf{C}}}$ and, by mapping transactions to entry nodes, the attacker can ultimately map transactions to clients. More specifically, the attacker monitors *inv* messages with transaction hashes received over all the established connections, and for each received transaction, it collects the addresses of Bitcoin servers that forwarded the associated *inv* message at each round of transaction advertisements. The attacker finally correlates the sets of servers that advertised each transaction at each round and extracts pairs $\langle \mathrm{entry-node}; \mathrm{transaction} \rangle$ from the matching pairs.

Eventually, the adversary creates a list $\mathrm{List} = \langle \mathrm{IP}_{\mathsf{C}}, \mathrm{Id}_{\mathsf{C}}, \mathrm{PK}_{\mathsf{C}} \rangle$, where IP_{C} is the IP address of a peer or its ISP, Id_{C} distinguishes clients sharing the same IP, and PK_{C} is the address/pseudonym used in a transaction (hash of a public key).

5.3 ENHANCING PRIVACY IN BITCOIN

The significant limitations of Bitcoin with respect to user privacy have pushed the Bitcoin community to design mechanisms that ensure privacy-preserving payments in Bitcoin.

In this section, we overview and analyze proposals for strengthening privacy in Bitcoin. We start by describing *mixing services*, and then proceed to describing other crypto-based privacy extensions of Bitcoin. Mixing services achieve user privacy in a holistic way (i.e., in both the network and protocol layers) without degrading payment performance, but require absolute trust in a third-party. In contrast, cryptographic extensions of Bitcoin eliminate the need for trusted third-parties but tend to be taxing in terms of performance.

5.3.1 Mixing Services

Mixing services play the role of trusted mediators between the users and the Bitcoin system, allowing to some extent the mixing of coins pertaining to several users—thus effectively preventing the public traceability of coin expenditure in the network.

The first mixing services (also called tumblers) mix one's funds with other people's coins with the intention to confuse the trace back to the funds' original source. In traditional financial systems, the equivalent would be moving funds to banks located in countries with strict bank secrecy laws, such as the Cayman Islands, the Bahamas, and Panama.

The operation of a Bitcoin mixer is summarized in Figure 5.1. Users who make use of a mixing service for their payments are usually asked to open an account at that service, which serves as an out-of-band communication channel. Through this channel, the user and the service agree upon a service-owned address to which the user sends his payment. Assuming an honest service that has a nonnegligible number of registered users, there are two models to which such a service can fulfil its purpose:

Coin history resetter Here, the mixer sends back to the user someone else's coins of the same value. Clearly, in this case the user also needs to provide the service with a return address. From this point onward, the user can pay the intended recipient with the fresh coins that he or she received from the service. BitLaundry [18] and Bitcoin Fog [19] are some of the mixers operating under this model. Note that this model does not resist network layer attacks since the user is eventually making the payments.

Payment mediator Here, the mixer keeps the funds, circulates them among its other addresses, and finally pays the recipient of the payment as indicated by the user. Blockchain.info [20] and most online wallets operate in this fashion. Unlike the first model, this variant model resists network layer attacks on the user since the mixer is issuing the payments on behalf of users.

Mixing services usually charge a commission on the payment the user wishes to perform or on the value of the coins exchanged. Though mixing services offer—to a large extent—obfuscation of user payments, their impact on the privacy of user-transactions at the protocol layer has not been thoroughly assessed (up to the time of writing). At the same time, mixing services require absolute trust in the service itself that has the power to steal users' funds.

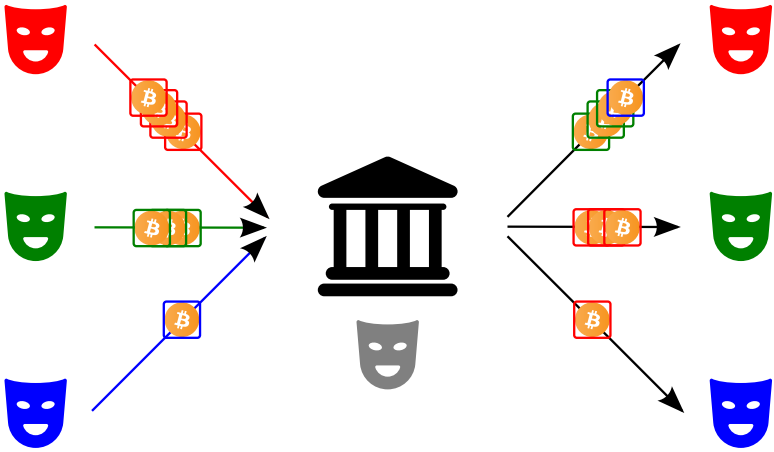

Figure 5.1 A mixing service acting as a payment mediator.

5.3.2 CoinJoin

CoinJoin [21] is a popular technique that aims to mix payments without the need to trust a single trusted party. CoinJoin leverages the ability of recent Bitcoin clients to include in the transactions' inputs originating from different (potentially remote) wallets. This allows a number of different entities to form a single transaction that mixes/shuffles all their coins among themselves. By doing so, these entities effectively hide their addresses among the anonymity set comprised by the clients participating in the CoinJoin protocol (see Figure 5.2). As such, CoinJoin transactions have the same number of outputs as inputs, and as long as the inputs have identical values, and all output addresses are for a single use and receive the same amount, this technique can hide a potential payer among the payers in the input.

Although CoinJoin removes the need to trust a single third-party (e.g., the mixing service), it does require communication and cooperation among multiple parties (i.e., the ones contributing transaction inputs).

Note that although the privacy of payments is stronger the longer the list of transaction inputs (and users creating them), a large number of transaction inputs results in higher computation and communication overhead at transaction creation time (due to multiple signatures generated/communicated by different parties) and computation overhead at verification time (due to multiple signature

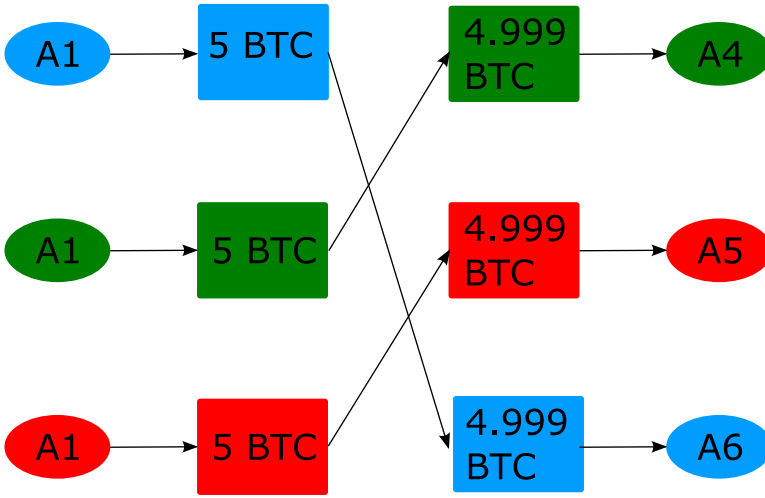

Figure 5.2 Main intuition behind CoinJoin.

validation). Such transactions are therefore typically expected to have higher fees than conventional transactions.

We additionally point out that the CoinJoin protocol implicitly requires that all participants do not misbehave. That is, if one participant does not correctly complete the signing process or double-spends his or her own inputs, then the entire transaction cannot be confirmed. That is, any single participant can easily mount a denial-of-service attack on the CoinJoin protocol.

5.3.3 Privacy-Preserving Bitcoin Protocol Enhancements

The research community features a number of proposals to enhance user privacy in Bitcoin without the need for modifying the original Bitcoin trust model. Examples includes ZeroCoin [8], Extended ZeroCoin [10], and ZeroCash [9]. These constitute the most prominent initiatives that involve the conversion of BTCs to coins that can be spent anonymously.

In the sequel, we first elaborate on the trust model assumed by these protocols, along with their security requirements and guarantees. We then detail how these properties are achieved in each of the aforementioned protocols. Note that we will

solely focus on extensions of the Bitcoin *protocol*—these are orthogonal to the network level linkability of transaction announcements in the Bitcoin network.

5.3.3.1 Model

In the privacy extensions of Bitcoin that we will be discussing in this chapter, we assume that users convert BTCs to *untraceable* or *anonymous* coins (zerocoins in [8], extended zerocoins in [10], and zerocash in [9]) through an operation called Mint. Users subsequently spend these anonymous coins in two possible ways:

- The first payment type consists of converting anonymous coins back to BTCs that are sent to a payee's address; this is the reverse operation of Mint and is referred to by Spend. ZeroCoin only supports this type of spending.
- The second type of payment consists of transforming anonymous coins to other anonymous coins that are under the control of the payee using an operation denoted by Pour. ZeroCash and Extended ZeroCoin support this type of payment.

Given the considered adversarial model, we identify the following security notions for Bitcoin: *balance*, *anonymity*, and *activity unlinkability*. Informally, the balance property requires that an adversary who has legitimately acquired a set of BTCs can spend anonymous coins (to other users) of at most the value of the BTCs he originally owned. The unlinkability property refers to the fact that an adversary should not be able to link two different spending transactions that pertain to a user. Finally, anonymity refers the fact that the spending of a coin should not be linked to a certain conversion transaction (i.e., Mint). Although the spirit of these definitions is the same across all systems presented in this chapter, we provide in the following more formal definitions that are adjusted to the operations performed in each case.

5.3.3.2 Cryptographic primitives

We start by describing a number of cryptographic building blocks that are essential for the operations of privacy-preserving payment systems.

Commitment schemes: Commitment schemes allow a party (committer) to commit to a chosen message (chosen statement) while keeping it hidden to others, with the ability to reveal the committed value later. Commitment schemes consist of three operations:

- $\mathrm{params} \leftarrow \mathrm{Setup(k)}$, a randomized setup algorithm given input by a security parameter k, and outputting public parameters to be used for the computation of the commitment
- $\mathrm{C_r} \leftarrow \mathrm{Commit_{params,r}(m)}$, where a commitment on message m is computed using randomness r
- $\langle \mathrm{r}, \mathrm{v} \rangle \leftarrow \mathrm{DeCommit_{params}(C_r)}$, where the committer opens the commitment to a verifier who can then validate it.

Commitment schemes are designed so that they satisfy the following two properties:

- *Binding:* Given a commitment $\mathrm{C_r}$ to a message m, the committer cannot change the value or statement m they have committed to. That is, they cannot construct another message $\mathrm{m}' \neq \mathrm{m}$ such that $\mathrm{m}' \leftarrow \mathrm{DeCommit_{params}(C_r)}$.
- *Hiding:* Given a commitment value $\mathrm{C_r}$ to a message m, the attacker should not be able to retrieve m.

For the schemes described below, Pedersen commitments [22] will be used, where the *hiding* property is guaranteed against an information theoretic attacker (information theoretically hiding), and the *binding* property is guaranteed against a computational attacker (computationally binding).

Zero Knowledge Proofs of Knowledge and Signatures of Knowledge: Proof of knowledge protocols allow a party (prover) to prove that a statement is true to another party (verifier) (e.g., that a value v is part of a language L). Zero-knowledge proofs of knowledge enable the prover to achieve the same property without revealing anything else to the verifier apart from the fact that the statement holds.

Most privacy-preserving payment systems built atop Bitcoin use zero-knowledge proofs of knowledge (ZKPoK), and in particular protocols where the prover proves knowledge of a committed value v, without leaking any information on v. In this chapter, we leverage the ZKPoK schemes of Schnorr [23] and its extensions [24–27], and convert them to noninteractive *signatures of knowledge* [28] using the Fiat-Shamir heuristic [27].

In signature of knowledge schemes, the knowledge of the secret committed value v is used as signing key. The unforgeability property of these schemes implies that no one but the party that has knowledge of v is able to provide a valid signature

on any message m (i.e., a signature on m) that the signature verification algorithm accepts.

In the following, we will use the notation of Camenisch and Stadler [25,28,29] when referring to these proofs. Namely,

$$\mathrm{NIZKPoK}\{(\alpha, \beta) : h = g^{\alpha} \wedge c = g^{\beta}\}$$

denotes a noninteractive zero-knowledge proof of knowledge of the elements α and β that satisfy both $h = g^{\alpha}$ and $c = g^{\beta}$. All values not enclosed in ()s are known to the verifier. Similarly, the extension

$$\mathrm{ZKSoK[m]}\{(\alpha, \beta) : h = g^{\alpha} \wedge c = g^{\beta}\}$$

indicates a signature of knowledge of α and β on message m.

Accumulators: Cryptographic accumulators basically constitute one-way membership functions; these functions can be used to answer a query whether a given candidate belongs to a set without revealing any meaningful information about the other set members. For the rest of the chapter, we will be referring to the accumulator Acc introduced by Camenisch and Lysyanskaya [30] that supports the following operations:

- $\{\mathrm{N}, \mathrm{u}\} \leftarrow \mathrm{ACC.Setup}(k)$. On input a security parameter k, sample primes p and q (with polynomial dependence on the security parameter), Setup computes the RSA modulus $\mathrm{N} = \mathrm{pq}$ and chooses value $\mathrm{u} \in \mathrm{QR_N}, \neq 1$. Finally, Setup outputs (N, u), which will be denoted by params.
- $\{\mathrm{Acc}\} \leftarrow \mathrm{ACC.Accumulate}(\mathrm{params}, \mathrm{PN})$. On input params and a set of prime numbers $\mathrm{PN} = \{p_1, ..., p_n | p_i \in [\mathrm{A}, \mathrm{B}]\}$, where A, B can be chosen with arbitrary polynomial dependence on k, as long as $2 < \mathrm{A}$ and $\mathrm{B} < \mathrm{A}^2$, $\mathrm{ACC.Accumulate}$ computes the accumulator $\mathrm{Acc} = p_1 p_2 \cdots p_n \ (\mod \mathrm{N})$.
- $\omega \leftarrow \mathrm{ACC.GenWitness}(\mathrm{params}, \mathrm{v}, \mathrm{PN})$. On input $\mathrm{params} = (\mathrm{N}, \mathrm{u})$, a set of prime numbers PN as described above, and a value $\mathrm{v} \in \mathrm{PN}$, the witness ω is the accumulation of all the values in Acc besides v (i.e., $\omega = \mathrm{ACC.Accumulate}(\mathrm{params}, \mathrm{Acc}|\mathrm{v})$).
- $\{0, 1\} \leftarrow \mathrm{ACC.Verify}(\mathrm{params}, \mathrm{Acc}, \mathrm{v}, \omega)$. On input params (N, u), an element v, and witness ω, $\mathrm{ACC.Verify}$ computes $\mathrm{Acc}' \leftarrow \omega^{\mathrm{v}} (\mathrm{mod N})$ and outputs 1 if and only if $\mathrm{Acc}' = \mathrm{Acc}$, v is prime, and $\mathrm{v} \in [\mathrm{A}, \mathrm{B}]$.

Accumulators in [30] satisfy the strong collision-resistance property if the strong RSA assumption is hard. Informally, this ensures that no computational

adversary can produce a pair (v,ω) such that v $\notin$ PN and yet ACC.Verify is satisfied. In fact, in [30], Camenisch and Lysyanskaya describe an efficient ZKPoK scheme that proves that a committed value is contained in an accumulator. We refer to the proof in [30] using the following notation:

$$\mathrm{NIZKPoK}(\nu, \omega) : \mathrm{Acc.Verify}((\mathrm{N}, \mathrm{u}), \mathrm{Acc}, \nu, \omega) = 1.$$

As described before, the Fiat-Shamir heuristic [27] can be used to convert this scheme to a noninteractive signature of knowledge.

ZK-Snarks: ZK-SNARKs [31] is a special kind of succinct noninteractive argument of knowledge (SNARK). The latter is a cryptographic primitive that enables a prover to *argue* that a certain statement satisfies an arithmetic circuit. Similar to *signatures of knowledge*, ZK-SNARKs are publicly verifiable and ensure zero-knowledge properties.

In what follows, we informally define ZK-SNARKs for arithmetic circuit satisfiability. We refer the reader to [31] for a formal definition.

Let $\mathcal{F}$a field, and an arithmetic circuit C with bilinear gates. We say that C is a $\mathcal{F}$-arithmetic circuit when its inputs and outputs are elements residing in $\mathcal{F}$. To capture nondeterminism, nondeterministic circuits with input $x \in \mathcal{F}^n$, are enhanced with an auxiliary input $a \in \mathcal{F}^h$, also known as witness.

The arithmetic circuit satisfiability problem for an $\mathcal{F}$-arithmetic circuit, $\mathrm{C} : \mathcal{F}^n \times \mathcal{F}^h \rightarrow \mathcal{F}^l$, relates to the finding of input values $\langle x, a \rangle \in \mathcal{F}^n \times \mathcal{F}^h$, such that the output of the gates of C output 0 and can be well represented by the relation

$$\mathcal{R}_{\mathrm{C}} = (x, a) \in \mathcal{F}^n \times \mathcal{F}^h : \mathrm{C}(x, a) = 0^l,$$

and its language:

$$\mathcal{L}_{\mathrm{C}} = x \in \mathcal{F}^n : \exists a \in \mathcal{F}^h : \mathrm{C}(x, a) = 0^l.$$

Assuming a field $\mathcal{F}$, a (publicly verifiable preprocessing) ZK-SNARK for an $\mathcal{F}$-arithmetic circuit satisfiability consists of the following algorithms:

- $\langle \mathrm{pk}, \mathrm{vk} \rangle \leftarrow \mathrm{KeyGen}(\lambda, \mathrm{C})$, which on input a security parameter λ and circuit C, generates a proving key pk and verification key vk, that can henceforth be considered as the public parameters of the system.
- $\pi \leftarrow \mathrm{Prove}(\mathrm{pk}, x, a)$, where with input a proving key pk, a pair of input x, and witness a, where $(\mathrm{x}, \mathrm{a}) \in \mathcal{R}_{\mathrm{C}}$, the prover outputs a noninteractive proof π, that input x is part of C satisfiability language $\mathcal{L}_{\mathrm{C}}$.

- $\text{accept/reject} \leftarrow \text{Verify}(\text{vk}, x, l)$ where the verifier using verification key vk, and proof π for input x, confirms that $x \in \mathcal{L}_{\text{C}}$ (accept), or outputs an error message (reject).

Informally, ZK-SNARKs satisfy the following properties:

Completeness Here, it is required that for any statement that is a member of the circuit language, $\mathcal{L}_{\text{C}}$, an honest prover running Prove should be able to convince the verifier; that is, Verify should not output an error message.

Succinctness This property ensures that Prove and Verify run in a number of steps polynomial to the security parameter λ.

Proof of knowledge This property ensures that the verifier can only accept proof outputs of a computationally bounded prover running Prove if and only if the prover knows a valid witness for the provided input.

Perfect zero knowledge Based on this property, no information on the secret statements or witnesses is revealed to the verifier. More formally, zero knowledge requires that for a given Verify transcript there is a simulator simulating KeyGen and Prove whose outputs cannot be distinguished from honestly executing KeyGen and Prove.

5.3.3.3 ZeroCoin

ZeroCoin [8] is one of the first cryptographic extensions of Bitcoin that aims at enhancing its privacy. It was introduced by Miers et al. in [8] to remedy the fact that Bitcoin enables the public tracing of coin expenditure in the network.

ZeroCoin leverages ZKPoP protocols and cryptographic accumulators to implement a cryptographic mixer. More specifically, ZeroCoin *resets* the history of a BTC by transforming it into a ZeroCoin coin referred to in the sequel by ZC. During this conversion, ZCs are added to a cryptographic coin mixer (essentially an accumulator) that is publicly available.

The resulting ZCs can be proven in zero-knowledge to have originated from valid and unspent BTCs (i.e., that they are part of the unspent subset of coins in the mixer). In this way, an entity is prevented from linking a transaction with the BTC (and the corresponding address) that generated the zc used therein. In other words, ZeroCoin ensures that the origin of a zc is hidden among all BTCs that were converted to eZCs. In addition, ZeroCoin preserves the payment security guarantees of Bitcoin (e.g., double-spending resistance). That is, no party can spend more BTCs

or ZCs than the ones he or she possesses.

Protocol specification: ZeroCoin consists of the following procedures: Setup, where the system parameters are set, the Mint operation, where a Bitcoin (BTC) is converted to a ZeroCoin (ZC), the Spend operation, where a ZC is spent/deposited to Bitcoin address, and automatically converted to a fraction of a BTC, and the Verify operation, through which Bitcoin peers can verify the validity of the ZC transaction and include it in a block. More specifically,

- $\mathrm{params} \leftarrow \mathrm{ZC.Setup}(1^k)$, where k is the security parameter. params include a group $\mathcal{G}$ of RSA modulus and of order o and its generators $g, h : \langle g \rangle = \langle h \rangle = \mathcal{G}$.

- $\{\mathrm{pk_{zc}}, \mathrm{sk_{zc}}\} \leftarrow \mathrm{ZC.Mint}(\mathrm{params}, \mathrm{btc})$, through which a BTC btc is converted to a fixed ZeroCoin value zc with secret information $\mathrm{sk_{zc}}$ and public information $\mathrm{pk_{zc}}$. On can see this operation as a conventional Bitcoin transaction, where the converted btc is the input and the output is $\mathrm{pk_{zc}}$, that uniquely defines the generated zc. Thus, the miners check again whether btc is valid (i.e., whether it is owned by the address that has signed the transaction and it has not been spent before by that address). If btc is valid, then the $\mathrm{pk_{zc}}$ included in the transaction is included in the next block. All the zc-public information that are included in blocks are added in an accumulator Acc.[2] Public $\mathrm{pk_{zc}}$ included in the transaction is a Pedersen commitment to a serial number s of the form

$$\mathrm{zc} = \mathrm{Commit}_r^{PED}(s) = g^s \cdot h^r,$$

where $r \leftarrow_R \mathcal{G}$. The secret information related to zc is set to $\mathrm{sk_{zc}} = (s, r)$ and is partially revealed in the $\mathrm{ZC.Spend}$ operation.

- $\{\pi, s\} \leftarrow \mathrm{ZC.Spend}(\mathrm{params}, \mathrm{sk_{zc}}, \mathrm{pk_{zc}}, \mathrm{Acc})$, which is performed by the user who wishes to spend a zc with public information $\mathrm{pk_{zc}}$. To do so, the user reveals s and computes a ZKPoK π that s corresponds to a zc that has been confirmed in a block (i.e., is part of the accumulator). As such, ZeroCoin can be integrated in existing Bitcoin transactions without modifications: (s, π) constitute the inputs to a standard Bitcoin transaction, which takes as output a regular Bitcoin address. This special transaction contains a signature of knowledge

2 Note that the accumulator value is computed by the peers locally. Given the public parameters of the accumulator, and the confirmed ZC, each peer can locally compute the accumulator value at any point in the blockchain.

produced by π, and is released in the network to be confirmed in a block.

- $\{\pi, s\} \leftarrow \mathrm{ZC.Verify}(\mathrm{params}, s, \pi, \mathrm{Acc})$, which can be executed by every peer in the Bitcoin network to verify whether the signature of knowledge deriving from ZKPoK π is valid and that the serial number s has not been used before (and thus that the corresponding zc has not been spent before). Verified pairs of π, s are included by the Bitcoin miners in the next generated block, to establish their validity to all Bitcoin peers.

Limitations: In ZeroCoin, each ZC corresponds to exactly one (or a predefined number of) BTC; transactions whose values are larger than one BTC would therefore result in several back-to-back ZeroCoin transactions. This results in considerable overhead in propagating the corresponding transactions in the network and including them in valid blocks.

Furthermore, while ZeroCoin prevents the traceability of constant value coins, it does not conceal the transaction amounts; multiple ZC spendings for the same payment are likely to be linked in time and the total amount per payment can be retrieved (since each ZC corresponds to a single BTC).

Furthermore, ZeroCoin does not hide the total number of BTCs redeemed by Bitcoin addresses when the owners of these addresses transform their ZCs back to BTCs. As mentioned earlier, recent studies [3] have shown that tracing coin expenditure is not the only source of information leakage in Bitcoin. More specifically, Androulaki et al. have shown in [3] that behavior-based clustering algorithms can be used to acquire considerable information about the user profiles in Bitcoin. These algorithms mainly leverage user spending patterns, such as transaction amounts, transaction times, and so on in order to profile users. Clearly, ZeroCoin does not prevent such analysis, since the transaction times, transaction amounts, and address balances can still be derived from the blockchain.

5.3.4 Extending ZeroCoin: EZC and ZeroCash

In what follows, we present a number of proposals that address some of the limitations of ZeroCoin. In particular, we present EZC and ZeroCash, two independently designed systems, that support transactions where one or more anonymous coins can be spent in the form of fresh anonymous coins. In these proposals, the value of the coins spent is hidden from the rest of the network, and the coin recipient has complete control over the spent coins.

Throughout the rest of this chapter, we present Extended ZeroCoin that, as its name suggests, builds on top of ZeroCoin. We then move on to ZeroCash that constitutes a ZK-SNARK based implementation of *decentralized anonymous payment systems* [9].

Besides overviewing the operations of EZC and ZeroCash, we will additionally compare these proposals with ZeroCoin.

5.3.4.1 Extended ZeroCoin

Extended ZeroCoin is an enhanced variant of ZeroCoin, dubbed EZC, that enables the expenditure of transaction amounts exceeding the value of 1 BTC while concealing these amounts from the network. Similar to ZeroCoin, EZC leverages accumulators and ZKPoK protocols to construct multi-valued ZCs. The resulting coins can be either spent as regular Bitcoins or can be spent directly in the network without the need to transform them back to equivalent value BTCs. Additionally, EZC transactions require that the transaction amounts only need to be revealed to the payment sender and recipient, and not to the rest of the participants of the system. It is noteworthy, that since EZC coins (henceforth referred to as eZC) do not have to be exchanged back to BTCs, this scheme also prevents the leakage of the balances of those addresses who opt out from exchanging their coins to BTCs.

Figure 5.3 compares the main operations of EZC to ZC.

Similar to ZeroCoin, a fundamental assumption for the security of EZC is that the public parameters of the system are generated honestly (i.e., in a way that complies with the security definition of the primitives). As mentioned earlier, EZC supports Mint operation, denoted by EZC.Mint, where arbitrary valued BTCs are converted to an anonymous EZC coin (eZC), Spend, denoted by EZC.Spend, where eZCs are spent in the form of BTCs, and Pour, denoted by EZC.Pour, where eZCs are spent in the form of freshly generated eZCs.

Similar to conventional Bitcoin payments, the validity of each transaction is verified by the network peers, who subsequently work toward confirming valid transactions in blocks. For simplicity, we will refer to a transaction resulting from operation O by an O transaction (e.g., a Mint transaction refers to a transaction output by the Mint procedure).

Overview of EZC**:** During setup, the system runs the Setup of a dynamic accumulator scheme (i.e., Acc.Setup). Let this accumulator be denoted by $\mathrm{Acc}_{\mathrm{EZC}}$. $\mathrm{Acc}_{\mathrm{EZC}}$ includes all properly minted and confirmed eZCs.

In EZC, the EZC.Mint transaction is constructed in a similar way to the that of ZeroCoin, and consists of an input (in BTCs) and an output that includes

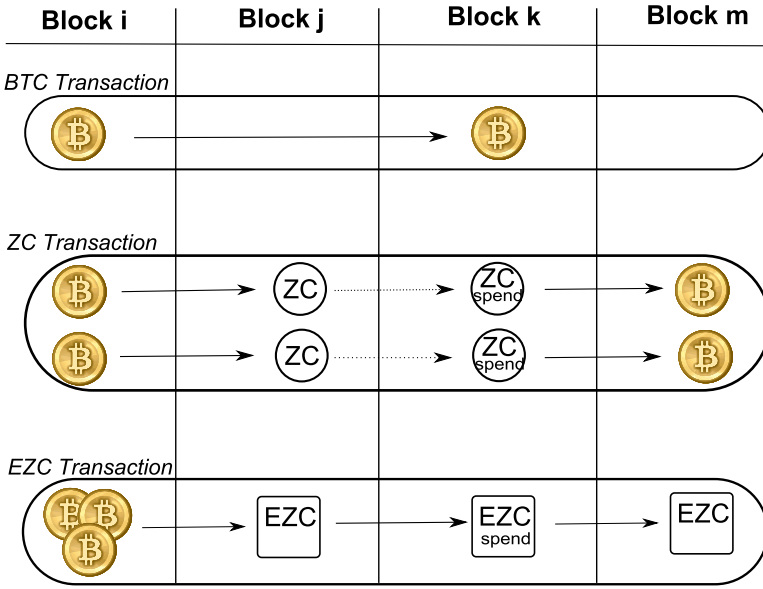

Figure 5.3 Comparison between EZC and ZC. Each ZC corresponds to a single BTC and can only be spent in the form of BTCs. EZC, on the other hand, enables the construction of a multivalued eZC and can be spent in eZCs without the need to transform them back to BTCs.

information related to the created eZCs. However, unlike ZeroCoin, the coins generated with EZC.Mint can accommodate any payment value *val* (e.g., in BTCs).

The outputs of this transaction consist of a commitment $\mathrm{cmtr}(ser, val)$ to *val* and to a random serial number *ser*, and of a proof of c's correctness. As we show later, *val* is revealed by the peer who runs EZC.Mint to all the peers in the network but *ser* is kept private until the eZC that is minted is spent. The constructed transaction is signed using the Bitcoin private keys corresponding to the input address(es). The correctness of EZC.Mint transactions is verified by the rest of the peers in the network and correct EZC.Mint transactions are included in the longest blockchain. After confirmation of an EZC.Mint transaction, the commitment $\mathrm{cmtr}(ser, val)$ is added (using the Acc.Add procedure) as a member of $\mathrm{Acc}_{\mathrm{EZC}}$.

To spend an eZC in the form of BTCs of value *val*, the eZC-owner—who knows *ser* and the opening of $\mathrm{cmtr}ser, val$—constructs a proof π that *ser* indeed

corresponds to a commitment to a value *val* that is a member of $\mathrm{Acc_{EZC}}$. He or she then engages in EZC.Spend by constructing a transaction signed with a signature of knowledge of π. The resulting transaction has a similar structure to a Bitcoin transaction, where π is used as input and its output can distribute *val* to one or more Bitcoin addresses. Note that for peers to be able to verify the correctness of such a transaction, the serial *ser* must be revealed; nevertheless, no entity is able to link the EZC.Spend transaction to a particular EZC.Mint transaction, and, thus to the btc-s that created it.

On the other hand, to spend an eZC in the form of anonymous eZCs of the same or smaller value *val'*, the payer reveals *ser*, and engages in a similar set of operations as in EZC.Spend to construct a proof π that the coins involved in the transaction are properly created. To accommodate for the creation of the recipient's eZC, the two parties construct a commitment $\mathrm{cmt}r'(val', ser')$ to a freshly generated serial *ser'* and to the payment amount *val'*. π contains a proof that the payment amount (*val*) does not exceed the value of the payer's coin (*val'*). Finally, π is used to produce a signature of knowledge on the output commitment $\mathrm{cmt}r'(val', ser')$ into an EZC.Pour transaction, which is released to the network with *ser'*. As soon as the latter is confirmed, $\mathrm{cmt}r'\, val', ser'$ is added by the peers to $\mathrm{Acc_{EZC}}$. Note that *val'* is kept private between the payer and the payment recipient, while *ser'* and the opening of $\mathrm{cmt}r'\, val', ser'$ is only known to the payment recipient.

Protocol specification: In what follows, we detail the operations in EZC. In the sequel, we denote the public information associated with an eZC by $\mathrm{pub_{eZC}}$and the corresponding private information by $\mathrm{sec_{eZC}}$.

- $\mathrm{params} \leftarrow \mathrm{EZC.Setup}(\lambda)$, the setup of EZC, running with input a security parameter λ and produces the system parameters params. More specifically, EZC.Setup runs $\mathrm{Acc_{EZC}.Setup}(\lambda)$ to obtain (N, u), and generates primes p, q, such that $\mathrm{p} = 2^f \cdot \mathrm{q} + 1, f \geq 1$. It then picks g, h, and w such that

$$\mathcal{G} = \langle g \rangle = \langle h \rangle = \langle w \rangle \subset Z_{\mathrm{q}}^*.$$

  Finally, it sets

$$\mathrm{params} = \{\mathrm{N}, \mathrm{u}, \mathrm{p}, \mathrm{q}, g, h, w\}.$$

- $\pi, \mathrm{pub_{eZC}}, u(\mathrm{sec_{eZC}})\} \leftarrow \mathrm{EZC.Mint}(\mathcal{I}_{\mathrm{BTC}}, \mathrm{params})$, that performs the conversion of one or more BTCs to eZCs. This operation is executed by the owner u of a set of BTCs $\mathcal{I}_{\mathrm{BTC}}$ that are converted into an eZC with public information $\mathrm{pub_{eZC}}$and private information $\mathrm{sec_{eZC}}$. Here, u picks $ser, r \leftarrow_R \mathcal{G}$, where *ser* is

the serial number of the generated eZC computes $\mathrm{pub}_{\mathrm{eZC}} = \mathrm{cmt}_{\mathit{ser}, \mathit{val}}$, and a ZKPoK π asserting that $\mathrm{pub}_{\mathrm{eZC}}$ is correctly formed:

$$\mathrm{NIZKPoK}(\alpha, \beta) : \mathrm{pub}_{\mathrm{eZC}} = g^{\alpha} \cdot h^{\mathit{val}} \cdot w^{\beta}.$$

Note that the $\mathrm{EZC.Mint}$ transaction is constructed similarly to a standard Bitcoin transaction, where the BTCs are used as input and $\langle \mathrm{pub}_{\mathrm{eZC}}, \pi \rangle$ is used as output. Subsequently, peers verify that $\mathrm{pub}_{\mathrm{eZC}}$ is correctly formed by running the ZKPoK verification protocol for π and by confirming that the input BTCs were not spent in the past, as is currently done in Bitcoin. If the transaction is deemed valid by the majority of the computation power of the network, $\mathrm{pub}_{\mathrm{eZC}}$ is included in the blockchain, and $\mathrm{pub}_{\mathrm{eZC}}$ is considered as a valid member of the public accumulator $\mathrm{Acc}_{\mathrm{EZC}}$. User u's private output is $\mathrm{sec}_{\mathrm{eZC}} = \langle \mathit{ser}, r \rangle$, while $\{\mathrm{sec}_{\mathrm{eZC}}, \mathit{val}\}$ is stored in u's local memory.

- $\mathcal{O}_{\mathrm{BTC}} \leftarrow \mathrm{EZC.Spend}[\mathrm{params}, \mathit{ser}_S, u_S(\mathrm{sec}_{\mathrm{eZC}_S}, \mathrm{pub}_{\mathrm{eZC}_S})]$ that is performed to spend eZCs back to BTCs. This operation results in a transaction that uses as input ($\mathrm{sec}_{\mathrm{eZC}_S}$,$\mathrm{pub}_{\mathrm{eZC}_S}$) and spends them in BTCs of value val to a set of Bitcoin addresses $\mathcal{I}_{\mathrm{BTC}}$. Here, the sender u_S first computes the public accumulator value $\mathrm{Acc}_{\mathrm{EZC}}$ locally by running $\mathrm{Acc}_{\mathrm{EZC}}.\mathrm{Accumulate}(\mathrm{N}, \mathrm{u}, \{\mathrm{pub}_{\mathrm{eZC}}\}_{\forall \in \mathsf{pubLo}}$ for the set of EZC commitments that have appeared in the longest blockchain so far. The sender retrieves $\mathrm{sec}_{\mathrm{eZC}_S}$ from his or her local memory and runs:

$$\mathrm{Acc}_{\mathrm{EZC}}.\mathrm{GenWitness}(\mathrm{params}, \{\mathrm{pub}_{\mathrm{eZC}}\}_{\forall \in \mathsf{pubLog}}, \mathrm{pub}_{\mathrm{eZC}_S}),$$

to compute the witness w_S for $\mathrm{pub}_{\mathrm{eZC}_S}$'s membership in $\mathrm{Acc}_{\mathrm{EZC}}$. Furthermore, u_S computes a ZKPoK π to show that ser_S corresponds to an eZC whose public information (here, $\mathrm{pub}_{\mathrm{eZC}_S}$) is part of $\mathrm{Acc}_{\mathrm{EZC}}$, and that it corresponds to a value val. Subsequently, it converts π to a signature of knowledge on $\mathcal{O}_{\mathrm{BTC}}$:

$$\mathrm{ZKSoK}[\mathcal{O}_{\mathrm{BTC}}](\alpha, \beta, \gamma) :$$

$$\alpha = g^{\mathit{ser}_S} h^{\mathit{val}} w^{\beta} \wedge \mathrm{Acc.Verify}(\mathrm{N}, \mathrm{u}, \mathrm{Acc}_{\mathrm{EZC}}, \alpha, \gamma) = 1\}.$$

Finally, u_S announces the corresponding signature within a transaction to the EZC network, which, after confirming the transactions' correctness, it includes the latter in a block.[3]

3 Note that if fees are to be supported, the fee amount should be explicitly stated within the message in the signature (transaction).

- $\{\langle \mathrm{pub}'_{\mathrm{eZC}_S}, \mathrm{pub}_{\mathrm{eZC}_R}, u_S(\mathrm{sec}'_{\mathrm{eZC}_S}), u_R(\mathrm{sec}_{\mathrm{eZC}_R})\rangle / \perp\} \leftarrow \mathrm{EZC.Pour}$ $(\mathrm{params}, \mathit{ser}_S, \mathrm{Acc}_{\mathrm{EZC}}, u_S(\mathit{val}\, e_R, \mathit{val}_S, r_S, \mathit{ser}'_S, r'_S), u_R(\mathit{val}_R, \mathit{ser}_R, r_R))$, where an eZC is spent to one or more other eZCs. This is an interactive operation between a payment sender u_S and a payment recipient u_R. It takes as input the information associated with an eZC of u_S (e.g., $\mathrm{sec}_{\mathrm{eZC}_S}$ and $\mathrm{pub}_{\mathrm{eZC}_S}$) and spends it in the form of a new eZC that belongs to u_R, eZC_R; if change amounts should be incorporated, this operation outputs additionally a eZC that belongs to u_S denoted by eZC'_S.

  Here, u_R's private input $\mathrm{sec}_{\mathrm{eZC}R}$ consists of a serial number ser_R for his or her new coin and a random number $\mathrm{r_S} \in Z_{(p-1)/2}$ that will be used in her new coin's commitment. Assuming that $\langle \mathit{ser}_S, r_S, \mathit{val}_S\rangle$ is the entry for eZC_S in u_S's local memory, u_S announces the serial number ser_S of eZC_S and privately contributes r_S and val_S to compute the eZC_S validity proof as in EZC.Spend. Finally, u_S's private input includes the values $\langle \mathit{ser}'_S, r'_S\rangle$ used for $\mathrm{eZC}_{S'}$'s construction. We emphasize that $\mathrm{sec}_{\mathrm{eZC}R}$ should be kept private even toward u_S so the latter is not able to trace further spendings of eZC_R.

  In more detail, the payment sender u_S and recipient u_R engage in the following operations:

  1. u_S proves the validity of eZC_S. This is achieved as follows. u_S runs $\mathrm{ACC.Accumulate}(\mathrm{N}, \mathrm{u}, \{\mathrm{pub}_{\mathrm{eZC}}\}_{\forall \mathsf{pubLog}})$ for the set of eZC-commitments that appear in the EZC-blockchain to compute the current public accumulator value $\mathrm{Acc}_{\mathrm{EZC}}$. Subsequently, u_S runs $\mathrm{ACC.GenWitness}(\mathrm{params}, \mathrm{pub}_{\mathrm{eZC}_S}, \mathrm{sec}_{\mathrm{eZC}_S}, \mathrm{Acc}_{\mathrm{EZC}})$ to extract a witness w_S that eZC_S has been confirmed in a block. Then, u_S computes π as described in the previous section; that is:

$$\mathrm{NIZKPoK}(\alpha, \beta, \gamma, \delta) :$$

$$\alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathrm{ACC.Verify}(\mathrm{N}, \mathrm{u}, \mathrm{Acc}_{\mathrm{EZC}}, \alpha, \delta) \;=\; 1.$$

  2. u_S mints eZC'_S. This is achieved as follows. For the transaction output side, u_S picks $\mathit{ser}'_S \leftarrow_R Z_{\mathrm{p}-1}$, and $r'_S \leftarrow_R Z_{\mathrm{p}-1}$, and computes the public information associated to eZC'_S, as $\mathrm{pub}'_{\mathrm{eZC}_S} = g^{\mathit{ser}'_S} h^{\mathit{val}'_S} w^{r'_S}$, where $\mathit{val}'_S = \mathit{val}_S - \mathit{val}_R$ is the change value. Finally, u_S updates π to include a proof that $\mathrm{pub}'_{\mathrm{eZC}_S}$ is properly formed:
  $\mathrm{NIZKPoK}(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta) :$

$$\alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathrm{ACC.Verify}(\mathrm{N}, \mathrm{u}, \mathrm{Acc}_{\mathrm{EZC}}, \alpha, \delta) \;=\; 1 \wedge$$

$$\mathrm{pub}'_{\mathrm{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \ \wedge \ \zeta \in Z_{(\mathrm{p}-1)/2}.$$

3. u_S needs to enable u_R to privately mint for the payment coin eZC_R. For that purpose, u_S picks $r_{SR} \leftarrow_R Z_{p-1}$, computes the auxiliary token $\ell_{SR} = h^{val_R} w^{r_{SR}}$, and updates ZKPoK π so as to include a proof of correctness of ℓ_{SR}, and sends it to u_R along with r_{SR}:

$$\begin{aligned}&\mathrm{NIZKPoK}(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta) : \\ &\alpha = g^{ser_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \ \mathrm{ACC.Verify}(\mathrm{N}, \mathrm{u}, \ \mathrm{Acc}_{\mathrm{EZC}}, \ \alpha, \ \delta) \\ = \ 1 \wedge \mathrm{pub}'_{\mathrm{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \wedge \zeta \in Z_{(\mathrm{p}-1)/2} \wedge \ell_{SR} \cdot \mathrm{pub}'_{\mathrm{eZC}_S} = g^{\epsilon} \cdot h^{\beta} \cdot w^{\theta}.\end{aligned}$$

4. u_R mints eZC_R: u_R picks $ser_R \leftarrow_R Z_{\mathrm{p}-1}$, and $r_R \leftarrow_R Z_{\mathrm{p}-1}$ and computes $\mathrm{pub}_{\mathrm{eZC}} R = g^{ser_R} \cdot h^{val_R} \cdot w^{r_R}$; he or she extends π to include proof of correctness of $\mathrm{pub}_{\mathrm{eZC}_R}$ and converts it into a ZKSoK to sign ser_S and $\mathrm{pub}_{\mathrm{eZC}} R$ and $\mathrm{pub}_{\mathrm{eZC}} S$ in the longest blockchain resulting into another transaction:

$$\mathrm{ZKSoK}[ser_S, \mathrm{pub}_{\mathrm{eZC}_R}](\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \mu, \rho) :$$

$$\begin{aligned}\alpha = g^{ser_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathrm{ACC.Verify}(\mathrm{N}, \mathrm{u}, \ \mathrm{Acc}_{\mathrm{EZC}}, \ \alpha, \ \delta) \ = \ 1 \\ \wedge \mathrm{pub}'_{\mathrm{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \ \wedge \ \ell_{SR} \cdot \mathrm{pub}'_{\mathrm{eZC}_S} \\ = g^{\epsilon} \cdot h^{\beta} \cdot w^{\theta} \wedge \ell_{SR} \\ = h^{\iota} \cdot w^{\kappa} \ \wedge \ \mathrm{pub}_{\mathrm{eZC}_R} \\ = g^{\mu} \cdot h^{\iota} \cdot w^{\rho} \wedge \zeta \in Z_{(p-1)/2} \wedge \iota \in Z_{(p-1)/2}.\end{aligned}$$

The resulting transaction is announced to the network of EZCpeers, who upon verification of its correctness work toward its inclusion into a block. After such a transaction is included into a block, $\mathrm{pub}_{\mathrm{eZC}} R$ and $\mathrm{pub}_{\mathrm{eZC}} S'$ are considered members of $\mathrm{Acc}_{\mathrm{EZC}}$.

The security of Extended ZeroCoin rests upon the standard security assumptions of the underlying signature of knowledge, and dynamic accumulator schemes used within. It is easy to see that a spending operation in Extended ZeroCoin costs more in terms of computation that a ZeroCoin spending. However, note that in ZeroCoin direct spendings from ZeroCoins to ZeroCoins is not possible, and one would

need to convert a ZeroCoin to a Bitcoin and the latter to a ZeroCoin to be able to always maintain its coin belongings in secrecy. If one takes into account that a ZeroCoin has a fixed value, and one would need to perform multiple spendings to perform a single payment worth of few Bitcoins, the overall cost of maintaining anonymous payments in ZeroCoin may end up being comparable to the one of Extended ZeroCoin.

5.3.4.2 ZeroCash

ZeroCash offers a more practical implementation of Extended ZeroCoin capabilities. Namely, ZeroCash implements a decentralized anonymous system (DAP) by leveraging ZK-SNARK (see Section 5.3.3.2 for more detail).

In what follows, we offer an informal description of DAP model, outline its security provisions, and then provide an intuition of how ZeroCash leverages ZK-SNARKs to implement it.

Decentralized Anonymous Payments. Decentralized anonymous payments propose a system that leverages a ledger to announce messages associated to the DAP system payments. The ledger system is referred to as *Basesystem* while the associated currency is called *Basecoin*.

Similar to Extended ZeroCoin, each coin coin in DAP systems is strongly associated with the coin's serial number ser_{coin}, the coin value val_{coin} (i.e., the denomination of the value of that coin in Basecoins), and a commitment value $\mathrm{cmt}_{\mathrm{coin}}$ (i.e., a string uniquely associated to coin's creation). For example, in the case of EZC, this corresponds to the string that appears on the ledger once an eZC is minted. In addition, DAP systems consider a fixed association of a coin with an address key-pair: $\langle \mathrm{apk}_{\mathrm{coin}}, \mathrm{ask}_{\mathrm{coin}} \rangle$. Clearly, the owner of coin is assumed to be in possession of $\mathrm{ask}_{\mathrm{coin}}$.

DAP systems consider two types of transactions: Mint and Pour. Both transactions rely on cryptographic operations in order to enable the creation of new DAP coins from Basecoins (Mint) or the transfer of DAP coins to other DAP coins (Pour).

More specifically, a Mint transaction of a DAP coin coin denoted by tx_{mint} is the tuple $\langle \mathrm{cmt}_{\mathrm{coin}}, val_{\mathrm{coin}} \rangle$. Similarly, a Pour transaction, denoted by tx_{pour}, represents the pouring of two existing DAP coins

$$\mathrm{coin}_{\mathrm{i}}^{\mathrm{old}} = \{\langle ser_{\mathrm{coin}_{\mathrm{i}}^{\mathrm{old}}}, val_{\mathrm{coin}_{\mathrm{i}}^{\mathrm{old}}}, \mathrm{cmt}_{\mathrm{coin}_{\mathrm{i}}^{\mathrm{old}}}, \rangle\}_{i=0,1},$$

to two fresh ones $\{\mathrm{coin}_j^{\mathrm{new}}\}_{j=0,1}$ with associated commitments $\{\mathrm{cmt}_{\mathrm{coin}_j^{\mathrm{new}}}\}_{j=0,1}$ and has the following form:

$$\langle \mathrm{root}, \mathit{ser}_{\mathrm{coin}_1^{\mathrm{old}}}, \mathit{ser}_{\mathrm{coin}_2^{\mathrm{old}}}, \mathrm{cmt}_{\mathrm{coin}_1^{\mathrm{new}}}, \mathrm{cmt}_{\mathrm{coin}_2^{\mathrm{new}}}, \mathrm{pub}, \mathrm{info}, * \rangle.$$

Here *info* includes information about the coin recipient and root represents the current status of the underlying ledger. For example, root can correspond to the root of the Merkle tree composed by the commitments of DAP coins that have been generated and included in Basesystem blocks so far.

Within a DAP system, one needs to reference the state of the DAP system itself and the underlying system ledger. For example, a Pour transaction takes as input a representation of the set of DAP coins that have so far been created (among others). The DAP system leverages the series of (ordered) blocks that have so far been advertised within the underlying Basesystem ledger and considers the Merkel tree over the set of DAP coin commitments advertised in DAP Mint or Pour transactions in the Baseledger. We denote this by $\mathrm{root}_{\mathrm{cmt}}^n$ assuming a Baseledger of length n.

Decentralized anonymous payment systems assume the following operations:

- $\mathsf{params} \leftarrow \mathrm{Setup}(\lambda)$: This algorithm takes as input the security parameter, λ, and outputs the system public parameters. Note that it is imperative that this algorithm is initially executed by a trusted party. This party would no longer be needed as soon as the generated public parameters are announced to all entities in the system.

- $\langle \mathrm{apk}_{\mathrm{coin}}, \mathrm{ask}_{\mathrm{coin}} \rangle \leftarrow \mathrm{CreateAddress}(\mathsf{params})$. $\mathrm{CreateAddress}$ generates a new address key pair $\mathrm{apk}_{\mathrm{coin}}, \mathrm{ask}_{\mathrm{coin}}$. Similar to Bitcoin addresses, the secret part $\mathrm{ask}_{\mathrm{coin}}$ is known, maintained privately by the address owner, and used by the latter to redeem any coins owned by the address public key $\mathrm{apk}_{\mathrm{coin}}$.

- $\{\mathrm{tx}_{\mathrm{mint}}, \mathrm{coin}\} \leftarrow \mathrm{Mint}(\mathsf{params}, \mathit{val}_{\mathrm{coin}}, \mathrm{apk}_{\mathrm{coin}})$, where a new DAP coin is generated with value $\mathit{val}_{\mathrm{coin}}$ and is owned by the address $\mathrm{apk}_{\mathrm{coin}}$ assuming the system parameters params. The Mint operation results in the creation of a new coin coin and the associated transaction.

- $\{\mathrm{tx}_{\mathrm{pour}}, \mathrm{coin}_1^{\mathrm{new}}, \mathrm{coin}_2^{\mathrm{new}}\} \leftarrow \mathrm{Pour}(\mathsf{params}, \mathrm{root}_{ctm}^{new}, \mathrm{coin}_0^{\mathrm{old}}, \mathrm{coin}_1^{\mathrm{old}}, \{\mathrm{cmt}_{\mathrm{coin}_i^{\mathrm{new}}}, \mathit{val}_{\mathrm{coin}_i^{\mathrm{new}}}, \pi_{\mathrm{coin}_i^{\mathrm{new}}}\}_{i=0,1}, \mathrm{info})$. Here, two existing ZeroCash coins $\mathrm{coin}_0^{\mathrm{old}}$ and $\mathrm{coin}_1^{\mathrm{old}}$ are spent into two freshly generated coins $\mathrm{coin}_0^{\mathrm{new}}$ and $\mathrm{coin}_1^{\mathrm{new}}$ of values $\mathit{val}_{\mathrm{coin}_0^{\mathrm{new}}} and \mathit{val}_{\mathrm{coin}_1^{\mathrm{new}}}$, respectively. Additionally, this operation takes as input $\mathrm{root}_{\mathrm{cmt}}^{\mathrm{new}}$ that constitutes the current representation of the ledger state and $\pi_{\mathrm{coin}_i^{\mathrm{new}}}$, for $i = 0, 1$ that prove that $\mathrm{coin}_0^{\mathrm{old}}$ and $\mathrm{coin}_1^{\mathrm{old}}$

are included in the root. For example $\mathrm{root}^{\mathrm{new}}$ could constitute the root of the Merkel tree consisting of all DAP coin commitments that have been added to the ledger up to the point where $\mathrm{coin}_0^{\mathrm{new}}$ and $\mathrm{coin}_1^{\mathrm{new}}$ are generated. In this case, $\{\pi_{\mathrm{coin}_i^{\mathrm{new}}}\}_{i=0,1}$ would constitute the corresponding sibling (authentication) paths of $\mathrm{cmt}_{\mathrm{coin}_0^{\mathrm{old}}}$ and $\mathrm{cmt}_{\mathrm{coin}_1^{\mathrm{old}}}$ with respect to root of the Merkle tree. The outputs of this operation are $\mathrm{coin}_0^{\mathrm{new}}$, $\mathrm{coin}_1^{\mathrm{new}}$, and the associated transaction.

In short, the input values $\mathit{val}_{\mathrm{coin}_0^{\mathrm{new}}}$ and $\mathit{val}_{\mathrm{coin}_1^{\mathrm{new}}}$ correspond to the new value of the two freshly generated coins that are to be deposited in the two input address public keys $\mathrm{apk}_{\mathrm{coin}_0^{\mathrm{new}}}$ and $\mathrm{apk}_{\mathrm{coin}_1^{\mathrm{new}}}$, respectively.

$\mathit{val}_{\mathrm{pub}}$ specifies the amount to be publicly spent (i.e., allocated to transaction fees). Overall, it should hold that

$$\mathit{val}_{\mathrm{pub}} + \mathit{val}_{\mathrm{coin}_0^{\mathrm{new}}} + \mathit{val}_{\mathrm{coin}_1^{\mathrm{new}}} = \mathit{val}_{\mathrm{coin}_0^{\mathrm{old}}} + \mathit{val}_{\mathrm{coin}_1^{\mathrm{old}}},$$

where $\mathit{val}_{\mathrm{coin}_1^{\mathrm{old}}}$ and $\mathit{val}_{\mathrm{coin}_2^{\mathrm{old}}}$ denote the values corresponding to $\mathrm{coin}_0^{\mathrm{old}}$ and $\mathrm{coin}_2^{\mathrm{old}}$, respectively.

- $\top/\bot \leftarrow \mathrm{VerifyTransaction}(\mathsf{params}, \mathcal{L}_{\mathrm{ZCash}}, \mathrm{tx})$, that essentially verifies whether a transaction tx is correctly formed and does not conflict with the current version of the ledger.

Security Analysis: The security of a DAP requires that three properties are respected: *ledger indistinguishability*, *transaction nonmalleability*, and *balance*.

Each of these properties is formalized as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In each game, the transaction activity of honest parties is determined by the adversary. To enable this, the behavior of honest parties is realized via an oracle $\mathcal{O}$ that maintains a ledger $\mathcal{L}$ and provides a DAP interface (i.e., accepting queries) for executing any of the algorithm below on behalf of honest parties.

$$\{\mathrm{CreateAddress}, \mathrm{Mint}, \mathrm{Pour}, \mathrm{VerifyTransaction}, \mathrm{Receive}\}$$

To control the behavior of honest parties, the adversary constructs a query mapped to the transaction that it wants an honest party to perform and passes this query to $\mathcal{C}$ that (after sanity checks) proxies the query to $\mathcal{O}$. For each query that requests an honest party to perform an action, $\mathcal{A}$ specifies the identities of previous transactions and the input values and learns the resulting transaction, but not any of the secrets or trapdoors involved in producing that transaction. The oracle $\mathcal{O}$ also

provides a special query, Insert, that allows $\mathcal{A}$ to directly add arbitrary transactions to the ledger $\mathcal{L}$.

We now proceed to describe the security provisions of ZeroCash with respect to each of these properties.

- **Ledger Indistinguishability.** This is a property defined for the first time within the ZeroCash paper [9]. This property aims to represent the feature that a computationally bounded adversary who is given access to the ledger should not be able to derive more information about the content of the ledger than what is publicly available for that ledger (i.e., the number of transactions in the ledger, the addresses participating in Mint transactions, and so on).

  In a more abstract way, ledger indistinguishability represents in the realm of privacy-preserving Bitcoin (payment) transactions the equivalent to *message indistinguishability* in the realm of encrypted messages.

  Recall that the standard message indistinguishability game take places between a challenger who sets up the encryption scheme and generates key-pairs and an attacker who tries to break the security of the encryption scheme leveraging these keys. At the challenge phase, the adversary specifies two messages of the same length, and asks the challenger to encrypt exactly one of them using the encryption scheme whose security is to be proven. The challenger is assumed to pick one of the two messages at random, encrypts it, and sends the corresponding ciphertext to the adversary. The latter is asked to guess which message the ciphertext corresponds to. The challenged encryption scheme is said to offer message indistinguishability as long as the attacker cannot provide the correct answer with better probability than $\frac{1}{2}$.

  Ledger indistinguishability evolves in a similar way. It involves a challenger and a (computational) adversary who is given access to certain address keys and the power to control (to some extent) the transaction activity of honest users. Namely, the challenger samples a random bit b and initializes two DAP oracles implementing the DAP system interface $\mathcal{O}_0$ and $\mathcal{O}_1$, each maintaining a separate ledger $\mathcal{L}_0$ and $\mathcal{L}_1$, respectively. The adversary is then allowed to submit queries to the challenger in pairs, one destined for $\mathcal{O}_0$ and $\mathcal{L}_0$, and the other destined for $\mathcal{O}_1$ and $\mathcal{L}_1$. The challenger processes these queries; namely, it forwards these queries to the associated oracle if and only if these queries have matching type and are identical in terms of information available to the adversary (or the addresses it has corrupted). The challenger always provides the responses to these queries of the adversary to the two ledgers in a randomized order. That is, the challenger presents the adversary first with $\mathcal{L}_{\text{left}} := \mathcal{L}_b$ and then with $\mathcal{L}_{\text{right}} := \mathcal{L}_{1-b}$.

The adversary is then requested to guess b and distinguish the order in which the two ledgers are presented. Ledger indistinguishability requires that the adversary is not able to guess b with better probability than $\frac{1}{2}$.

- **Transaction nonmalleability.** This property requires that a computationally bounded adversary is not able to modify the contents of transactions of other users at any point after these transactions were issued.

  Similar to ledger indistinguishability, transaction nonmalleability can also be expressed by means of a security game. In particular, we assume that the adversary constructs a ledger by communicating with an oracle and, as a result, the adversary sees the set of transactions $\mathcal{T}$ that were added to the ledger. At the challenge phase, the adversary is requested to output a transaction $\mathrm{tx}^* \notin \mathcal{T}$. The adversary wins the game, if there is another transaction $\mathrm{tx} \in \mathcal{T}$, such that tx^* and tx have the same serial number, i.e., both transactions attempt to spend the same coin, and tx^* is accepted by $\mathrm{VerifyTransaction}$ in the ledger that preceded tx.

  Transaction nonmalleability requires that the probability that the adversary succeeds in this game is negligible with respect to the security parameter.

- **Balance.** This property requires that a computationally bounded adversary should not be able to own more coins than the coins that were converted through Mint operations or received from others via Pour operations.

  The balance property is formalized in a similar way as transaction nonmalleability and assumes a bounded adversary who is in possession of a set of addresses A and is given oracle access to a ledger. At the challenge phase, the adversary is asked to present a set of valid coins owned by addresses in A such that the total amount of these coins exceeds those acquired from Mint and Pour transactions (that appear in the ledger). Clearly, this property is guaranteed if and only if the adversary succeeds in this game with negligible probability with respect to the security parameter.

ZeroCash: Implementing DAP Using ZK-Snarks. ZeroCash is an instantiation of a decentralized payment system using zk-SNARKS, and Bitcoin as Basecoin. In this paragraph, we take a closer look at how ZeroCash works.

As opposed to Extended ZeroCoin, coins in ZeroCash are associated with addresses. That is, a ZeroCoin is associated with an address public key $\mathrm{apk_c}$, and the knowledge of the respective secret key $\mathrm{ask_c}$ is needed to spend the coin.

To mint a coin c, the user is required to sample a random number ρ_{c} that will be used for the generation of the serial number ser_{c} of the minted coin as follows:

$$ser_{\mathrm{c}} = \mathrm{PRF}^{ser}_{\mathrm{ask}_{\mathrm{c}}}(\rho_{\mathrm{c}}),$$

where $\mathrm{PRF}^{ser}_{\mathrm{ask}_{\mathrm{c}}}$ is a collision resistant pseudorandom function using $\mathrm{ask}_{\mathrm{c}}$ as seed and ρ_{c} as input. To complete Mint, the coin address public key $\mathrm{apk}_{\mathrm{c}}$, the sampled number ρ_{c}, and coin value val_{c} are incorporated into a commitment value $\mathrm{cmt}_{\mathrm{c}}$.

First, a commitment is generated to bind values $\mathrm{apk}_{\mathrm{c}}$ together with ρ_{c} into $\tilde{\mathrm{cmt}}_{\mathrm{c}}$:

$$\tilde{\mathrm{cmt}}_{\mathrm{c}} = \mathsf{cmt}_{\tilde{\mathrm{r_c}}}(\mathrm{apk}_{\mathrm{c}}, \rho_{\mathrm{c}}),$$

for randomly selected $\tilde{\mathrm{r_c}}$, and then bind $\tilde{\mathrm{cmt}}_{\mathrm{c}}$ with val_{c} into $\mathrm{cmt}_{\mathrm{c}}$ as follows:

$$\mathrm{cmt}_{\mathrm{c}} = \mathsf{cmt}_{\mathrm{r_c}}(\tilde{\mathrm{cmt}}_{\mathrm{c}}, val_{\mathrm{c}}),$$

where $\mathrm{r_c}$ is selected uniformly at random. Note that because of the two-layer commitment process, it is possible that given $\mathrm{r_c}$, val_{c}, and $\tilde{\mathrm{cmt}}_{\mathrm{c}}$, one can prove that the minted coin c has value val_{c} without revealing the coin's serial number or $\mathrm{apk}_{\mathrm{c}}$.

Now, we have a coin c generated. The question that comes next is how to spend this coin—an operation otherwise known as Pour. To this end, assume that one wishes to spend $\mathrm{c}_{\mathrm{old}}$ into two fresh coins $\mathrm{c}^0_{\mathrm{new}}$ and $\mathrm{c}^1_{\mathrm{new}}$ of value $val_{\mathrm{c}^0_{\mathrm{new}}}$ and $val_{\mathrm{c}^1_{\mathrm{new}}}$, respectively. Recall that:

$$\mathrm{c}_{\mathrm{old}} = \{\rho_{\mathrm{c}_{\mathrm{old}}}, \mathrm{apk}_{\mathrm{c}_{\mathrm{old},}}, val_{\mathrm{c}_{\mathrm{old}}}\}.$$

Initially, the spender follows the same process as before to compute the output coins, that is:

$$\mathrm{c}^{\mathrm{i}}_{\mathrm{new}} = \{\rho_{\mathrm{c}^{\mathrm{i}}_{\mathrm{new}}}, \mathrm{apk}_{\mathrm{c}_{\mathrm{new,i}}}, val_{\mathrm{c}^{\mathrm{i}}_{\mathrm{new}}}\}.i = 0, 1$$

The spender also computes the resulting commitment values $\mathrm{cmt}_{\mathrm{c}^{\mathrm{i}}_{\mathrm{new}}}\}_{i=0,1}$. ZeroCash leverages zk-SNARKs to show that $\mathrm{c}_{\mathrm{old}}$ has not been already spent and that value of the output coins (i.e., $val_{\mathrm{coin}^{\mathrm{new}}_0} + val_{\mathrm{coin}^{\mathrm{new}}_1}$) does not exceed the value of the spent coin (i.e., $val_{\mathrm{c}_{\mathrm{old}}}$). In the sequel, we denote by $\mathrm{root}_{\mathrm{cmt}}$ the root of the Merkle tree built from coin commitments that have been added to the ledger $\mathrm{root}_{\mathrm{cmt}}$. The spender of $\mathrm{c}_{\mathrm{old}}$ generates a zk-SNARK proof π for the following statement. *Given* $\mathrm{root}_{\mathrm{cmt}}$, *coin serial number* $ser_{\mathrm{c}_{\mathrm{old}}}$, *and commitments* $\{\mathrm{cmt}_{\mathrm{c}^{\mathrm{i}}_{\mathrm{new}}}\}_{i=0,1}$, *there are coins* $\mathrm{c}_{\mathrm{old}}$, $\mathrm{c}^0_{\mathrm{new}}$, $\mathrm{c}^1_{\mathrm{new}}$, *and coin address* $\mathrm{ask}_{\mathrm{c}_{\mathrm{old},}}$ *such that the following hold:*

- Coins $\mathrm{c}_{\mathrm{old}}$, $\mathrm{c}^0_{\mathrm{new}}$, and $\mathrm{c}^1_{\mathrm{new}}$ are well formed. That is, their commitment values were correctly computed using $\{\rho_{\mathrm{coin}}, \mathrm{apk}_{\mathrm{c}_{\mathrm{coin}}}, \mathit{val}_{\mathrm{coin}}\}_{\mathrm{coin}=\mathrm{c}_{\mathrm{old}},\mathrm{c}^0_{\mathrm{new}},\mathrm{c}^1_{\mathrm{new}}}$.
- Serial number $\mathit{ser}_{\mathrm{c}_{\mathrm{old}}}$ was correctly computed using $\mathrm{ask}_{\mathrm{c}_{\mathrm{old}},}$ and $\rho_{\mathrm{c}_{\mathrm{old}}}$.
- $\mathrm{cmt}_{\mathrm{c}_{\mathrm{old}}}$ appears in the Merkle tree of coin commitments with root $\mathrm{root}_{\mathrm{cmt}}$.
- $\mathit{val}_{\mathrm{c}_{\mathrm{old}}} = \mathit{val}_{\mathrm{c}^0_{\mathrm{new}}} + \mathit{val}_{\mathrm{c}^1_{\mathrm{new}}}$.

Proof π (along with $\mathit{ser}_{\mathrm{c}_{\mathrm{old}}}$, $\mathrm{cmt}_{\mathrm{c}^0_{\mathrm{new}}}$, $and\mathrm{cmt}_{\mathrm{c}^{0,1}_{\mathrm{new}}}$) constitutes essentially the Pour transaction but still does not offer nonmalleability. To remedy this, the spender samples an additional signature key-pair $\langle \mathrm{pk}^{\mathrm{c}_{\mathrm{old}}}_{\mathrm{Sig}}, \mathrm{sk}^{\mathrm{c}_{\mathrm{old}}}_{\mathrm{Sig}} \rangle$ that binds the spending of $\mathrm{c}_{\mathrm{old}}$ as follows:

1. The spender computes $\mathrm{h}_{\mathrm{Sig}} = \mathrm{hash}(\mathrm{pk}^{\mathrm{c}_{\mathrm{old}}}_{\mathrm{Sig}})$, where hash is a collision-free hash function.
2. The spender computes $\mathrm{h} = \mathrm{PRF}_{\mathrm{ask}_{\mathrm{c}_{\mathrm{old}},}}(\mathrm{h}_{\mathrm{Sig}})$.
3. The spender adds $\mathrm{h}, \mathrm{h}_{\mathrm{Sig}}$ to the Pour transaction content.
4. The spender extends the statement associated with the Pour transaction in order to include a proof that $\mathrm{h}and\mathrm{h}_{\mathrm{Sig}}$ are correctly computed.
5. The spender uses $\mathrm{sk}^{\mathrm{c}_{\mathrm{old}}}_{\mathrm{Sig}}$ to sign the extended Pour transaction.

Observe that the ledger indistinguishability and balance properties are implicitly provided by the zero-knowledge, and proof of knowledge (soundness) provisions of zk-SNARKS.

5.4 SUMMARY

In this chapter, we detailed a number of prominent attacks on the privacy provisions of Bitcoin. More specifically, we outlined a number of network-based attacks that leverage information exchanged by Bitcoin peers throughout their participation in the network. We also discussed a number of protocol-based attacks on the system and quantified the privacy offered by Bitcoin in light of these attacks. Finally, we discussed a number of research attempts, such as ZeroCash and Extended Zerocoin, to enable privacy-preserving payments in Bitcoin.

Our analysis shows that existing implementations of Bitcoin leak considerably information about the profiles of users. This is especially evident when an

adversary can leverage the use of behavior-based clustering algorithms and combine their use with a number of heuristics that capture multi-input transactions and shadow addresses. Note that the manual creation of new addresses can only partly conceal the profiles of users who participate in a small amount of transactions. Such a countermeasure, however, does not increase the privacy of users who are active in the network and participate in a large number of transactions.

To enhance user privacy in Bitcoin, mixing transactions emerges as an effective technique to hide the linkability between inputs and outputs of Bitcoin transactions. Existing solutions rely on a mixing server to harden the tracing of coin expenditure in the network; here, the mixing server still needs to be trusted to ensure anonymity, since it learns the mapping of coins to addresses. Even when mixers can be instantiated without the need of a centralized mixing server, such protocols cannot fully prevent clustering analysis since they do not hide the amounts and times of payments.

On the other hand, although privacy-preserving cryptographic enhancements of Bitcoin can prevent the leakage of transaction amounts and times, such protocols incur considerable computational overhead and require significant modifications to the Bitcoin protocol.

We note that the lack of privacy offered by the current Bitcoin system can however be seen as an enabler for accountability measures in the system. Recall that incorporating accountability measures in Bitcoin is essential to deter misbehavior, especially given the lack of workable mechanisms to ban/punish Byzantine nodes.

References

[1] Fergal Reid and Martin Harrigan. *An Analysis of Anonymity in the Bitcoin System*, pages 197–223. Springer New York, New York, NY, 2013.

[2] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*, 2013.

[3] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, pages 34–51, 2013.

[4] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, pages 6–24, 2013.

[5] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A Fistful of Bitcoins: Characterizing Payments Among Men

with No Names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, 2013. ACM.

[6] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *Financial Cryptography and Data Security*, 2014.

[7] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *ACM Conference on Computer and Communications Security*, 2014.

[8] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 397–411, Washington, DC, USA, 2013. IEEE Computer Society.

[9] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Practical decentralized anonymous e-cash from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014.

[10] Elli Androulaki and Ghassan O. Karame. Hiding transaction amounts and balances in bitcoin. In *Trust and Trustworthy Computing*, 2014.

[11] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the Bitcoin transaction graph. *Future Internet*, 5(2):237–250, 2013.

[12] Bitcoin—Wikipedia, 2013. available from `https://en.bitcoin.it/wiki/Introduction`.

[13] Andreas Pfitzmann and Marit Hansen. Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management: A Consolidated Proposal for Terminology. pages 111–144, 2008.

[14] Bitcointalk Forum, available from `https://bitcointalk.org/`.

[15] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is Bitcoin a Decentralized Currency? *IEEE Security and Privacy Magazine, 2014*, May/June issue 2014, 2014.

[16] Malte Möser, Rainer Böhme, and Dominic Breuker. Towards risk scoring of bitcoin transactions. In *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014*, pages 16–32, 2014.

[17] TOR project. available from `https://www.torproject.org/`.

[18] Bitcoin Laundry. Bitcoin Laundry Mixing Service, 2013. available from `https://bitlaunder.com`.

[19] Bitcoin Fog. Bitcoin Fog Mixing Service, 2009. available from `www.bitcoinfog.com`.

[20] Blockchain.info. Blockchain Mixing Service, 2013. available from `www.blockchain.info`.

[21] CoinJoin: Bitcoin privacy for the real world, 2013. available from `https://bitcointalk.org/index.php?topic=279249.0`.

[22] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992.

[23] C. P. Schnorr. Efficient signature generation for smart cards. pages 239–252, 1991.

[24] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[25] Jan Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, 1998. ETH Series in Information Security and Cryptography.

[26] Stefan Brands. Rapid Demonstration of Linear Relations Connected by Boolean Operators. In *EUROCRYPT*, 1997.

[27] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, 1986.

[28] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, 2006.

[29] Man Ho Au, Willy Susilo, and Yi Mu. Proof-of-Knowledge of Representation of Committed Value and Its Applications. In *ACISP*, 2010.

[30] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.

[31] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct noninteractive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, 2013.

Chapter 6

Security and Privacy of Lightweight Clients

In this chapter, we briefly overview and analyze the security of simple payment verification in Bitcoin. We then discus the operation of current lightweight clients and analyze their security and privacy provisions.

6.1 SIMPLE PAYMENT VERIFICATION

We start by describing simple payment verification (SPV) in Bitcoin.

6.1.1 Overview

As described in Chapter 4, Bitcoin requires peers in the system to verify all broadcasted transactions and blocks. Clearly, this comes at the expense of storage and computational overhead. Currently, a typical Bitcoin installation requires more than 70 GB of disk space and considerable time (and computational resources) to download and locally index blocks/transactions that are contained in the blockchain. Moreover, the continuous growth of Bitcoin transactional volume incurs significant computational overhead on the Bitcoin clients when verifying the correctness of broadcasted blocks and transactions in the network. This problem becomes even more evident when users wish to perform/verify Bitcoin payments from resource-constrained devices such as mobile devices and tablets.

To remedy that, lightweight client implementations have been proposed in [1]. These clients only perform a so-called simplified payment verification (SPV). SPV

clients do not store the entire blockchain, nor do they validate all transactions in the system. Notably, SPV clients only perform a limited amount of verifications, such as the verification of block difficulty and the presence of a transaction in the Merkle tree, and offload the verification of all transactions and blocks to the full Bitcoin nodes.

Note that this clearly comes at costs of decentralization, since a critical security component of the network is outsourced to few nodes in the system. Recall that SPV clients currently connect by default to four regular nodes; if all these nodes are malicious, then they could effectively control the view of the SPV client on the network and could prevent the client from sending and receiving transactions.

6.1.2 Specification of SPV Mode

We now describe the detailed operations undergone by an SPV client.

In the SPV mode, the sending of transactions is performed similarly to the regular Bitcoin protocol (see Chapter 4). Namely, SPV clients are equipped with public/private key pairs that enable them to assert ownership of coins and issue transactions in the system.

Unlike full Bitcoin nodes, SPV clients do not receive all the transactions that are broadcast within the Bitcoin P2P network, but instead receive a subset of transactions filtered for them by the full nodes to which they are connected. Currently, SPV clients connect to a default of four different randomly chosen nodes.

In order to calculate their own balance, SPV clients request full blocks from a given block height on; here, the full Bitcoin nodes can also provide filtered blocks to the SPV client that only contain relevant transactions from each block. The nodes verify the signature of the received transactions, verify the proof-of-work of the blocks, and verify that these transactions were indeed included in the Merkle tree committed in the received blocks. Note that the client can rely on the block depth in order to determine the transactions' validity. For instance, if a transaction has been confirmed in an old block that appears in the blockchain, then it is highly unlikely that the transaction is incorrect.

Recall that each block contains the root of the Merkle tree of all transactions that were confirmed in the block.

To verify that a given transaction was included in a block, an SPV client requests a proof of membership from one of the (full) Bitcoin nodes that he or she connects to. The latter outputs the membership proof, π_M. The SPV client then uses π_M and the Merkle root committed in the block in order to verify the inclusion of a given transaction x.

Note that lightweight clients are not involved in the mining process. Readers should not confuse the operations of lightweight client with the notion of SPV mining. SPV mining refers to the act of mining blocks without validating the previous block (and the transactions included therein) in the chain. Note that since such miners do not know which transactions have been included in the last block (since they skip the verification of the transactions), they need to mine without including any transaction (except for the coinbase transaction) in order to avoid the risk of including transactions that conflict with the previous block.

As discussed in [2], this strategy allows the adversary to gain an advantage in the mining process (e.g., to be the first in finding the correct PoW). SPV mining recently caused a fork in the blockchain [3] due to the fact that some mining pools continued mining blocks that originally referenced an invalid block header.

6.1.3 Security Provisions of SPV mode

Clearly, the security offered by SPV mode is considerably weaker than that of the standard Bitcoin protocol. Nevertheless, SPV mode still offers reasonable security guarantees, provided that *at least one of the client's connections correctly follows the protocol and has up-to-date knowledge on the longest blockchain*.

Namely, if all connections of the SPV client are dishonest, then these connections could control the view of the SPV client on the network and could prevent the client from sending and receiving transactions. For example, if all the connections of the SPV client are populated by malicious nodes, then SPV client might not learn the height of the longest blockchain. In this case, malicious connections could then convince the SPV client of a chain that is not necessarily the one adopted by the network (i.e., of a fork chain that is smaller than the current blockchain adopted by the network). In this respect, SPV clients have no way to verify whether the blocks and transactions that they receive from their connections are part of the main blockchain. Clearly, it suffices that the SPV client learns the height of the longest blockchain from one honest neighbor in order to detect this misbehavior. Alternatively, the SPV client can measure the generation times of the received block headers; the client can suspect the occurrence of such an attack if he or she does not receive on average a block every 10 minutes.

At all times, we point out that malicious nodes cannot convince the SPV client to accept an ill-formed block or transaction, since the SPV client always verifies the proof-of-work for all received block headers and verifies the membership of transactions of interest in the respective blocks.

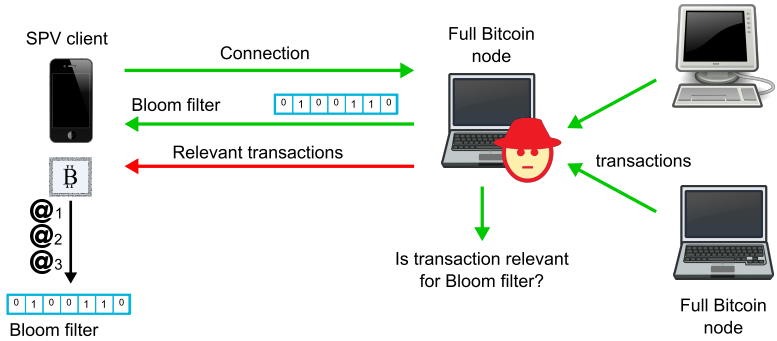

Figure 6.1 Sketch of the operation undergone by an SPV client. SPV clients connect to a regular Bitcoin node to which it also outsources its various Bloom filters. The regular node only forwards to the SPV clients the transactions relevant to their Bloom filters.

6.2 PRIVACY PROVISIONS OF LIGHTWEIGHT CLIENTS

In this section, we discuss the privacy provisions of existing lightweight Bitcoin clients as reported in [4]. We start by overviewing the operation of Bloom filters that are used in the SPV mode to filter transactions that are not relevant for SPV clients.

6.2.1 Bloom Filters

As mentioned earlier, SPV clients do not receive all the transactions that are broadcast within the Bitcoin P2P network, but instead receive a subset of transactions filtered for them by the full nodes to which they are connected. To reduce bandwidth consumption, SPV clients make use of Bloom filters [5, 6]. These filters basically consist of space-efficient probabilistic data structures that are used to test membership of an element. An SPV client constructs a Bloom filter by embedding all the Bitcoin addresses and public keys that appear in its wallets. The SPV client then outsources the constructed Bloom filter to a full Bitcoin node, as shown in Figure 6.1. Whenever the full node receives a transaction, it first checks to see if its input and/or output match the SPV client's Bloom filter. If so, the full node forwards the received transaction to the SPV client.

A Bloom filter B of an SPV client is typically specified by the maximum number of elements that it can fit, denoted by M, and a target false-positive rate P_t.

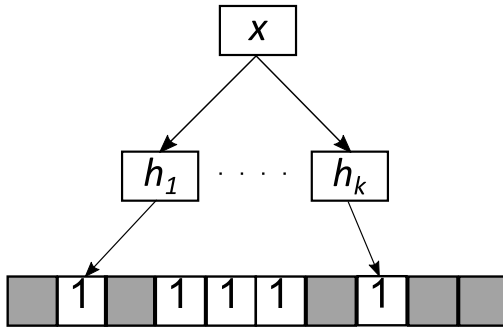

Figure 6.2 Sketch of the basic operation of Bloom filters.

As shown in Figure 6.2, a Bloom filter B basically consists of an array $B[.]$ of n bits accessed by k independent hash functions $H_1(.), \ldots, H_k(.)$, each of which maps an input string $x \in \{0,1\}^*$ to one of the n bits of the array; all bits of $B[.]$ are initialized to zero. To insert an element s, one has to set the bits at position $h_1(x), \ldots, h_k(x)$ in $B[.]$ to 1. Similarly, to test whether an element is a member of $B[.]$, one has to check the bits at positions $h_1(x), \ldots, h_k(x)$; if any of those bits are not 1, then the element is not a member of $B[.]$.

Bloom filters can generate a number of false positives, *but cannot result in false negatives.* In this book, we compute the false positive rate of a filter $B(M, P_t)$ which has m elements, $P_f(m)$, as follows [7]:

$$P_f(m) = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k \tag{6.1}$$

Here, note that $P_f(M) \approx P_t$. That is, the target false positive rate of a Bloom filter is only reached when the number of elements contained in the filter matches M [4].

6.2.2 Privacy Provisions

In what follows, we analyze the privacy provisions of SPV clients and show that the reliance on Bloom filters within existing SPV clients leaks considerable information about the addresses of Bitcoin users. We also show that this information leakage is further exacerbated when users restart their SPV clients and/or when the adversary

has access to more than one Bloom filter pertaining to the same SPV client. Motivated by these findings, we also describe an efficient countermeasure introduced in [4] in order to enhance the privacy of users that rely on SPV clients; this countermeasure can be directly integrated within existing SPV client implementations.

In our analysis, we assume that the adversary can compromise one or more full Bitcoin nodes and eavesdrop on communication links in order to acquire one or more Bloom filters pertaining to an SPV client. Here, the goal of the adversary is to identify the Bitcoin addresses that are inserted within a Bloom filter created by a particular SPV client. The addresses inserted in the Bloom filter typically correspond to addresses that the SPV client is interested in receiving information about (e.g., these addresses typically belong to the wallet of the SPV client). For example, the adversary might be connected to the node that generated the Bloom filter or might try to assign an identity to nodes according to their addresses. Note that since the Bitcoin network provides currently less than 10,000 reachable full Bitcoin nodes, it is likely that regular nodes receive one or more filter pertaining to each SPV client over a sufficiently long period of time.

6.2.3 Leakage Due to the Network Layer

Clearly, the adversary can try to link different Bloom filters to a single wallet by identifying the IP addresses used to outsource the Bloom filters. If, for example, the same IP address outsources two different Bloom filters to a regular node, then that node could directly infer that those filters belong to the same entity.

This leakage is even more exacerbated since an adversary who is connected to an SPV client can see the transactions issued by the client and could potentially use this in order to learn the clients' addresses.

6.2.3.1 Countermeasure

Information leakage that originates from the network layer can be countered, for example, by SPV clients using anonymizing networks such as Tor [8] whenever they issue Bitcoin transactions or they outsource their Bloom filters.

6.2.4 Leakage Due to the Insertion of Both Public Keys and Addresses in the Bloom filter

In current implementations of SPV clients, both the addresses and their public keys are inserted in the outsourced Bloom filter. As such, if the adversary knows both the

address and its public key, then she he or can trivially test whether an address is a true positive of the filter by checking whether both the address and its public key are inserted within the filter. If not, then it is highly likely that the address is a false positive of the filter.

We believe that the inclusion of both the address and its public key in the Bloom filter is a severe flaw in the current SPV client implementations.

6.2.4.1 Countermeasure

More than 99% of all Bitcoin transactions consist of payments to Bitcoin addresses (or the public key hash); moreover, only 4,587 out of 33 million studied addresses in the system received transactions destined for *both* their public keys and their public key hashes.[1] This means that for the vast majority of Bitcoin clients, there is no need to include both the public keys and their hashes (i.e., the Bitcoin addresses) in the Bloom filters; inserting one or the other would suffice (in more than 99% of the cases). Note that only inserting the addresses in the Bloom filter would suffice since regular nodes can easily hash the public keys and check whether they match the Bloom filter. However, this clearly incurs additional computational overhead on regular Bitcoin nodes.

6.2.5 Leakage under a Single Bloom Filter

In the sequel, we assume that SPV clients use anonymizing networks when connecting to regular Bitcoin nodes. As mentioned earlier, this alleviates potential leakage in the network layer. We additionally assume that SPV clients do not insert the public key and the corresponding Bitcoin address into the same filter in order to prevent trivial leakages of their embedded addresses (see Section 6.2.4). We show that even with these measures, Bloom filters still leak considerable information about the embedded client addresses.

Namely, in existing SPV clients, a node initializes its Bloom filter B_i with a random nonce r and specifies its target false positive rate P_t that can be achieved when a number of elements M have been inserted in the filter. By default, M is set to $m + 100 = 2N + 100$, where N is the number of Bitcoin addresses inserted in B_i. Here, the additional number 100 was originally added to m by the Bitcoin developers in order to avoid the recomputation of a new filter in the case where a user inserts up to 50 additional Bitcoin addresses (recall that a Bitcoin address is

1 These numbers were obtained by parsing the Bitcoin blockchain until block # 296000.

inserted into the Bloom filter by adding both the corresponding public key and the public key hash to the filter; therefore $m = 2N$).

The default target false positive rate of the Bloom filter P_t is set to 0.05% at the time of writing. The size of the filter n and the number of hashes k in Bloom filters are computed as follows:

$$n = -\frac{M \ln(P_t)}{(\ln(2))^2} \tag{6.2}$$

$$k = \ln(2)\frac{n}{M} \tag{6.3}$$

Note that if the SPV client restarts (e.g., mobile phone reboots, mobile application is restarted), then the Bloom filter will be recomputed (the SPV client stores the Bloom filter in volatile memory). When the user acquires 50 or more additional addresses such that $m > M$, then the SPV client will resize the Bloom filter by recomputing $M = 2N + 100$, and will send the updated Bloom filter to the full Bitcoin nodes that it is connected to.

Note that given n and k, the number of elements contained in a Bloom filter can be estimated by the adversary as follows [9]:

$$m \approx -n\frac{ln(1 - \frac{X}{n})}{k} \tag{6.4}$$

Here, X corresponds to the number of bits of the Bloom filter set to one. Given n and P_t, M can also be computed by the adversary from (6.2).

Note that, in April 2014, the Bitcoin blockchain comprised nearly $|\mathbb{B}|$ =33 million addresses. This means that an adversary can simply try all possible addresses in the Bitcoin system in order to compute the positives of the Bloom filter B_i, denoted in the sequel by $\mathcal{B}_i$.

Following from [4], we quantify the privacy offered by a Bloom filter using the probability, $P_{h_{(j)}}$, that the adversary correctly guesses any j true positives of a Bloom filter among all positives that match the filter and which are not included in the knowledge of the adversary.[2] More specifically, we measure $P_{h_{(j)}}$ achieved by a Bloom filter B_i, as follows: $P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} = \frac{N}{N+S} \cdot \frac{N-1}{N+S-1} \ldots$. Here, N refers to the number of Bitcoin addresses inserted into B_i and S denotes the cardinality of the set $\{\mathcal{F}_i - \mathbb{K}\}$; S therefore corresponds to all false positives that match B_i, but for which the adversary does not have any knowledge about.

2 Clearly, the higher is $P_{h_{(.)}}$, the smaller is the privacy of the SPV node.

It is straightforward to see that:

$$P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} \approx \prod_{k=0}^{j-1} \frac{N-k}{N+|\mathbb{B}-N|P_f(2N)-k} \tag{6.5}$$

Here, $N \ll |\mathbb{B}|$, and $m = 2N$ is the number of elements contained in the Bloom filter seen by the adversary.

Note that if the adversary is able to identify an SPV client (e.g., by some side channel information), then simply identifying any address pertaining to that client would be a considerable violation of its privacy. Otherwise, if the adversary can link a number of addresses to the same anonymous client, then the information offered by the clustering of these addresses offers the adversary considerable information about the profile of the client, such as its purchasing habits, and so on.

It is worthy to note that $P_{h_{(.)}}$ may not always capture the probability of guessing addresses that belong to the user of the SPV client, for example, in the case where the SPV client may embed addresses that do belong to the user in its Bloom filter.

6.2.5.1 Poorly-populated Bloom Filters

Following from (6.5), $P_{h_{(1)}}$ (the probability of correctly guessing one address as a true positive) is large when $2N/M \leq 0.4$, as long as $N < 100$. Given a modest number of addresses in the Bloom filter (i.e, $N < 100$), $P_f(m = 2N)$ is small when $\frac{m}{M}$ is small. As $\frac{m}{M}$ increases, $P_f(m = 2N)$ increases (and $P_{h_{(1)}}$ decreases). For example, when $N = 10$, $P_{h_{(10)}} = 0.99$ which corresponds to the probability of correctly guessing all the true positives when the SPV client has 10 addresses.

This means that the information leakage in SPV clients is considerable for new Bitcoin users or for Bitcoin users that restart their SPV clients. For instance, at initialization time, the Bloom filter of SPV clients is typically instantiated using $M = 102$. Moreover, if the user is new in the Bitcoin system and only has 1 Bitcoin address, $P_{h_{(1)}} \approx 1$—which results in complete lack of privacy. Recall that this observation also holds when the SPV client restarts and $N < 100$.

Gervais et al. analytically compute $P_{h_{(j)}}$ when the SPV client has 5, 10, 15 and 20 addresses [4]. Their results show that guessing all addresses given one filter that embeds less than 15 addresses can be achieved with almost 0.80 probability. This probability decreases as the number of addresses embedded within the filter increases beyond 15. This analysis has been validated by means of experimentation in the real Bitcoin network by Gervais et al. [4].

On the other hand, when the Bloom filter comprises a considerable number of elements (i.e., when $2N/M > 0.4$), then $P_f(m = 2N)$ is close to P_t.

6.2.6 Leakage under Multiple Bloom Filters

We now describe the information leakage due to multiple Bloom filters as analyzed in [4]. For that purpose, we assume that the adversary can acquire $b > 1$ Bloom filters pertaining to different users. For example, the adversary might be connected to SPV clients for a long period of time and receive their updated Bloom filter. Alternatively, the adversary can acquire additional Bloom filters by compromising/colluding with other full Bitcoin nodes. Similar to Section 6.2.5, we assume that SPV clients do not embed public keys and their corresponding addresses in the same filter; we also assume that these clients connect to regular nodes using an anonymizing network in order to avoid obvious leakage due to network layer information.

6.2.6.1 Two Bloom Filters

We start by analyzing the case where the adversary acquires two different Bloom filters B_1 and B_2. In the sequel, we focus on computing $P_{h_{(.)}}$ corresponding to filter B_1, which we assume to be the smallest of the two filters (in size). In analyzing the information leakage due to the acquisition of two Bloom filters, we distinguish two cases.

6.2.6.2 B_1 and B_2 Belong to Different Users

Recall that each Bloom filter is initialized with a random seed chosen uniformly at random from $\{0, 1\}^{64}$. Therefore, if B_1 and B_2 pertain to different users, then it is highly likely that they are initialized with different random seeds. This means that the false positives generated by each filter are highly likely to correspond to different addresses. Moreover, since different users will have different Bitcoin addresses, B_1 and B_2 will contain different elements. Therefore, $\mathcal{B}_1 \cap \mathcal{B}_2$ is likely to comprise only few addresses, if any. Notably, when B_1 and B_2 pertain to different users, then

$|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx (|\mathcal{B}_1| - N_1)|\mathcal{B}_2|\frac{1}{|\mathbb{B}| - N_1} \tag{6.6}$$

$$\approx \frac{P_f(m_1)P_f(m_2)|\mathbb{B}|^2}{|\mathbb{B}| - N_1} \tag{6.7}$$

$$\approx P_f(m_1)P_f(m_2)|\mathbb{B}|, \tag{6.8}$$

where N_1 corresponds to the number of elements inserted in B_1. $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ is the expected number of elements that match $\mathcal{B}_2$ and $\mathcal{B}_1$. The number of elements in $\mathbb{B}$ that match $\mathcal{B}_2$ is given by $P_f(m_2)|\mathbb{B}|$. Then, $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ can be computed by assuming a binomial distribution with success probability $P_f(m_2)$ and with $P_f(m_2)|\mathbb{B}|$ number of trials.

Note that the adversary can compute m_1 (using (6.4)); if $m_1 > |\mathcal{B}_1 \cap \mathcal{B}_2|$, then this offers a clear distinguisher for the adversary that the two acquired Bloom filters B_1 and B_2 pertain to different user wallets.

6.2.6.3 B_1 and B_2 Belong to the Same User

On the other hand, in the case where B_1 and B_2 correspond to the same SPV client, three subcases emerge:

B_1 and B_2 use the same size/seed: This is the case when users, for example, create additional Bitcoin addresses and need to update their outsourced Bloom filters to include those addresses. In this case, $\mathcal{B}_1$ and $\mathcal{B}_2$ are likely to comprise similar Bitcoin addresses. This includes both the actual elements of the filters (i.e., the Bitcoin addresses of the user and the false positives generated by the Bloom filter). In this case, $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + P_f(2N_1)|\mathbb{B}| \tag{6.9}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(2N_1)|\mathbb{B}| - k} \tag{6.10}$$

In this case, $P_{h_{(j)}}$ is not affected by the acquisition of the second filter B_2.

B_1 and B_2 use different seeds: In existing SPV clients, the random nonce r used to instantiate the Bloom filter is stored in volatile memory. Therefore, each time

the SPV client is restarted (e.g., smartphone reboots), a new filter will be created with a new seed chosen uniformly at random. If the adversary acquires two Bloom filters of the same user that are initialized with different seeds, then these filters are likely to exhibit different false positives. B_1 and B_2 will however comprise a number of identical elements (which map to the Bitcoin addresses of the user). More specifically,

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + (|\mathcal{B}_1| - N_1)\frac{|\mathcal{B}_2|}{|\mathbb{B}| - N_1} \tag{6.11}$$

$$\approx N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| \tag{6.12}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| - k} \tag{6.13}$$

Note that the obtained $P_{h_{(j)}}$ is considerably large in this case when compared to the case where the adversary has access to only one filter.

B_1 and B_2 use the same seed, but have different sizes: This is the case when users, for example, create additional Bitcoin addresses beyond the capacity of their current Bloom filters. SPV clients therefore need to resize their Bloom filters. Note that filter resizing typically shuffles the bits of the Bloom filters; the resulting distribution of bits in the new resized filter is not necessarily pseudorandom (since the same seed is used) and depends on the sizes of the filters. As such, only the lower bound on $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be estimated using (6.12) (which estimates the worst case where filter resizing causes a pseudorandom permutation of the bits of the filters).

Recall that since there are only a few tens of millions of addresses in Bitcoin, the adversary can brute-force search the entire list of Bitcoin addresses in order to acquire $\mathcal{B}_1$ and $\mathcal{B}_2$ and compute $\mathcal{B}_1 \cap \mathcal{B}_2$. Given any two Bloom filters B_1 and B_2, the adversary can easily guess whether these two Bloom filters contain Bitcoin addresses from the same wallet. Indeed, if $|\mathcal{B}_1 \cap \mathcal{B}_2|$ is small, then it is highly likely that B_1 and B_2 map to different elements (if m_1 and m_2 are not small), and therefore pertain to different users. On the other hand, when $|\mathcal{B}_1 \cap \mathcal{B}_2| \gg 0$, it is highly likely that all the Bitcoin addresses in the set $\mathcal{B}_1 \cap \mathcal{B}_2$ belong to the same SPV client.

Table 6.1

$P_{h_{(.)}}$ with respect to the Number b of Bloom Filters Belonging to the Same User

b	$P_{h_{(1)}}$ ($P_t = 0.05\%$)	$P_{h_{(1)}}$ ($P_t = 0.1\%$)	$P_{h_{(N/2)}}$ ($P_t = 0.05\%$)	$P_{h_{(N/2)}}$ ($P_t = 0.1\%$)	$P_{h_{(N)}}$ ($P_t = 0.05\%$)	$P_{h_{(N)}}$ ($P_t = 0.1\%$)
1	0.1713	0.0926	0	0	0	0
2	0.9978	0.9911	0.0091	0	0	0
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1

6.2.6.4 Multiple Bloom Filters

In the previous paragraphs, we discussed the case where the adversary is equipped with only two Bloom filters. Note that our analysis equally applies to the case where the adversary possesses any number $b > 2$ of Bloom filters pertaining to the same entity.

As mentioned earlier, by computing the intersection between each pair of filters, the adversary can find common elements to different filters; this also enables the adversary to guess with high confidence whether different filters have been generated by the same client. Given b filters that belong to the same SPV client, the adversary can compute the number of elements inserted within each filter using (6.4). In the sequel, we assume that filters $B_1, \ldots, B_b$ are sorted by increasing number of elements (i.e., B_b contains the largest number of elements), and that filters are constructed using different seeds. Let $\mathcal{K}_j = \mathcal{B}_j \cap \cdots \cap \mathcal{B}_{(b-1)}, \forall j \in [1, b-1]$.

Note that $|\mathcal{K}_1| \leq |\mathcal{K}_2| \cdots \leq |\mathcal{K}_{(b-1)}|$. Here, the larger the number of Bloom filters at the disposal of the adversary, the smaller is the error of the adversary in correctly classifying the genuine addresses of the SPV client, and the larger is $P_{h_{(.)}}$. That is, the larger is b, the smaller are the number of common false positives that are exhibited by the different filters, and the higher is the confidence of the adversary in identifying the false positives of B_j.

Moreover, as j increases, $\mathcal{K}_j$ will contain more false positives, and $P_{h_{(j)}}$ will decrease.

In what follows, we analytically validate this analysis and investigate the impact of having $b > 2$ Bloom filters pertaining to the same SPV client. For that purpose, we use 5 Bloom filters B_1,B_2,...,B_5 generated using different seeds with $N = \{3070, 3120, 3170, 3220, 3270\}$. We then compute $\mathcal{K}_j = \mathcal{B}_1 \cap \cdots \cap$

$\mathcal{B}_{(j+1)}, \forall j \in [1, b-1]$, and the corresponding $P_{h_{(.)}}$ as follows:

$$E[|\mathcal{K}_1|] = \min(|\mathcal{B}_1|, |\mathcal{B}_2|, \dots) \approx N_1 + |\mathbb{B}| \prod_{\forall j} P_f(m_j) \tag{6.14}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_i - k}{N_i - k + |\mathbb{B}| \prod_{\forall j} P_f(m_j)} \tag{6.15}$$

The results (depicted in Table 6.1) validate the aforementioned analysis[3] and show that the larger the number b of acquired Bloom filters of the same SPV client, the larger is $P_{h_{(.)}}$ and the smaller is the privacy of the user's addresses. For instance, if the adversary is able to collect $b > 2$ different Bloom filters pertaining to the same wallet, then the adversary will be able to recover 100% of the true positives of the smallest Bloom filter.

6.2.7 Summary

We now summarize our findings with respect to the privacy provisions of existing SPV clients.

- The number of elements inserted within a Bloom filter significantly affects the resulting false positive rate of the filter. This is especially true when the filter's size is modest (e.g., < 500). Indeed, the number of elements inserted in the filter should match at all times the filter's size in order to achieve the target false positive rate (i.e., $P_f(m) = P_t$).
- The acquisition of multiple Bloom filters considerably reduces the privacy of SPV clients. Namely, assume that the adversary is able to acquire two or more Bloom filters B_1 and B_2, then the following holds:
  - Given any two Bloom filters B_1 and B_2, if $|B_1 \cap B_2| \ll \frac{\min(m_1, m_2)}{2}$ (here, m_1, m_2 denote the number of elements inserted in B_1 and B_2, respectively), then an adversary can be certain that B_1 and B_2 do not belong to the same wallet.
  - If the two Bloom filters acquired by the adversary belong to the same SPV client, the adversary can identify whether the SPV client has restarted while generating his or her Bloom filters.
  - $P_{h_{(.)}}$ corresponding to $b = 2$ filters is considerably larger when compared to the case where the adversary has access to one Bloom filter. This means that

3 Here, we assume that each filter is generated using a different seed.

an adversary that can acquire more than one Bloom filter pertaining to an SPV client can learn considerable information about the addresses of the node—irrespective of the size of the Bloom filter and P_t. In this case, our results show that $P_{h_{(N)}}$ approaches 1, which signals full leakage of the addresses of the SPV client. $P_{h_{(.)}}$ increases to 1 as the number b of Bloom filters of the same SPV client captured by the adversary increases.

- SPV clients should keep state about their outsourced Bloom filters (i.e., on persistent storage) to avoid the need to recompute a filter that contains the same elements using different parameters.
- Inserting both the public key and the public key hash (the address) in the Bloom filter provides a sufficient distinguisher for the adversary in guessing whether an address is a true positive or not. Note that for the most common transaction type *Pubkey Hash* (P2PKH), inserting the hash of the public key is sufficient. However, there might be other transaction types where it is beneficial to also store the public key in the Bloom filter. In this case, the client can insert either a Bitcoin address or its corresponding public key (but not both) in the same Bloom filter.

6.2.8 Countermeasure of Gervais et al.

In what follows, we describe a countermeasure devised by Gervais et al. in [4] to enhance the privacy of SPV clients. This countermeasure emerges naturally from the current limitations of existing implementations.

Here, each SPV client generates N Bitcoin addresses at the start, and embeds them in a Bloom filter that can fit $M = m = N$. Clients only insert *the address* within each filter. Note that the Bloom filter is constructed with a realistic target false positive rate P_t, which combined with N and M, results in a target privacy level (see (6.1)). Moreover, we note that since $M = m$, then we ensure that the Bloom filter's false positive rate matches P_t. Clearly, since the user might not directly use all N addresses, some of his or her Bitcoin addresses will not be revealed and will remain in the wallet. In this case, shadow addresses (i.e., addresses that are typically generated by Bitcoin clients to receive change) are then automatically chosen from those unused addresses among the total N addresses. This eliminates the need for clients to constantly update their outsourced Bloom filters whenever a new shadow address has been created by their client—thereby minimizing bandwidth and maximizing privacy.

Whenever users run out of their N addresses and need to get additional addresses, they repeat the aforementioned process. That is, users create an additional set of N addresses and embed them in a new Bloom filter—constructed with a new initial seed— with $M = m = N$, and using the previously chosen P_t. In this case, the advantage of an adversary that captures one or more Bloom filters pertaining to the same SPV client is negligible, since these filters do not have any element in common.

Additionally, this solution requires the SPV clients to keep state, for example, about each Bloom filter, to avoid the need of recomputation of the same filter if the client restarts at any point in time.

This proposed solution can be directly integrated within existing SPV clients and only incurs in small modifications to existing client implementations. Moreover, this solution does not incur additional overhead on the SPV clients—apart from the pregeneration of N Bitcoin addresses (which is only done at setup time), and the storage space required for each generated Bloom filter. More detail on this countermeasure can be found in [4].

References

[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[2] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. *IACR Cryptology ePrint Archive*, 2015:578, 2015.

[3] What is spv mining, and how did it (inadvertently) cause the fork after bip66 was activated? available from `https://bitcoin.stackexchange.com/questions/38437/what-is-spv-mining-and-how-did-it-inadvertently-cause-the-fork-after-bip66-wa`.

[4] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. On the Privacy Provisions of Bloom filters in Lightweight Bitcoin Clients. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, New Orleans, LA, USA, December 8-12, 2014*, pages 326–335, 2014.

[5] Mike Hearn. Connection bloom filtering, 2012. available from `https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki`.

[6] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[7] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.

[8] TOR project. available from `https://www.torproject.org/`.

[9] S Joshua Swamidass and Pierre Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of Chemical Information and Modeling*, 47(3):952–964, 2007.

Chapter 7

Bitcoin's Ecosystem

by Angelo De Caro and Ghassan Karame

In this chapter, we explore the salient points of the Bitcoin ecosystem. We also discuss the impact that this cryptocurrency is fueling with respect to business innovations.

At the time of writing, Bitcoin is heavily used to perform cross-border payments, machine-to-machine transactions, as well as stock settlements, just to name a few.

Data from `blockchain.info` shows that the overall daily transaction volume has increased over time moving from the approximately 100,000 confirmed Bitcoin transactions in June 2015 to the 230,000 of May 2016 [1]. Furthermore, data from Bitcoin merchant processor *BitPay*, launched in 2011, shows a general trend toward increased merchant transactions [2]. November 2015 recorded an all-time high of around 100,000 Bitcoin transactions. *TigerDirect*, a publicly traded online electronics retailer, has seen exciting results: among all clients that used Bitcoin in TigerDirect, 46% of them were new customers. Furthermore, orders placed with Bitcoin were 30% larger.

Certainly, one of the main reason of this exciting expansion is related to the growth of the number of Bitcoin-accepting merchants. While only 2% of merchants currently accept Bitcoin, 25% of merchants expect to offer it within the next two years, according to a recent survey by Goldman Sachs and the Electronic Transactions Association [3]. Estimations also foresee that the number of merchants accepting Bitcoin will raise from the current 160,000 to 1.8 million in 2017.

Bitcoin as an asset class is also maturing. For the majority of 2015, Bitcoin's exchange price has remained relatively nonvolatile and constant—fluctuating between $200 to $300. From January 1, 2013 to January 1, 2014, the price went from $13.41 to $808.05, going as high as $1,147.25 on December 4. Just one month earlier, on November 4, 2013, the price was $225.20. On the other hand, the market cap of Bitcoin is down from an all-time high of nearly $14 billion to around $7 billion at time of writing.

Regulations seem also to be changing. Organizations such as *Coin Center* and the *Chamber of Digital Commerce* work to help regulators in drafting rules that will ensure Bitcoin can continue to grow worldwide.

The remainder of this section is organized as follows. In the first three sections, we explore the financial aspects of the Bitcoin ecosystem. Namely, we describe how payments and exchanges can be executed and how to store the Bitcoin coins (BTCs) that one possesses. We also discuss the security of Bitcoin wallets. Then, we will delve into mining and how different users can join their forces to mine BTCs. Finally, we discuss the impact of Bitcoin on the gambling business.

7.1 PAYMENT PROCESSORS

One of the biggest benefits of the decentralized nature of Bitcoin it that anyone can start accepting payments without the need to register an account with a third-party provider. Nevertheless, for many businesses, especially the small ones, the learning curve required by the new system is still sometimes too steep. For them, it is still more convenient to pay a small fee to a payment processor.

Namely, instant conversion of Bitcoin (BTC) to local fiat currencies (e.g., USD, EUR, Yuan) is one of the most popular services offered by payment processors. This service is crucial for all the businesses that accept Bitcoin payments, but still have to pay all or part of their own costs using fiat currencies. Instant conversion reduces the risk of losses derived by the fluctuation of the exchange rates between Bitcoin and the fiat currencies.

Instant conversion is not the sole service offered by payment processors. These usually provide an entire suite of tools and services to make the adoption of Bitcoin convenient and simple.

We can divide the payment mechanisms in two main categories. The first one is the so-called *person-to-person* payment mechanism that addresses small businesses and is the simplest way to accept BTCs, while the second one is the

point-of-sale (POS) solution that targets larger organizations. In Chapter 4, we discuss in detail the security of Bitcoin payments.

In the context of person-to-person payment mechanism, the simplest way to accept Bitcoin payments is by having the customer sending the required amount of BTCs directly to the digital wallet of the merchant. In order to streamline this process, CoinBox [4], a leading Bitcoin trading platform in Malaysia, offers an interesting solution. Here, the merchant, using an application on his or her smartphone, can convert the price of a good or service to a QR code that contains the amount to be paid and the address of the recipient. The customer scans the QR code with his or her Bitcoin wallet application and the payment is sent.

Despite their simplicity, person-to-person payment systems are unlikely to be used by large businesses that are interested in solutions to accept BTCs that integrate well and smoothly with their existing POS systems. Therefore, the market offers many POS solutions that a merchant can choose from in order to satisfy his or her specific requirements. *Coinify* [5] is a Danish firm that offers POS solutions that allow payments to be accepted in person anywhere from anyone. Merchants can get paid in 17 digital currencies or fiat currency or a mixture of the two.

CoinKite [6] is a start-up that offers a Bitcoin payment terminal looking exactly like the usual chip-and-PIN terminals that are commonly found in stores. This handset reads a Bitcoin-based debit card, also offered by CoinKite, and can also serve as a Bitcoin and Litecoin ATM, as well as offer the option to print QR codes for customers to scan with their smartphone applications.

BitPay [7] is an international payments processor for businesses and charities. It is integrated into the SoftTouch POS system for bricks-and-mortar retail stores. However, BitPay has an API that could be integrated easily within any other POS system. BitPay has various tariffs that merchants can subscribe to, enabling features such as using the service on a custom domain for online stores.

Revel Systems [8] offers different iPad-based POS solutions to satisfy various merchant categories from restaurants to retail outlets. They also support Bitcoin as a method of payment.

Paystand Bitcoin Merchants [9] aims to be a multipayment gateway that eliminates merchant transaction fees by supporting digital currency acceptance.

Finally, *XBTerminal* [10] provides a Bitcoin POS device that allows the merchant's customers to pay from any mobile Bitcoin wallet by NFC or QR code. Payment from off-line mobile devices is supported by Bluetooth. Payments take place through the company's platform and, if desired, BTCs can be converted instantly to fiat currency at the time of sale.

7.2 BITCOIN EXCHANGES

Essentially, Bitcoin exchanges allow the transfer of fiat currencies into BTCs or other digital currencies, and vice versa.

The basic workflow is quite simple: a user, equipped with an account in his or her preferred exchange service, first deposits money (in the currencies supported by the exchange service) to their account. Subsequently, the user can start trading with other users of the same exchange service or with the service itself, and withdraw money from their account.

Similar to traditional currency exchange services, this exchange is performed by placing *buy* or *sell* orders. The exchange service is in charge of matching the orders. A *buy* order is an offer to buy BTCs in exchange for another currency (fiat or digital). On the other hand, a *sell* order is an offer to sell BTCs. An exchange can be performed if the price of a buy order is higher than the price asked by a sell order. As long as the service behaves correctly, there is no risk of losing money.

While exchanging BTCs, users must take particular care given the intrinsic nature of the currency; once a BTC is spent, it is hard to reverse such a payment. Therefore, exchanging BTCs with payment methods like credit cards exposes to the risk of charge-back fraud [11].

A number of solutions are currently offered in order to streamline the exchange process. For example, Coinbase has an option to link a user's bank account to his or her Coinbase wallet [12]. Coinbase also offers automatic purchasing of BTCs at regular intervals. On the other hand, BitStamp acts as a mediator enabling a user to trade with other users.

Exchanges are not the only way to acquire BTCs. A popular service called *Local Bitcoins* [13] pairs up potential buyers and sellers. People from different countries can exchange their local currency to BTCs. Local Bitcoins offers an escrow service to protect the buyer of BTCs, meaning that Local Bitcoins holds money on behalf of transacting parties.

7.3 BITCOIN WALLETS

As mentioned earlier, the main goal of wallets is to securely store the private keys needed to spend the BTCs that one possesses. They come in various forms designed to satisfy specific requirements.

The cheapest way to store BTCs securely is by using *paper wallets*. Basically, a paper wallet service generates an address for the user and creates an image that

consists of two QR codes. The first one encodes the address the user can receive BTCs to and the second QR encodes the secret key to be used to spend the BTCs received at the address assigned to the user.

A number of *mobile wallets* also emerged in order to allow payments via smartphones. These wallets typically locally store the private keys needed to spend the BTCs and enable the payment directly from the phone. NFC technology can further streamline the payment process. By tapping the phone against a reader, the payment can be processed without the need for extra interaction.

Mobile wallets are made possible using the simple payment verification mechanism (SPV). Indeed, SPV allows a smartphone to verify that a transaction is included in the Bitcoin blockchain without downloading the entire blockchain (see Chapter 6).

On the other hand, classical *desktop wallets* can offer more advanced services such as support for transaction anonymization to prevent tracking. In this space, Electrum [14] is one of the most interesting offerings. It has support for *multisig* (see Chapter 3) to split the permission to spend your coins between several wallets and Bitcoin hardware wallets, and supports SPV mode for transaction verification.

Another category in the Bitcoin wallet space is that reserved to the *online wallets*. These are web-based wallets that store the private keys in the cloud and provide high availability and ubiquity. An online wallet can be accessed from virtually everywhere and from any device. At the same time, these wallets exhibit one fundamental drawback: the private keys are not under the control of the owner anymore. This puts the BTCs that one possesses at serious risk.

Hardware wallets are dedicated devices that digitally hold private keys and offer payments assistance. The market offers various devices that are sometimes also certified against different kinds of attacks (both physical and logical). In the case of hardware wallets, recoverability is probably one of the most crucial aspect in case of hardware failure. The combination of services offered by an online wallet and security provided by hardware wallets, is certainly very attractive and can reduce the risks of having one's BTCs stolen. These wallets are currently very limited in number and can be tamper-resistant. Examples include the Trezor hardware wallet and the Ledger USB wallet.

Although Bitcoin is the most prominent cryptocurrency at the time of writing, it is not the only one. This often leads to the situation where several different digital wallets need to be maintained to keep the coins, at least one for each of them. A viable solution to this issue is to use the so-called *multicurrency wallets* that unify the management of multiple coins under a single interface.

A Ripple wallet [15] can be used to hold any currency or asset (including fiat money as well as digital currencies) for which there is a *gateway*. Gateways are businesses that provide a way for money and other forms of value to move in and out of the Ripple [16]. The list of supported currencies is quite large [17].

HolyTransactions [18] is another interesting multicurrency web wallet that supports different coins and allows spending them from the same place or exchanging them instantly one for another.

7.3.1 Securing Bitcoin Wallets

Recall that Bitcoin transactions basically consist of transferring the outputs of unspent previous transactions to a new public key (address). Therefore, to redeem a given coin/transaction, peers simply have to sign the transfer of this coin using a private key that matches the public key to whom the transaction was sent (see Figure 4.1). Clearly, the compromise or loss of a private key means that peers can no longer redeem any transaction sent to the corresponding public key.

By default, each user possesses a digital wallet that is part of the standard (desktop and/or mobile) Bitcoin installation. Wallets can be migrated from one machine to the other and contain/manage all the private keys corresponding to the user. Clearly, if the machine of the user is broken, these keys will be lost and the coins owned by the user will be consequently lost. Moreover, the literature features a number of anecdotes where private keys were stolen from the devices of users, wallets, and exchange markets [19]. For example, almost half a billion dollars have been claimed to be stolen from one of the biggest Bitcoin exchange markets, Mt. Gox [19]. In another incident affecting the MyBitcoin web wallet, more than 78,739 BTC [20] were stolen.

Protecting private keys from damage, loss, and compromise is therefore of outmost importance. In what follows, we discuss the security of existing Bitcoin wallets.

7.3.1.1 Security of Online Wallets

Different types of online wallets have emerged. Some store the private keys on the server side (some of which encrypt the private keys before storing them), while others store them locally in the browser of the user. Depending on where the private key is stored, online operators can gain unilateral powers over the BTCs of their users.

For example, in April 2013, a theft of 923 BTCs occurred in the mining pool OzCoin. A subset of the stolen BTCs were transferred to a web wallet hosted by StrongCoin. Although StrongCoin claims that it supports user privacy and does not have access to the user funds, StrongCoin intercepted the allegedly stolen BTCs and transferred them back to OzCoin [21]. This exemplifies the degree of control that an online wallet can have on the BTCs owned by its clients.

Note that if the private keys of clients are stored in the browser of the users or are stored encrypted (with a key stored at the user's premises), then private keys can still be lost if, for example, the computer breaks. Therefore, existing solutions that place minimum trust at the wallet operator might not necessarily increase the resilience against the loss of BTCs (e.g., due to hardware failure).

7.3.1.2 Security of Hardware Wallets

Hardware wallets safely store the private keys of individuals without the need to rely on third-party Bitcoin storage services. Note that in the case of loss of the hardware wallet, the private keys are irrecoverable, as are the associated BTCs of the clients.

7.3.1.3 Security of Paper Wallets

Since paper wallets are not stored digitally, they are not subject to cyber attacks or hardware failures. However, in the event of a loss of a paper wallet, the corresponding private keys will also be lost and cannot be recovered. We also point out that the paper wallet operator (i.e., the website that generates the QR codes) could learn the public/private key pair associations and could then potentially leak those keys or spend the BTCs received by the generated public keys. Some paper wallet operators, such as Bitcoinwalletpaper.com [22], mitigate this threat by performing client-side key generation (and by making their code open-source for the public to check the code's correctness).

7.3.1.4 Multisig Transactions

Multisignature addresses are addresses associated with more than one ECDSA private key. Generally speaking, these addresses could be m out of n addresses, where n private keys are created for a given public address such that any $m \leq n$ private keys can spend the coins stored within the public address.

The primary use of multisignatures is to considerably increase the difficulty of stealing coins. For example, the m private keys could be stored on different

machines/devices. Moreover, this scheme can resist the loss of up to $(n-m)$ private keys. Finally, multisignatures can also be used in scenarios where an address is shared by multiple people, and a majority vote is required to spend the BTCs stored within that address.

7.3.1.5 Trusted Computing

One possible alternative to secure the storage of private keys would be to borrow techniques from trusted computing [23]. For example, one can leverage hardware support, such as TPM chips, ARM Trustzone [24], and Intel SGX [25], to securely seal the private keys stored on users' personal devices. These private keys can then only be unsealed and recovered if the software state of the device at the time of recovery is exactly the same at the time of sealing—thus ensuring that no malware is present at the time of unsealing. Note that this does not protect against hardware failures; in this case, private keys might not be recoverable.

7.3.1.6 Multicloud Storage

Another alternative to secure private keys against loss or theft would be to rely on multicloud storage systems. Multicloud storage systems typically rely on a number of commodity cloud providers (e.g., Amazon, Google) with the goal of distributing trust across different administrative domains. This model is receiving increasing attention nowadays with leading cloud storage providers such as EMC, IBM, and Microsoft, offering products for multicloud systems [26].

Here, the private keys could be secret-shared and each share can be stored across multiple clouds. Recall that secret-sharing schemes allows a user to distribute a secret among a number of entities, such that only authorized subsets of shareholders can reconstruct the secret. In threshold secret-sharing schemes, the user can define a threshold t such that any t out of n shares can reconstruct the secret. Secret-sharing guarantees security against a nonauthorized subset of shareholders; the combination of any $0 \leq m < t$ shares does not leak any meaningful information about the secret [27–29].

By secret-sharing the private keys in a multicloud storage system, the security of the private keys is ensured unless t or more cloud operators collude. Moreover, such a solution resists against the failure of up to $(n-t)$ clouds. Note that applications for multicloud storage systems can be made available on all devices of the users, thereby allowing users to seamlessly synchronize their keys across

Table 7.1
Provisions of Alternatives to Secure Bitcoin Wallets

Technique	Resists Hardware Failures	Resists Cyber Attacks	Resists Loss
Standard wallets	No	No	No
Online wallets	Yes	No	Yes
Hardware wallets	No	Yes	No
Paper wallets	Yes	Yes	No
Multisig transactions	Yes	Partially	Partially
Trusted computing	No	Yes	No
Multicloud storage	Yes	Yes	Yes

their devices. Table 7.1 summarizes the provisions of the investigated alternatives to secure Bitcoin wallets.

7.4 MINING POOLS

The current difficulty level of mining BTCs is so prohibitively high that it reduces the incentives for miners to operate alone. Joining a mining pool is an attractive option to receive a portion of the Bitcoin block reward on a consistent basis. Namely, mining pools offer a way for miners to contribute their resources to generate a block and to split the reward between all the pool members following a certain reward payment scheme. Shares of the reward are assigned by the mining pool to its members who presents valid proof-of-work. In more detail, a mining pool sets a difficulty level between 1 and the currency's difficulty. Subsequently, a share is assigned to those miners that provide a block header that scores a difficulty level between the pools difficulty level and the currency's difficulty level. The main purpose of these block headers is to show that the miner is contributing with a certain amount of processing power.

When deciding which mining pool to join, it is important to understand the reward payment scheme used by the pool and the fees that the mining pool operator deducts. Typical fee values range from 1% to 10%. However, some pools do not deduct any fees, meaning that the full block reward is distributed among the mining pool participants.

The most basic reward payment scheme is the *Pay Per Share* (PPS) that offers an instant, guaranteed payout for each share that is solved by a miner. The pools use

their existing balance to pay the miners who can withdraw their payout immediately without the need to wait for a block to be solved or confirmed. As such, the PPS scheme requires a large reserve of money to avoid bankruptcy. A variation of the PPS is the *Shared Maximum Pay Per Share* (SMPPS) that never pays more than the Bitcoin mining pool has earned [30]. In the *Equalized Shared Maximum Pay Per Share* (ESMPPS) [31], payments are distributed equally among all miners in the pool. *Recent Shared Maximum Pay Per Share* (RSMPPS), on the other hand, gives priority to the most recent Bitcoin miners. The *Pay On Target* (POT) takes in consideration also the difficulty level of the proof-of-work submitted by the miners [32].

A different approach is that offered by the *Proportional* (PROP) scheme [33]. PROP proposes a proportional distribution of the reward among all miners when a block is found—based on the number of shares they have each found. A variation on the PROP scheme is the *Pay Per Last N Shares* (PPLN) [34], where rather than counting the number of shares in the round, it looks at the last N shares (where N is a parameter of the scheme), no matter when they were generated. The *SCORE* scheme [35] uses a system whereby a proportional reward is distributed and scored by the time the work was submitted. The miners are then rewarded based on the score of their shares rather than their amount.

Another approach is that offered by the *double geometric method* (DGM) scheme [36]. This scheme was designed to resist pool-hopping, a form of misbehavior where a miner only participates at the beginning of a round. In fact, the expected payout per share is always the same no matter when it was submitted.

In order to reduce the risk for the mining pool to be cheated by miners that switch pools during a round, the *Bitcoin pooled mining* (BPM) [37] scheme assigns lower scores to older shares (from the beginning of a block round), and higher scores to more recent shares.

Another important aspect to take into consideration when deciding to start mining is to choose which cryptocurrency to mine given that there are many alternative cryptocurrencies. An attractive option—whenever one is not sure which currency to mine—is a pool called *Multipool* that automatically switches one's mining hardware between the most profitable currency [38].

7.4.1 Impact of Mining Pools on De-centralization

Figure 7.1 depicts the distribution of computing power among mining pools in the Bitcoin network between October 10 and October 14, 2015. Our findings show that more than 75% computing power in Bitcoin was controlled in the investigated

period by five major centralized mining pools. If these pools were to collude in order to acquire more than 50% of computing power share in the network, they could effectively control the confirmation of all transactions occurring in the system. This includes preventing transactions from being executed, approving a specific set of transactions, and double-spending transactions [39]. Moreover, given the results of [40–42], F2Pool and AntPool, which control around 38% of the computing power in the network, can further considerably increase their advantage in the network by performing selfish mining; in the worst case, these two mining pools can determine together the fate of all transactions in the Bitcoin system.

Note that conscious miners can play an important goal in preventing a mining pool from controlling the computing power in the network. For example, miners of the Ghash.io pool actively abandoned the pool in 2014 in the fear of allowing the pool operator to acquire more than 51% of the computing power in the network [43]. This has forced Ghash.io's owner to issue a public statement reassuring the community that the mining pool will make sure to take "all necessary precautions" in order to avoid controlling 51% of the computing power in the network.

We however point out that it suffices that a given mining pool controls the computing power in the network for a short amount of time (i.e., before raising the suspicion of miners) in order to cause considerable damage in the network. Namely, a malicious pool could, for example, double-spend all transactions occurring within that period of time or selectively reverse payments—a misbehavior that might be economically justified when reversing/ill-spending large amounts.

While most existing mining pool protocols assume the existence of a logically centralized operator that orchestrates the block generation process, a number of fully decentralized mining pools, such as P2Pool, have been proposed. Such pools share the benefits of centralized pools since all the participating users get regular payouts that reflect their contribution toward generating a block. However, these pools do not require the existence of any centralized coordinator and operate in a completely decentralized fashion. Currently, P2Pool only holds a marginal share of the computing power in the network; Bitcoin users can only hope that such decentralized pools can be transformed into profitable businesses in the near future in order to attract most miners. However, the Bitcoins reward mechanism provides no particular incentive for users to use such decentralized alternatives.

Currently, members of mining pools need to frequently submit cryptographic proofs to the pool operator in order to demonstrate that they are indeed contributing their computing power to the benefit of the pool. This mechanism is used to enhance trust among different untrusted pool members. In [44], Miller et al. observed that

if the proof-of-work would effectively enable the miner to steal the reward without leaving any evidence, then miners do not have any incentive to join pools. Namely, any pool operator wishing to outsource the proof-of-work risks losing the mining reward. Miller et al. then proposed two constructs of such nonoutsoureable puzzles, which ensure that even if there are legal contracts between the pool operator and the miners, then miners can effectively steal the awards without leaving any evidence of misbehavior. Implementation results show that these puzzles add additional but tolerable overhead to the cost of Bitcoin blockchain validation.

Although few mining pools clearly control the computing power in the network (see Figure 7.1), we argue that there is still hope for reducing the impact of mining pools on the Bitcoin system. Given that Bitcoin relies on the notion of controlled supply that effectively limits the total number of generated BTCs (i.e., the amount of BTCs that are generated for each block is halved every 4 years), the ultimate dominance of mining pools is expected to decrease with time, since their profits would depend less and less on self-awarded BTCs and more on transaction fees. This, in turn, also increases the contribution of individual users to the Bitcoin economy. Indeed, mining pools operators would then have less incentives to accept client versions that are adopted by the minority of the clients (see Section 7.6). Users would then contribute more to the decision making process in Bitcoin by deciding to adopt a client version that suits their preferences. Recall that since the Bitcoin source code is open source, there are already a considerable number of different Bitcoin implementations [39].

7.5 BETTING PLATFORMS

The online gambling industry business is estimated around $30 billion [45] and is only expected to grow. Bitcoin offers to this industry key features such as pseudonymity, immediacy, and payment finality.

The market offers an array of different online Bitcoin gambling platforms, from sports betting to casino games with live dealers via poker that is currently in vogue. For many, BitcoinGG [46] is seen as the most prominent Bitcoin gambling platform; BitcoinGG constantly reviews the newest Bitcoin gambling site.

Fairness is surely an important aspect of the online games. Today, many gambling sites provide proof that they are operating legitimately by providing evidence for provably fair games [47]. More specifically, a gambling site that offers provably fair games allows public verifiability of the outcomes of the games based on the gambler's inputs and a secret information that is reveled at the end of a round.

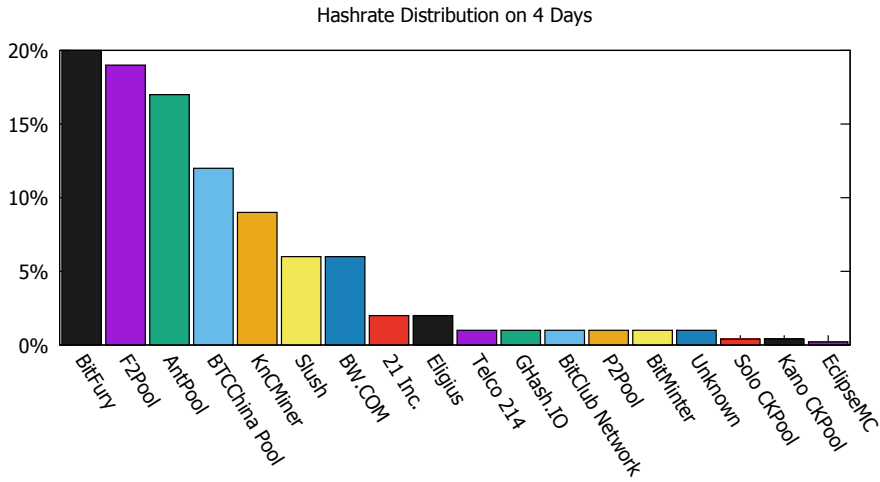

Figure 7.1 Distribution of computing power in Bitcoin between October 10, 2015, and October 14, 2015. More than 50% of the computing power in the network is controlled by F2Pool, AntPool, BTCChina Pool, and BitFury.

An example of such a gambling site is SatoshiDICE [48]. The game generates so-called *lucky numbers* by using the transaction ID and secret that is disclosed at the end of the day. Note that online gambling is not legal in a number of countries.

7.6 PROTOCOL MAINTENANCE AND MODIFICATIONS

The original Bitcoin client was developed by Satoshi Nakamoto in 2008. Nakamoto continued to support the maintenance and releases to the Bitcoin client until mid-2010 where he was replaced by a small group of Bitcoin developers.

The Bitcoin core developers have the authority to make all the necessary modifications to the Bitcoin protocol; according to the Bitcoin Github repository, all radical decisions require consensus among all the developers. For example, in the Bitcoin client version 0.8.2, the developers unilaterally introduced a fee policy change and decided to lower the default fee for low-priority transactions from 0.0005 BTC to 0.0001 BTC. Clearly, this empowers the Bitcoin developers to regulate and control the entire Bitcoin economy.

7.6.1 Bitcoin Improvement Proposals

In order to affect the Bitcoin development process, Bitcoin users are requested to file a Bitcoin Improvement Proposal (BIP) [39] that is assessed by the Bitcoin developers. The developers then unilaterally make a decision whether such a proposal will be supported by the future Bitcoin releases.

This limits the impact that users have, irrespective of their computing power, to affect the development of the official Bitcoin client. Recent events reveal that contributing within the Bitcoin community is not a trivial process [49, 50].

Recently, several of the original lead developers of Bitcoin decided to stop supporting the system due to a large debate on the future of emerging currency. At the core of this debate lies deep misunderstandings about the source code governance namely, with respect to expanding the block sizes in Bitcoin. More specifically, a subset of the core developers were in favor of increasing the block size beyond the default cap of 1 MB in order to better cope with the growth of the network, while the rest of developers opposed such a move in the fear of changing/worsening the current network dynamics. In theory, such a debate should be resolved by the computing power in the network (i.e., the miners). Note that most of the computing power was collectively held at the time among two Chinese mining pools; these considerably biased the decision of keeping the maximum block size at 1 MB. Some allege that this decision was politically motivated by the desire of the Chinese mining pools to prevent the growth of the system [50].

This large debate resulted in the exit of developers who were favoring the increase of the maximum block size; another immediate outcome of this debate was that Coinbase—one of the known Bitcoin start-ups—was banned from community forums for siding with those developers [50]. These events clearly show the lack of democracy in the governance of Bitcoin even among those developers who are behind the Bitcoin core software.

7.6.2 The Need for Transparent Decision Making

In some settings, it is inevitable that various client versions/implementations require constant maintenance and development by a group of leading developers. Here, problems arise in those situations where the developers have to take action in order to resolve possible conflicts that may have arisen. Indeed, this process needs to be completely transparent and should be tightly regulated in order not to abuse the trust of users and to minimize such unilateral interventions in the system. For example, in order to prevent the (ab-)use of alerts in Bitcoin, these alerts should be accompanied

with provable and undeniable justifications. Based on these proofs, users can then decide whether to accept such warnings. For instance, double-spending alerts can include the double-spending transactions [51]; this provides irrefutable proof that a given address is double-spending.

Finally, careful planning and testing of version releases is required so as to ensure backward compatibility with previous versions.

7.7 CONCLUDING REMARKS

In this chapter, we analyzed the current Bitcoin ecosystem. Although Bitcoin does not truly solve all of the challenges faced by previously proposed digital currencies, Bitcoin grew to witness a wider adoption and attention than any other digital currency proposed to date. At the time of writing, Bitcoin holds the largest market share among all existing digital currencies, with a market cap exceeding $3 billion [52]. As described in this chapter, there are numerous businesses, exchange platforms, and wallets that are currently built around the Bitcoin ecosystem.

Unlike previous electronic cash proposals, Bitcoin's proposal was rather straightforward, relying on basic cryptographic constructs, such as hash functions and digital signatures. The release of the proof of concept implementation of Bitcoin shortly after the dissemination of the white paper was extremely timely and important for the subsequent growth of Bitcoin. The working implementation confirmed that, unlike previous proposals, the system is clearly feasible/workable, and scales to a large number of nodes.

We believe that an additional reason that led to the sustainability and growth of the Bitcoin system was the ability of the developers to assimilate research results from the security community and integrate them swiftly within the development of released client implementations. In the remaining chapters, we discuss in detail the various security and privacy provisions of Bitcoin and its underlying blockchain—effectively capturing eight years of thorough research on these subjects in the Bitcoin community.

We however note that a large number of centralized services currently host Bitcoin and control a considerable share in the Bitcoin market. Even worse, Bitcoin developers retain privileged rights in conflict resolution and maintenance of the clients' software. These entities altogether can decide the fate of the entire Bitcoin system, thus bypassing the will, rights, and computing power of the multitude of users that populate the network.

Currently, almost every financial system is controlled by governments and banks; Bitcoin substitutes these powerful entities with other entities such as IT developers and owners of mining pools. While current systems are governed by means of transparent and thoroughly investigated legislations, vital decisions in Bitcoin are taken through the exchange of opinions among developers and mining pool owners on mailing lists. In this sense, Bitcoin finds itself now in unfamiliar territory: on one hand, the Bitcoin ecosystem is far from being decentralized; on the other hand, the increasing centralization of the system does not abide by any transparent regulations/legislations. This could, in turn, lead to severe consequences for the fate and reputation of the system.

References

[1] Blockchain.info. The number of daily confirmed bitcoin transactions. available from https://blockchain.info/charts/n-transactions.

[2] Blockchain.info. Understanding bitcoin's growth in 2015. available from https://blog.bitpay.com/understanding-bitcoins-growth-in-2015/.

[3] Goldman Sachs. Eta goldman sachs merchant acquirer and iso survey: Spring 2015. available from http://www.electran.org/wp-content/uploads/ETA-GS-Merchant-Acquirer-and-ISO-Survey-Spring2015.pdf.

[4] coinbox.biz. Real time, safe and easy with coinbox. available from https://coinbox.biz/.

[5] coinify.com. Conify: Blockchain payments. available from https://www.coinify.com/.

[6] coinkite.com. Coinkite: bitcoin wallet with multi-signature bank-grade security. available from https://coinkite.com/.

[7] bitpay.com. Bitpay: A bitcoin payment processor. available from https://bitpay.com/.

[8] revelsystems.com. Revel ipad point-of-sale software. available from http://revelsystems.com/.

[9] www.paystand.com. Paystand: 0% business payments. available from http://www.paystand.com/.

[10] xbterminal.io. Xbterminal: Bitcoin pos system. available from https://xbterminal.io/.

[11] Danny Bradbury. How credit card fraud sank one bitcoin exchange. available from http://www.coindesk.com/how-fraud-sunk-bitcoin-exchange/.

[12] coinbase.com. Coinbase: Bitcoin wallet. available from https://www.coinbase.com/.

[13] localbitcoins.com. Localbitcoins.com: Fastest and easiest way to buy and sell bitcoins. available from https://localbitcoins.com/.

[14] electrum.org. Electrum: Bitcoin wallet. available from https://electrum.org/.

[15] ripple.com. Ripple: Instant, certain, low-cost international payments. available from https://ripple.com/.

[16] ripple.com. What is a ripple gateway? available from https://ripple.com/knowledge_center/gateways/.

[17] ripple.com. Ripple popular gateways. available from https://ripple.com/knowledge_center/gateway-information/.

[18] holytransaction.com. Multi-currency wallet that actually works. available from https://holytransaction.com/.

[19] 2014. The 6 Biggest Bitcoin Heists in History, available from http://gizmodo.com/the-6-biggest-bitcoin-heists-in-history-1531881137.

[20] List of Major Bitcoin Heists, Thefts, Hacks, Scams, and Losses, available from https://bitcointalk.org/index.php?topic=83794.0.

[21] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is Bitcoin a Decentralized Currency? *IEEE Security and Privacy Magazine, 2014*, May/June issue 2014, 2014.

[22] Bitcoin paper wallet generator, 2014. available from https://bitcoinpaperwallet.com/.

[23] Alexandra Dmitrienko, David Noack, Ahmad-Reza Sadeghi, and Moti Yung. Poster: On offline payments with bitcoin. In *FC'2014: Financial Cryptography and Data Security Conference*, 2014.

[24] Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009.

[25] Software Guard Extensions Programming Reference. https://software.intel.com/sites/default/files/329298-001.pdf, 2013.

[26] Ghassan O. Karame, Claudio Soriente, Krzysztof Lichota, and Srdjan Capkun. Securing cloud data in the new attacker model. *IACR Cryptology ePrint Archive*, 2014:556, 2014.

[27] A. Shamir. How to Share a Secret? In *Communications of the ACM*, pages 612–613, 1979.

[28] H. Krawczyk. Secret Sharing Made Short. In *International Conference on Advances in Cryptology*, 1993.

[29] Amos Beimel. Secret-sharing schemes: A survey. In *Third International Workshop on Coding and Cryptology (IWCC)*, pages 11–46, 2011.

[30] eligius.st. Mining pool: Shared maximum pps. available from http://eligius.st/wiki/index.php/Shared_Maximum_PPS.

[31] bitcointalk.org. Mining pool: Equalized shared maximum pay per share. available from https://bitcointalk.org/index.php?topic=12181.msg378851#msg378851.

[32] bitcointalk.org. Mining pool: Pay on target. available from https://bitcointalk.org/index.php?topic=131376.0.

[33] en.wikipedia.org. Mining pool: Proportional. available from https://en.wikipedia.org/wiki/Mining_pool#Proportional.

[34] en.wikipedia.org. Mining pool: Pay-per-last-n-shares. available from https://en.wikipedia.org/wiki/Mining_pool#Pay-per-last-N-shares.

[35] en.bitcoin.it. Mining pool: Score. available from https://en.bitcoin.it/wiki/Comparison_of_mining_pools.

[36] bitcointalk.org. Mining pool: Double geometric method. available from https://bitcointalk.org/index.php?topic=39497.0.

[37] en.wikipedia.org. Mining pool: Bitcoin pooled mining. available from https://en.wikipedia.org/wiki/Mining_pool#Bitcoin_Pooled_mining.

[38] multipool.us. Multipool: A bitcoin, litecoin, and altcoin mining pool. available from https://www.multipool.us/.

[39] Bitcoin Wiki, available from https://en.bitcoin.it/wiki/.

[40] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 692–705, New York, 2015. ACM.

[41] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. *IACR Cryptology ePrint Archive*, 2015:578, 2015.

[42] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[43] Bitcoin miners ditch ghash.io pool over fears of 51% attack. available from http://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack/.

[44] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 680–691, New York, 2015. ACM.

[45] William Chambers. The breakdown: How big is real-money gaming? available from http://s4585.sites.pressdns.com/the-breakdown-how-big-is-real-money-gaming/.

[46] bitcoingg.com. The breakdown: How big is real-money gaming? available from http://www.bitcoingg.com/.

[47] provablyfair.org. Provably fair. available from http://provablyfair.org/.

[48] satoshidice.com. Satoshidice. available from https://www.satoshidice.com.

[49] [Bitcoin-development] Revisiting the BIPS process, a proposal, available from https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg02982.html.

[50] Mike Hearn. The resolution of the Bitcoin experiment, 2016. available from https://medium.com/@octskyward/the-resolution-of-the-bitcoin-experiment-dabb30201f7#.9g9iryds0.

[51] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917, New York, 2012. ACM.

[52] Bitcoin market cap, 2015. available from https://blockchain.info/charts/market-cap.

Chapter 8

Applications and Extensions of Bitcoin

In the last couple of years, most research was focused on the provisions of Bitcoin as a digital currency. Studies were analyzing the security and privacy of making payments in Bitcoin, the underlying economy of Bitcoin, and so on, but completely overlooked a key-enabling technology and a hidden potential within Bitcoin, the *blockchain*.

Indeed, Bitcoin's blockchain emerges as a truly genuine breakthrough. This blockchain implements a novel distributed consensus scheme that allows transactions, and any other data, to be securely stored and verified without the need for any centralized authority and while scaling to a large number of nodes. As such, the blockchain has fueled innovation in the last couple of years, and a number of innovative applications have already been devised by exploiting the secure and distributed provisions of the blockchain.

In this chapter, we overview a number of interesting extensions of Bitcoin. We also briefly discuss a number of applications that leverage Bitcoin's blockchain in order to create various services, such as decentralized storage and smart contracts, among others.

8.1 EXTENSIONS OF BITCOIN

At the time of writing, there are almost 500 alternate blockchains (also called altcoins) that offer alternative currency options besides BTCs. Most of these blockchains are clones of the Bitcoin blockchain, with minor configuration changes, namely:

Coin supply A number of altcoins vary the total coin supply in the system.

Hash function A number of altcoins rely on different underlying hash functions, such as SHA256 or SCRYPT.

Block generation times Some altcoins rely on different block generation times/difficu for the underlying proof-of-work.

In the remainder of this section, we briefly describe a number of prominent altcoin instantiations.

8.1.1 Litecoin

Litecoin is a well-known altcoin which, at the time of writing, holds the fourth largest market cap, after Bitcoin, Ripple, and Ethereum.

Litecoin's code is basically a clone of Bitcoin's with three basic differences. More specifically, the Litecoin network aims to generate a block every 2.5 minutes instead of the 10-minute interval featured by Bitcoin. This clearly allows for faster transaction confirmation times and in turn faster convergence on consensus in the network. This also suggests that Litecoin is more suited for fast payments where the time between the exchange of services and money is short (e.g., fast-food services). Another difference is that the Litecoin network is expected to generate 84 million Litecoins, which is four times larger than the number of currency units scheduled to be issued by the Bitcoin network.

A major difference between Litecoin and Bitcoin lies in the fact that Litecoin uses scrypt, a sequential memory-hard function, in order to reduce the advantage of computationally powerful miners. Namely, scrypt requires a large amount of memories in order to be efficiently computed; by doing so, scrypt trades off CPU-bound resources with memory-bound resources. The intuition is that memory access times vary much less than CPU speeds, and hence offer a fairer alternative for constructing PoW. There are currently several specialized ASIC mining hardware available for scrypt-based PoW systems.

8.1.2 Dogecoin

Dogecoin further reduces the confirmation times of transactions to almost 1 minute. A direct drawback is that Dogecoin features slightly higher probability of generating orphan blocks when compared to Litecoin and Bitcoin. Similar to Litecoin, Dogecoin also relies on scrypt, and increases the total coin supply that are forecast to be generated to around 100 billion Dogecoins.

The Dogecoin currency has gained considerable traction as an Internet tipping system in which users grant Dogecoin tips to others in exchange for providing

interesting or noteworthy content. At the time of writing, Dogecoin is among the top ten currencies with respect to the total market capitalization.

8.1.3 Namecoin

One of the first examples of the application of the blockchain is Namecoin [1]. Currently, the Internet Corporation for Assigned Names and Numbers (ICANN) governs nearly all top-level Web address domains such as ".com." Namecoin acts as a decentralized Domain Name Service that is resilient to censorship and serves as a new domain name system for registering Web addresses that end in ".bit." By doing so, Namecoin empowers its miners to distributively control domain names.

In Namecoin, each record consists of pair comprising a key and a value that can be up to 520 bytes in size. Each key points to a path, with the namespace preceding the name of the record. For example, key "d/example" signifies a record stored in the DNS namespace "d" with the name "example" and corresponds to the record for the example.bit website [1]. Note that the content of "d/example" should conform with the DNS namespace specification [2]. The current fee for inserting a record is 0.01 NMC (which denotes the currency in Namecoin) and records typically expire after 36000 blocks (approximately 200 days) unless they are updated. Similar to Bitcoin, Namecoin has a limited supply of 21 million Namecoins, which are released as a geometric series, by halving the generation amount every 4 years. Various statistics about the current usage dynamics of Namecoin can be found at `http://namecoin.webbtc.com/stats`.

Recent studies [3] have however shown that most users of Namecoin are not active and that the existing market for domains is almost nonexistent. For instance, [3] reveals that among among Namecoins roughly 120,000 registered domain names, only 28 have nontrivial content.

8.1.4 Digital Assets

Digital Assets is a technology provider for blockchain-based services mainly aimed at settling transactions of financial institutions. Digital Assets owns two main blockchain products [4]: Hyperledger [1] [5, 6] and Bits of Proof [2] [7].

Hyperledger is a blockchain that instantiates distributed ledgers using Practical Byzantine Fault Tolerant protocols. By doing so, Hyperledger is able to support real-time financial settlements on a scale of tens of thousands of transactions per

1 The *Hyperledger* brand name was donated to the Linux Foundation in December 2015.
2 The Bits of Proof code was donated to the Linux Foundation in December 2015.

second. This is achieved by leveraging a private blockchain environment where all participants are all known and permissioned,

On the other hand, Bits of Proof is an optimized variant of Bitcoin implemented in Java [8], which was later integrated within Hyperledger. Unlike BitcoinJ, Bits of Proof provides a Bitcoin server that is tailored for enterprise solutions with performance scalability in mind.

Namely, the merge between Hyperledger and Bits of Proof resulted in switching Hyperledger's codebase to Java/Scala and adopting the UTXO transaction model. UTXO transaction model allows Hyperledger to be interoperable with Bitcoin and other sidechains so that users of Hyperledger will benefit from the innovation from the Bitcoin community. A number of additional security features are also planned to be added to the Hyperledger blockchain: for instance, Hyperledger seeks to protect the confidentiality of transactions without hindering transaction validation process. This can be achieved, for example, by checking (in zero-knowledge) that the sum of outputs is smaller or equal to the sum of inputs. As mentioned earlier, the *Hyperledger* brand name was donated to the Linux Foundation in December 2015. In Chapter 9, we discuss in greater detail the current operation of the rebranded Hyperledger project.

8.2 APPLICATIONS OF BITCOIN'S BLOCKCHAIN

We now proceed to outlining a number of novel and innovative applications that leverage Bitcoin's blockchain.

8.2.1 Robust Decentralized Storage

As mentioned earlier, the blockchain allows different entities, such as banks, governments, and industrial players, to efficiently and securely reach consensus on the order of transactions, correctness of data, and so on. One of the envisioned exploitations of the blockchain lies in the construction of decentralized storage systems. The beauty behind this approach is that all data stored in the blockchain is expected to be replicated across a large number of nodes which ensures a high level of reliability.

In what follows, we start by discussing how to leverage the blockchain in order to store information. In the sequel, we assume that users have access to n storage nodes (e.g., public clouds), which have considerable storage capacity.

- Prior to storing object O on the nodes, the user generates a master secret K.

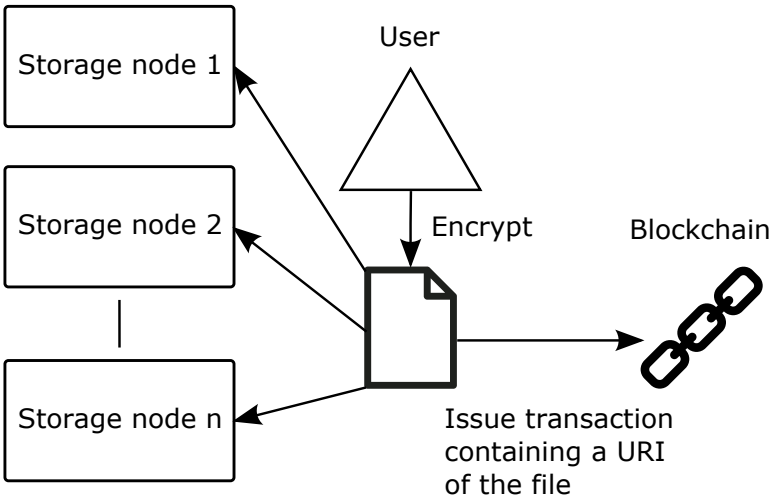

Figure 8.1 Using the blockchain as a metadata store.

- The user then computes $\mathrm{Enc}(K, O)$, which denotes the semantic secure encryption of object O under key K using function Enc.
- The user then stores the encrypted object $\mathrm{Enc}(K, O)$ redundantly on the n nodes and acquires n URIs $P_1, \ldots, P_n$ that point to the location of $\mathrm{Enc}(K, O)$ on each of the n nodes, respectively.
- The user then encrypts $P_1, \ldots, P_n$ using key K and stores the resulting encryption $\mathrm{Enc}(K, P_1), \ldots, \mathrm{Enc}(K, P_n)$ as well as $H(\mathrm{Enc}(K, O))$ on the *blockchain*. Here, $H(.)$ refers to a cryptographic hash function. That is, given $H(x)$, it is computationally infeasible to compute x (i.e., $H(.)$ is a one-way function), and it is likewise infeasible to compute $y \neq x$ such that $H(x) = H(y)$ (i.e., $H(.)$ is collision-resistant).
- Once the information stored by the user is confirmed in the blockchain, the user is certain that the metadata information (i.e., the object hash, the URIs) can never be modified by any entity.
- To retrieve the information, the user's client simply retrieves the encrypted URIs, decrypts them using key K, and uses them to fetch the data stored on one of the storage nodes.

- The user verifies that the hash of the downloaded object matches $H(\mathrm{Enc}(K, O))$ before decrypting and acquiring O.

Note that the aforementioned system (see Figure 8.1) is generic in the sense that the storage nodes could be emulated by the blockchain itself provided that the blockchain has enough storage capacity to store large objects. For instance, the PDF of Bitcoin's white paper was included in a Bitcoin transaction [9]. Currently, Bitcoin's blockchain contains a considerable amount of data [10], such as the biography of Nelson Mandela, Wikileaks documents, software, and so on.

However, in the particular case of Bitcoin, it is not practical to store large objects in the blockchain due to the various limitations imposed by the developers; recall that Bitcoin implements practices to ensure system scalability and to avoid large storage overhead. The main challenge therefore when utilizing Bitcoin's blockchain would be to store the minimum amount of data in the blockchain itself while still realizing robust storage.

This can be achieved by storing the actual objects onto dedicated storage nodes, while only storing the metadata within Bitcoin's blockchain. Recall that in Bitcoin, transactions can optionally have a field that can be used by developers to insert such metadata to transactions. Alternatively, metadata can be encoded within transactions in the form of (invalid) public keys. That is, instead of specifying a valid public key, developers can encode metadata (e.g., in HEX format) and include them in this field. Note that Bitcoin enables multioutput transactions and multisignature transactions. This would allow developers to encode considerable metadata in the fields reserved for multiple public keys. Clearly, by issuing such transactions, one has to pay the minimum Bitcoin fee to ensure that these transactions are included in the blockchain. Currently, the minimum fee is 0.0001 BTCs, which corresponds to almost 2 Euro cents given the current exchange rate.

This basic scheme is secure even when $n - 1$ storage nodes are arbitrarily malicious and as long as the there are enough honest blockchain nodes to ensure the security of the metadata stored therein. Note that a similar scheme can be used to construct erasure-coded storage. In this case, instead of storing exact replicas at the storage nodes, the user can invoke an information dispersal algorithm to encode object O into n chunks in such a way that any of the m chunks are enough to reconstruct O. The user stores the URIs of the chunks, as well as their individual cryptographic hashes in the blockchain. This variant scheme is secure even when $n-m$ storage nodes are arbitrarily malicious and as long as there are enough honest blockchain nodes to ensure the security of the metadata stored therein.

8.2.1.1 Authenticated Storage

Note that the aforementioned robust storage can be easily turned into an authenticated storage. Authenticated storage refers to a storage system where each entity can prove to another that it had stored a given object. Typical examples are court documents that need to be attested (e.g., that they are issued by a given entity) or modifications/updates to legal documents.

Namely, blockchain users are typically equipped with nonrepudiable public/private key pairs.[3] Since each transaction confirmed in the blockchain is authenticated, users can prove in the aforementioned storage protocol the ownership of object O.

Clearly, the blockchain can also be used to prove data ownership without revealing the actual data. For instance, one can publicly reveal a file digest (e.g., hash) for an object that has been committed in the blockchain and if conflict arises the person can prove that he or she has the data that matches the hash. This is especially useful for contracts, copyrighted material, patents, and so on. For example, one can prove that he or she developed a specific software revision at any given point in time by time-stamping the hash of the revision tree. BTProof [11] and Proof of Existence [12] already offer such services by leveraging Bitcoin's blockchain.

8.2.2 Permacoin

In [13], Miller et al. proposed Permacoin, a modification to Bitcoin that aims at repurposing its mining resources toward a more useful goal: decentralized storage. Permacoin's mining process requires investment in computational and storage resources. More specifically, Permacoin relies on a puzzle for Bitcoin based on Proofs of Retrievability (POR) [14, 15]. To mine coins, users need to prove access to a given copy of a file. By doing so, Permacoin motivates the construction of a highly decentralized storage system.

We start with a brief refresher on POR. *Proofs of Retrievability* (POR) are interactive protocols that cryptographically *prove* the retrievability of outsourced data. More precisely, POR consider a model comprising of a *single* user (or tenant) and a service provider that stores a file pertaining to the user. POR basically consist of a challenge-response protocol in which the service provider proves to the tenant that its file is still intact and retrievable. Note that POR only provide a guarantee that a fraction p of the file can be retrieved. For that reason, POR are typically performed

3 In the case of Bitcoin, each public key maps to a unique Bitcoin address.

on a file that has been erasure-coded in such a way that the recovery of any fraction p of the stored data ensures the recovery of the file [15]. A POR scheme consists of four procedures [14], setup, store, verify, and prove. The latter two algorithms define a protocol for proving file retrievability. We refer to this protocol as the POR *protocol* (in contrast to a POR scheme that comprises all four procedures).

setup. This randomized algorithm generates the involved keys and distributes them to the parties. In case public keys are involved in the process, these are distributed among all parties.

store. This randomized algorithm takes as input the keys of the user and a file $f \in \{0,1\}^*$. The file is processed and store outputs f^*, which will be stored on the server. store also generates a file tag τ, which contains additional metadata information about f.

prove. The prover algorithm takes as input the public key and the file tag τ and f that is output by store.

verify. The randomized verification algorithm takes the secret key, the public key, and the file tag τ outputted by store during protocol execution. Algorithm verify outputs at the end of the protocol run TRUE if the verification succeeds, meaning that the file is being stored on the server, and FALSE otherwise.

In Permacoin, mining is associated with the effort of performing a POR. This is achieved as follows:

- Each participant pseudorandomly samples the data segments to store. The seed of the pseudorandom function is based on the miner's public key. This also allows other participants in the network to verify that the participant is indeed storing the correct segments.
- Unlike existing POR schemes, the challenge/response protocol needs to be made noninteractive. This is achieved by publicising epoch-dependent unpredictable puzzle instances puz. The challenge is then simply $H(puz, s)$, where $H(.)$ is a cryptographic hash function, and s is a random seed selected by the participant. By doing so, each participant basically pseudorandomly challenges a number of segments that it is storing.
- The participant finally broadcasts a (POR) proof (based on Merkle trees) of possession of the challenged segments. Any participant in the network can simply check that the response is correct and that $H(puz, s)$ conforms with the current difficulty in the network. Only correct responses are broadcast in the network.

8.2.3 Decentralized Identity Management

The blockchain can be similarly used as a decentralized identity system. Namely, each entity can reserve and confirm its identity in the blockchain, which will inherently prevent any other entity from spoofing that identity. By confirming the identity records in the blockchain, such an approach ensures that (1) the identity cannot be changed/modified and (2) the identity is uniquely assigned to a single entity.

A successful instantiation of this application is OneName [16]. OneName is a protocol that enables the construction of decentralized identity system (DIS) using the Namecoin blockchain. Users are added to the OneName directory by means of a key-value store (KVS) interface, where the key is the username or ID and the value encodes the corresponding profile data (in JSON format).

8.2.4 Time-Dependent Source of Randomness

Bitcoin's blockchain (and variant altcoins' blockchain) can also be used to instantiate a time-dependent randomness generator $\mathsf{GetRandomness} : T \rightarrow \{0,1\}^{\ell_{\mathrm{seed}}}$ where T denotes a set of discrete points in time. In a nutshell, GetRandomness produces values that are unpredictable but publicly reconstructible.

More formally, let cur denote the current time. We define GetRandomness as follows. On input $t \in T$, GetRandomness outputs a uniformly random string in $\{0,1\}^{\ell_{\mathrm{seed}}}$ if $t \leq \mathsf{cur}$, otherwise GetRandomness outputs $\perp$. We say that GetRandomness is secure if the output of $\mathsf{GetRandomness}(t)$ cannot be predicted with probability significantly better than $2^{-\ell_{\mathrm{seed}}}$ as long as $t < \mathsf{cur}$.

Similar to [17,18], we instantiate GetRandomness by leveraging functionality from Bitcoin, since the latter offers a convenient means (e.g., by means of API) to acquire time-dependent randomness.

Recent studies show that a public randomness beacon—outputting 64 bits of min-entropy every 10 minutes—can be built atop Bitcoin [19].

Given this, GetRandomness then unfolds as follows. On input time t, GetRandomness outputs the hash of the latest block that has appeared since time t in the Bitcoin blockchain. Clearly, if $t > \mathsf{cur}$ corresponds to a time in the future, then GetRandomness will output $\perp$, since the hash of a Bitcoin block that would appear in the future cannot be predicted. On the other hand, it is straightforward to compute $\mathsf{GetRandomness}(t)$ for a value $t \leq \mathsf{cur}$ (i.e., t is in the past) by fetching the hash of previous Bitcoin blocks. In this way, GetRandomness enables an untrusted party to sample randomness without being able to predict the outcome ahead of

time. Note that the security of GetRandomness depends on the underlying security of the blockchain. More specifically, if an entity is able to predict the outcome of GetRandomness, then he or she is able to predict a future block hash in the blockchain.

8.2.5 Smart Contracts

Developers can leverage multisignature transactions in Bitcoin in order to construct smart contracts. Smart contracts refer to binding contracts between two or more parties and are enforced in a decentralized manner by the blockchain without the need for a centralized enforcer.

Recall that multisignature transactions (see Chapter 4) require $m > 1$ correct signatures to be considered valid transactions. Although the primary use of multisignature transactions mainly targeted resistance to coin theft, these transactions also support the construction of smart contracts in Bitcoin. In what follows, we discuss various types of achievable contracts in Bitcoin.

8.2.5.1 Making a Deposit

Recall that Bitcoin is mainly used to issue nonrefundable payments among users. However, there are a number of application scenarios where users need to make deposits (e.g., when using a service which requires assurance in case of damage or misuse). Bitcoin can plan for this case by enabling the creation of deposits to potentially untrusted entities.

As described in [20], a user A can make a deposit of v BTCs to entity B by constructing a transaction T_1 that spends v BTCs into an output address C in such a way that the signatures of both A and B are required to spend T_1. The user A does not immediately broadcast T_1 in the Bitcoin network; instead, A sends B $H(T_1)$, C, as well as a new address that is owned by A using an off-line channel (e.g., using a direct TCP connection). Upon reception of $H(T_1)$, B constructs another transaction T_2 that spends the BTCs stored in C (by linking it to $H(T_1)$) back to an address specified by A. T_2 is formed such that the $\mathrm{nLockTime}$ field is set to a future preagreed date, and the sequence number for the input is set to zero. B then sends T_2 to A using an off-line channel.

Subsequently, A verifies that T_2 is well-formed and signs T_1 and broadcasts both T_1 and T_2 in the network. At this stage, the v BTCs cannot be spent individually by either A or B. Once the date specified in $\mathrm{nLockTime}$ is reached, the contract is completed and A will receive the v BTCs back by spending transaction T_2 even

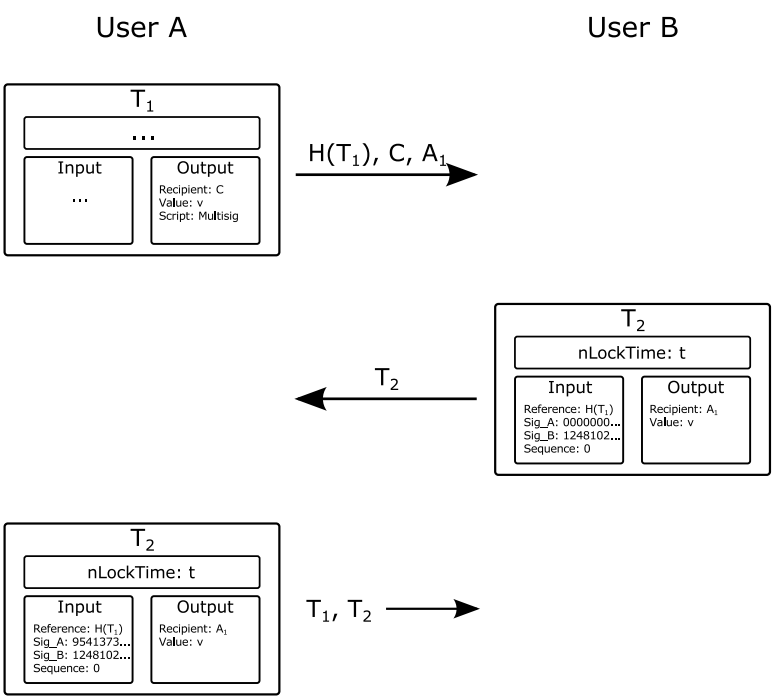

Figure 8.2 Making deposits in Bitcoin.

if B is not online. Note that by setting the sequence number to zero, the contract between A and B can be amended in future if both agree. This process is depicted in Figure 8.2.

8.2.5.2 Dispute Mediation

The aforementioned process of making deposits (see Section 8.2.5.1) can be inherently extended to deal with dispute mediation. For instance, A and B can agree on a neutral dispute meditator $\mathcal{M}$. Here, all transactions issued by A can be constructed so that they can be spent using the signatures of any two out of the three parties: A, B, and $\mathcal{M}$. We can now distinguish the following cases:

1. In case of a successful interaction between A and B, then the transactions will be spent as agreed in the contract.

2. In case some issue arises between A and B, then $\mathcal{M}$ acts as a mediator. If $\mathcal{M}$ aligns with A, then the transactions will not be spent. Otherwise, if $\mathcal{M}$ agrees with B, then the transactions will be spent to B's output address.

8.2.5.3 Managing Multiuser Funds

Bitcoin additionally enables different users to collaboratively raise funds for any given project without the need for an external arbiter, for example, to handle disputes (see Section 8.2.5.2). For instance, assume that different entities $A_1, \ldots, A_n$ decide to collaboratively raise funds of v BTCs in order to support a project. In this case, it is required that if v BTCs could not be jointly raised, then the funds committed by each entity should be reimbursed.

In this case, each of the entities $A_i, \forall i \in [1, n]$ can issue a transaction committing $v_1 < v$ of BTCs from one of their inputs to spend v BTCs to a common output address B. Clearly, this is not a correct transaction since the input amount is less than the output amount. The intuition here is that the input script is signed in such a way to allow an aggregator to combine all the various transactions issued by the different entities in a multi-input single-output transaction provided that at least v BTCs were raised by these entities; that is, the transaction is only valid if and only if $\sum_{i=1}^{n} v_i = v$. This can be achieved using the SIGHASH_ALL||SIGHASH_ANYONECANPAY that signs the entire transaction except any signature scripts, preventing modification of the signed parts. More detail about this process can be found in [20].

8.2.5.4 Using Smart Contracts for Crime

One major limitation of the aforementioned smart contracts is that they can also facilitate the mediation of illegal activities among distrustful criminal parties. Indeed, Bitcoin (and other decentralized frameworks such as Ethereum [21]) eliminate the need for trusted third-party intermediaries to conclude such contracts.

This would clearly render illegal activities and smart contracts established using Bitcoin and similar blockchain technologies harder for law enforcement agencies to trace/detect. While traditional approaches require the intervention of third-parties (which could be regulated and/or coerced by law enforcement agencies), Bitcoin does not necessarily require any third-party and minimizes interaction between criminal parties. This problem is even further exacerbated since Bitcoin inherently supports pseudonymity, which is an appealing property for criminal activities.

This problem was outlined by Juels et al. in [22]. More specifically, the authors show that criminal smart contracts (CSCs) for leakage of secrets are efficiently realizable in existing decentralized contracts. Additionally, the authors demonstrated that authenticated data feeds, another anticipated feature of smart contract systems, can facilitate CSCs for real world crimes. This indirectly motivates the need for policy safeguards to prevent the abuse of smart contracts.

8.3 CONCLUDING REMARKS

Bitcoins blockchain fueled innovation, and a number of innovative applications have already been devised by exploiting the secure and distributed provisions of the underlying blockchain.

This clearly shows the change of value enabled by the Bitcoin blockchain technology. Namely, while the original focus of the Bitcoin blockchain was to support money transfer among users, a number of applications built around the blockchain concern other possible use cases and scenarios. Prominent applications include secure time-stamping, secure commitment schemes, and smart contracts. Note that some of these extensions cannot be deployed without changing the code base of Bitcoin (i.e., via a hard fork). These are referred to as altcoins and preserve mining power by leveraging the already established Bitcoin community.

References

[1] Namecoin—Wikipedia, 2010. available from `https://en.wikipedia.org/wiki/Namecoin`.

[2] Namecoin—Domain Name Specification, 2010. available from `https://wiki.namecoin.info/index.php?title=Domain_Name_Specification`.

[3] Harry Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of Namecoin and lessons for decentralized namespace design. *WEIS '15: Proceedings of the 14th Workshop on the Economics of Information Security*, June 2015.

[4] Digital asset holdings acquires hyperledger and bits of proof. available from `http://digitalasset.com/posts/2015-06-25-hyperledger-acquired-by-digital-asset-holdings.html`.

[5] Archived webpage of digital asset holdings- hyperledger. available from https://web.archive.org/web/20150816190748/http://www.digitalasset.com/hyperledger/index.html.

[6] Hyperledger summary. available from https://web.archive.org/web/20150816190748/http://d1iohkh6wgqugq.cloudfront.net/static/resources/hyperledger-summary.pdf.

[7] Bits of proof. available from http://bitsofproof.com/.

[8] Bits of proof launches "red hat for bitcoin", 2013. available from http://www.coindesk.com/bits-of-proof-launches-red-hat-for-bitcoin/.

[9] Nakamoto's paper encoded in a Bitcoin transaction, 2014. available from https://blockexplorer.com/tx/54e48e5f5c656b26c3bca14a8c95aa583d07ebe84dde3b7dd4a78f4e4186e713.

[10] Hidden surprises in the Bitcoin blockchain, 2014. available from http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html.

[11] BTProof: Trusted timestamping on the Bitcoin blockchain, 2012. available from https://www.btproof.com/.

[12] Proof of Existence, 2012. available from https://www.proofofexistence.com/.

[13] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 475–490. IEEE Computer Society, 2014.

[14] Hovav Shacham and Brent Waters. Compact Proofs of Retrievability. In *ASIACRYPT*, pages 90–107, 2008.

[15] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, Zongren Liu, and Christian A. Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 831–843, New York, 2014. ACM.

[16] OneName, 2014. available from https://onename.com/.

[17] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, Zongren Liu, and Christian A. Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, November 3-7, 2014*, pages 831–843, 2014.

[18] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, and Franck Youssef. Transparent data deduplication in the cloud. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 886–900, New York, 2015. ACM. available from http://doi.acm.org/10.1145/2810103.2813630.

[19] Bitcoin as a public source of randomness, 2014. available from https://docs.google.com/presentation/d/1VWHm4Moza2znhXSOJ8FacfNK2B_vxnfbdZgC5EpeXFE/view?pli=1#slide=id.g3934beb89_034.

[20] Contracts—Bitcoin wiki, 2015. available from https://en.bitcoin.it/wiki/Contract.

[21] Ethereum Homestead Release. available from https://www.ethereum.org/.

[22] Ari Juels, Ahmed Kosba, and Elaine Shi. The ring of gyges: Using smart contracts for crime. available from http://www.arijuels.com/wp-content/uploads/2013/09/public_gyges.pdf.

Chapter 9

Blockchain Beyond Bitcoin

As mentioned in Chapter 8, there are currently more than 500 alternate blockchains, most of which are simple variants of Bitcoin.

Indeed, the blockchain instantiates a novel distributed consensus scheme which allows transactions, and any other data, to be securely stored and verified without the need for any centralized authority. Note that the entire community has been in search for a simple, scalable, and workable distributed consensus protocol for a considerable amount of time [1]. For instance, PoW-based blockchain is a permissionless system that does not require any identity management, and could scale to millions of miners. We contrast this to existing Byzantine Fault Tolerant (BFT) proposals, which are permissioned systems (i.e., they require the knowledge of the IDs of the miners prior to the start of the consensus protocol) and have limited scalability provisions.

However, Bitcoin's PoW has been often criticized due to its considerable waste of energy; indeed, the cost of mining per confirmed transaction is estimated to be 6.2$ at the time of writing. Existing studies also show that Bitcoin's blockchain can only achieve a modest transactional throughput bounded by seven transactions per seconds. To remedy the limitations of PoW, a number of alternative consensus protocols have been proposed, such as Ripple and Ethereum.

In this chapter, we overview a number of interesting blockchain proposals that are currently acquiring considerable attention in the media/literature.

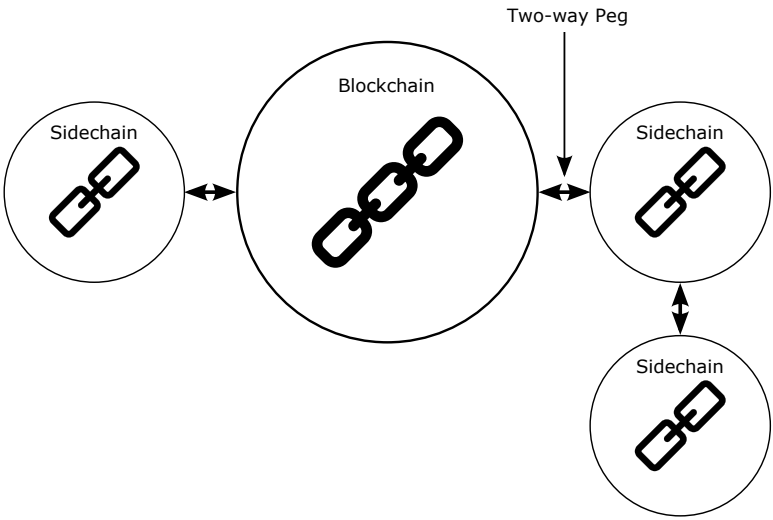

Figure 9.1 Sidechains are blockchains that are interoperable with Bitcoin and with each other. This allows assets to move freely across all blockchains.

9.1 SIDECHAINS

Sidechains are blockchains that can be interoperable with Bitcoin and with each other. This allows assets to move freely across all blockchains.

As mentioned earlier, a number of variants of Bitcoin were implemented as *altcoins* in order to provide enriched applications or better performance and security features. Those altcoins were maintained independently using a variant Bitcoin codebase and introduce their own currencies that are independent of Bitcoin. Clearly, this results in liquidity shortage and market fluctuation, since all altcoins are competing for market assets—which also discourages technical innovations for new altcoins.

Sidechains (see Figure 9.1) attempt to remedy this challenge. Namely, pegged[1] sidechains implement an infrastructure featuring interoperable blockchains where parties can easily switch and work with different blockchains. Here, SPV proof is used to transfer a coin from one sidechain (the *parent chain*) to the other via two waiting periods. First, a coin is locked by a transaction on the parent chain

1 The term "pegged" is used to emphasize that a sidechain supports coin transfer back and forth between sidechains.

until the *confirmation period* ends. In the meantime, a transaction will be created on the sidechain along with an SPV proof referring to the locked coin on the parent chain. Finally, the user must wait for the *contest period* before the newly transferred coin can be spent on the sidechain. With the peg protocol, the underlying currency can be reused within different blockchains (as well as most of the blockchain implementation).

Since sidechains are typically independent blockchains, this allows developers to test beta versions of the system as a sidechain without affecting the experience witnessed by users.

At the time of writing, the Elements Project [2] is exploring extended sidechains' features such as *Confidential Transactions* (to improve payer privacy), *Segregated Witness* (to prevent transaction malleability), and *New Opcodes* (to provide more powerful scripts for smart contracts). Those new features are regarded as *elements* and can be combined within a sidechain. For example, Liquid [3] is a sidechain that integrates *Confidential Transaction* among other elements. Liquid is released by Blockstream [4], the founder of the Elements Project.

Despite the flexibility that sidechains can provide, the security of different sidechains still needs to be independently addressed. This means that the sidechains are competing for the mining power in the market, which leaves newly introduced sidechains rather vulnerable to attackers (e.g., the new sidechain is typically supported by a small fraction of the computing power—which can be easily surpassed by the attacker). Therefore, one ought to be cautious when assets are transferred to a sidechain that exhibits such weaker security guarantees. To remedy this, merged mining [5, 6] was proposed to allow different blockchains to share their mining power by including the blocks of other chains during the mining process.

Another challenge of sidechains is that interchain transactions may incur high latencies, as the waiting periods are required to ensure that the transactions remain in the blockchain.

9.2 ETHEREUM

Similarly to Bitcoin, Ethereum leverages proof-of-work blockchain technology to achieve distributed consensus. By providing a fully fledged Turing-complete programming language instead of Bitcoin's simple scripting language, Ethereum allows arbitrary applications referred to as smart contracts to be run on its blockchain. For example, a basic Namecoin version for the Ethereum blockchain can be written

with a few lines of code. Creating subcurrencies only requires minimal programming effort as well. Another concrete use case of Ethereum is to build decentralized autonomous organisations (DAOs).

Ethereum in fact provides a decentralized platform to build smart applications. The contracts run exactly as programmed, with no possibility of downtime, censorship, fraud, or third-party interference [7].

In what follows, we discuss the basic operations of Ethereum in greater detail.

9.2.1 Accounts

Similar to Bitcoin, Ethereum features accounts that reside at specific addresses on the blockchain. At the time of writing, Ethereum provides two types of accounts: (1) externally owned accounts (EOAs), and (2) contract accounts. EOAs are controlled by private keys and akin to Bitcoin accounts. Contract accounts, on the other hand, are autonomous objects. They have associated code and persistent storage. Their code is executed whenever they receive a transaction or message (see Section 9.2.3).

Both account types have a balance field indicating the amount of *Ether* (Ethereum's internal currency) that the account currently possesses. Note the difference to Bitcoin where account balances are implicitly given by unspent transaction outputs (UTXOs). Moreover, accounts maintain a nonce field expressing the number of transactions that they have issued so far.

The nonce, the balance, and the hashes of the account's storage and code define the *account state*.

9.2.2 Transactions and Messages

Ethereum transactions correspond to data packages that are signed by the private key of the issuing EOA. Transactions contain the recipient, a signature identifying the sender, the amount of Ether to be transferred along with the transaction, and an optional data field.

Resources used for transaction execution are subject to fees. The corresponding unit is *gas*. Transactions include two further fields, which are related to gas. The *startGas* value specifies the maximal amount of gas that the execution of the transaction may consume. The *gasPrice* value indicates how to convert between Ether and gas.

Finally, transactions comprise a nonce field, whose value has to match the sender's account current nonce. This is necessary to prevent replay attacks.

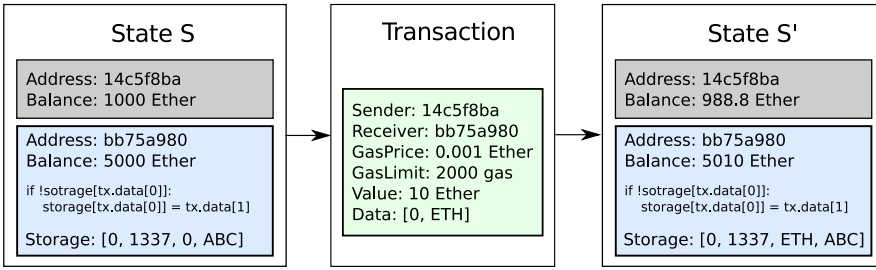

Figure 9.2 Transaction execution in Ethereum.

Note that in Ethereum, messages are constructed similarly to transactions, but are sent from contract accounts.

9.2.3 State and Transaction Execution

Ethereum is a state machine where transaction execution triggers a state transition. The Ethereum *state* is a mapping between account addresses and account states. The state is not stored within the blockchain, but maintained by the clients.

An exemplary transaction execution is depicted in Figure 9.2. Note that transaction execution is completely deterministic.

In this example, the transaction limits the amount of gas available for its execution to 2,000. The corresponding amount of Ether is then subtracted from the sender's balance up front. The sender has to pay for each byte in the transaction. We assume that this consumes 1,000 gas. In the next step, 10 Ether are transferred from the sender's balance to the receiver's balance. Since the receiving account is a contract, its associated code is triggered. Here, the account's storage is modified. Assuming that the code execution costs 200 gas, the remaining 800 gas are then refunded to the sender (in Ether).

Making all resource consumption subject to fees and limiting the amount of gas that can be spent during the execution of a transaction prevents infinite loops, which are necessary to prevent DoS attacks.

9.2.4 Blocks

A block is a collection of data consisting of a header, the transactions incorporated within the block, and a list of uncles (stale blocks). Ethereum uses a generalized notion of uncles, where an uncle is a direct child of a kth generation ancestor of

the including block, for $2 \leq k \leq 7$. An uncle cannot be a direct ancestor of the including block.

Block headers include a hash of the state (after all transactions in the block are executed) and the difficulty level of the block. Among other fields, block headers also contain the hash of the parent block's header and the beneficiary's address. The beneficiary is the account that is rewarded for successfully mining the block.

9.2.5 Mining and Blockchain

Similar to Bitcoin, mining is used to (1) confirm transactions and (2) issue Ether postlaunch. Mining consists of brute-forcing the header's nonce until the *Proof-of-Work* (PoW) algorithm output is below a certain threshold. Since this threshold is based on the block's difficulty, mining gives both meaning and credence to the notion of difficulty [8]. The difficulty level is dynamically adjusted to achieve an average block generation time of approximately 15 seconds [9].

Note that the structure resulting from mining is a block tree. Consensus is needed on the path, starting from the genesis block, that should be selected as the blockchain. This is achieved by choosing the path that exhibits the highest total difficulty and thus possesses the highest amount of computation backing it. Similar to Bitcoin, an attacker aiming at rewriting the history would need to outperform the honest part of the Ethereum network.

Successful mining of a winning block is rewarded in multiple ways. A static block reward of 5 Ether is added to the beneficiary's balance. The beneficiary also receives the gas expended by executing all transactions in the block. Finally, an extra reward is given for including uncles. Contrary to Bitcoin, miners of uncles are rewarded in Ethereum as well, and receive 7/8 of the static block reward [9].

Ethereum's PoW is expected to incorporate the GHOST protocol. GHOST [10] is an alternative to the longest chain rule for establishing consensus in PoW-based blockchains and aims to alleviate the negative impacts of stale blocks by incorporating the difficulty of uncle blocks when measuring the longest chain.[2]

Note that it is also envisioned that the PoW consensus utilized by Ethereum will be replaced by *virtual mining* in the form of *Proofs of Stake*, where each entity has to deposit some stake in the system to be held accountable in case of misbehavior.

2 At the time of writing, Ethereum does not incorporate the difficulty of uncle blocks when measuring the longest chain.

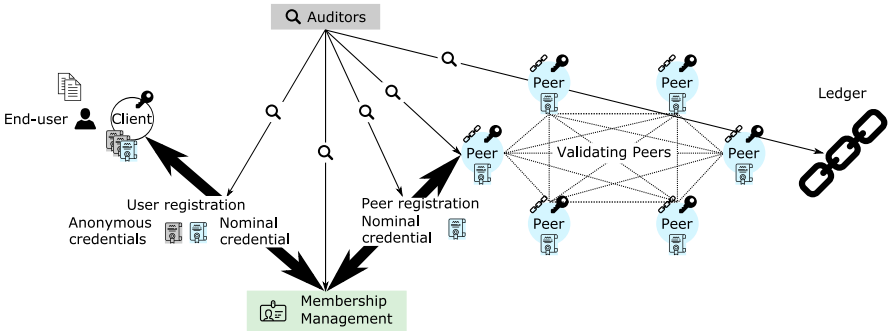

Figure 9.3 Basic architecture of Open Blockchain.

9.3 OPEN BLOCKCHAIN

Open Blockchain is a prominent project featuring an enterprise blockchain. Open Blockchain was originated by IBM, but later evolved as an open source project within the Hyperledger community [11].

Open Blockchain is a *permissioned* blockchain network, where end users or organizations go through a *registration* process that would authorize them to submit (as *users*) or process transactions (as *validators*) that are announced in the system. The permissioned nature of Open Blockchain allows the latter to mainly rely on nonproof-of-work based mechanisms, such as Byzantine fault-tolerant protocols, to achieve consensus in the network on transaction validation. The system has been designed in a modular way, allowing the easy replacement of these protocols with other protocols that can achieve consensus among participating nodes in the system.

The architecture of Open Blockchain is summarized in Figure 9.3. Similar to Ethereum, Open Blockchain aims at accommodating arbitrary logic within its transactions. To do so, the concept of *chaincode* was introduced; chaincode refers to pieces of code that are to be *deployed* and registered within the blockchain through *deploy* transactions and can be *invoked* through *invoke* transactions. Chaincode deployment is performed within a Docker container, within which subsequent invocations of that chaincode are also accommodated. Docker provides a secured, lightweight method to sandbox chaincode execution. Note that the chaincode concept is more general than the notion of smart contracts. Namely, Chaincode can be written in any mainstream programming language, and executed in containers

inside the Open Blockchain context layer. Chaincode provides the capability of restricting the functionality of the execution environment and the degree of computing flexibility to satisfy potential legal contractual requirements.

As depicted in Figure 9.3, the Open Blockchain system consists of the following entities:

- *Membership management infrastructure.* This denotes a set of entities that are responsible for identifying an individual user (using any form of identification considered in the system such as credit cards or identity cards). This infrastructure is also responsible for opening user accounts and issue the necessary credentials to successfully create transactions and deploy/invoke chaincodes successfully through Open Blockchain.
- *Peers.* These are classified as validating peers and nonvalidating peers. Validating peers (also known as validators) order and process (check validity, execute, and add to the blockchain) user-messages (transactions) submitted to the network. On the other hand, nonvalidating peers (or simply peers) receive transactions on behalf of users, and after performing validity checks, they forward the transactions to their neighboring validating peers. Peers maintain an up-to-date copy of the blockchain, but in contradiction to validators, they do not validate transactions (a process also known as transaction validation).
- *End users of the system.* These refer to the users that have registered to the membership service administration after having demonstrated ownership of what is considered identity in the system and having obtained credentials to install the client-software and submit transactions to the system.
- *Client-software.* This refers to the software that needs to be installed at the client side for the latter to be able to complete his or her registration to the membership service and submit transactions to the system. For simplicity, in the following, we will refer to the client-software by client.

9.3.1 Membership Services

One of the main security challenges in Open Blockchain is reconciling transactional privacy with identity management in order to enable competitive institutions to transact effectively on a common blockchain (for both intra- and interinstitutional transactions). This is achieved by forming a privacy-preserving permissioned network. The *permissioned* nature of this system is achieved as follows:

- Define a policy to determine the conditions under which a new entity can participate in transaction processing, evaluation, and validation (i.e., join the set of *validators* or *endorsers* of the system).
- Issue system certificates to all entities that are eligible to join the set of *validators* or *endorsers*, according to a predefined policy.
- Instruct validators to accept transaction evaluation-related messages from only certified validators of the system.
- Define a policy to determine the conditions under which a new entity can join the set of *users* of the system (i.e., submit transactions to the network of validators). That is, announce messages that would allow them to *deploy* Open Blockchain contracts, referred to as *chaincodes*, or *invoke* already deployed chaincodes.
- Issue system certificates to all entities that request membership to Open Blockchain system and fulfill the conditions listed in the corresponding policy.
- Instruct validators to only accept transactions that are *authenticated* to originate by properly formed certificates.

Membership services registration is an off-line process through which users prove that they satisfy the membership conditions defined in the associated Blockchain membership policy. To facilitate needs for privacy-preserving transactions, upon proper registration users can issue two types of credentials.

- Enrollment certificates (ECerts) that carry user's identity or long-term user-identifier in the system, also referred to by *enrollment identity*.
- Transaction certificates (TCerts) that faithfully but pseudonymously represent enrolled users. That is, transaction certificates do not carry the enrollment identity of their owner in the clear; on the contrary, enrollment identity of the owner of a transaction certificate is incorporated in the transaction certificate such that the former can only be revealed/proven with the consent of the transaction certificate owner or an auditor.

During enrollment, users generate two key-pairs. The first is an ECDSA key-pair that facilitates authentication of a long-term user identity. The second is an elliptic curve Diffie Hellman (ECDH) key-pair that allows users to establish secret communication channels within Open Blockchain transactions. Both key-pairs are generated at the client side. The secret signing and decryption keys are maintained in user premises and never shared with the membership service authorities. On the

other hand, the public signature verification, and encryption keys generated during a user enrollment, are encapsulated in the user's enrollment certificate.

Transaction certificates are issued upon an enrolled user request, and as discussed previously, contain the identity of their owner in encrypted form. Transaction certificates contain public signature keys, to enable a transaction certificate owner authentication in transactions. These keys are generated using the enrollment signature key-pair as a basis, and their secret material is also solely accessible by their owner.

Users can use their certificates to authenticate themselves as valid users of the system. In particular, Open Blockchain membership protocols guarantee the following security properties:

- *Unforgeability of proof of certificate owner.* That is, a computationally bounded attacker is not able to prove ownership of a credential he or she is not owning without the collaboration of the credential's owner. This property also requires that information endorsed by a certificate cannot be altered in such a way that the same certificate appears to have properly endorsed something else without the collaboration of the certificate's owner.
- *Nonrepudiation.* Users equipped with a certificate to endorse a piece of information (e.g., a transaction) cannot deny having generated the endorsement (i.e., repudiate their endorsement).
- *Nonframeability.* A computationally bounded attacker is not able to create an endorsement to a message that appears to have been generated by a certificate that it does not own.
- *Anonymity.* Anonymity requires that a certain transaction certificate or transaction certificate endorsement on any message does not distinguish the identity of a signer with better probability than choosing this identity at random among the members of the system.
- *Unlinkability.* Unlinkability requires that a set of transaction certificates or certificate endorsements cannot be linked together as having been generated by the same identity.

In the current version of Open Blockchain, the user and validator registration, and the associated issuing of credentials in Open Blockchain are performed by membership services that leverage a few trusted membership authorities (all of

which are potentially controlled by the same central authority). In particular, membership services in Open Blockchain are facilitated with the help of four (trusted) entities:

- *Registration authority.* The registration authority evaluates candidate system participant credentials and adds these entities to the database of registered users or validators. The database is automatically shared with the other components of membership services.
- *Enrollment certificate authority.* This authority issues enrollment certificates upon registered users' or validators' request.
- *Transaction certificate authority.* This authority issues transaction certificates upon enrolled users' requests.
- *TLS certificate authority.* This authority issues TLS certificates to registered users/validators. It also issues TLS certificates to the other components of the membership service.

At the time the time of writing, there were discussions of moving the functionality of membership services in Open Blockchain to a decentralized network.

9.3.1.1 Consensus Mechanism

At the time that Open Blockchain was conceived, existing open-source systems (e.g., Bitcoin, Ethereum) offered transaction validation rates in the order of tens of transactions per second. For instance, Bitcoin allowed for a maximum transaction validation rate of approximately seven per second. Depending on the use cases, such rates can be considered modest. Open Blockchain relies on algorithms to facilitate *consensus* over *validated* transactions that would offer much higher transaction validation rates.

Namely, Open Blockchain uses the practical Byzantine fault-tolerant (PBFT) algorithm to allow validator members of the chain to agree on the total order of transactions announced throughout the chain and the global state (i.e., the entire chaincode's state). At a high level, PBFT guarantees consensus safety, as long as at most one third of the validators in the chain can be *Byzantine* (i.e., not abide with the consensus protocol rule). An extension of this protocol has been in place (Sieve [12]) to allow for the exclusion of nondeterministic transactions.

9.3.2 Transactions Life-cycle

In Open Blockchain, transactions are created on the client side. The client can be either a plain client or a specialized application (i.e., a piece of software that handles (server) or invokes (client) specific chaincodes through the blockchain).

Developers of new chaincodes create a new *deploy transaction* by passing to the (fabric) infrastructure:

1. The confidentiality/security version or type they want the transactions associated to the new chaincode to conform with, the set of users who wish to be given access to parts of the chaincode and a proper representation of their (read) access rights. Transactions that are marked as confidential will contain a number of encrypted fields that can only be decrypted by the appropriate entities as defined in the access policy.
2. The code associated to the new chaincode.
3. Metadata associated and provided to the chaincode at execution time. This may contain configuration parameters, or, in some cases, key-material.
4. Application metadata, which corresponds to metadata attached to the transaction format that only the application can interpret and handle.

Other types of transactions, such as *invoke* and *query* transactions, which require confidentiality, are also created using a similar approach. More specifically, in both cases, the issuer provides the identifier of the chaincode to be executed, the name of the function to be invoked, and the invocation arguments. Optionally, the invoker can pass to the transaction creation function and the code invocation metadata that will be provided to the chaincode at the time of its execution. Transaction metadata is another field that the application or the invoker can additionally leverage.

Finally, transactions at the client side are signed by a certificate of their creator and released to the network of validators.

Validators receive the confidential transactions and pass them through the following phases:

- *Prevalidation phase.* In this phase, validators validate the transaction certificate against the accepted root certificate authority, verify the transaction certificate signature included in the transaction (statically), and check whether the transaction is a replay.

- *Consensus phase.* In this phase, the validators add this transaction to the total order of transactions (ultimately included in the ledger).
- *Pre-execution phase.* Here, validators verify the validity of the transaction/enrollmer certificate against the current validity period, decrypt the transaction (if the transaction is encrypted), and check that the transaction's plaintext is correctly formed (e.g., invocation access control is respected, included correctly formed TCerts). Preliminary replay-attack check is also performed here within the transactions of the currently processed block.
- *Execution phase.* In this phase, the (decrypted) chaincode is passed to a container, along with the associated code metadata, and (encrypted) updates of the chaincodes state are committed to the ledger with the transaction itself.

9.3.2.1 Confidentiality of Transactions

Transaction confidentiality requires that the plaintext of a chaincode (i.e., code or description) is not accessible or inferable (assuming a computationally bounded attacker) by any unauthorized entities (i.e., user or peer not authorized by the developer). It is thus important here that the chaincodes of both deploy and invoke transactions that remain concealed whenever confidentiality is required. In the same spirit, nonauthorized parties should not be able to associate invocations (invoke transactions) of a chaincode to the chaincode itself (deploy transaction) or associate these invocations to each other.

Confidentiality mechanisms in Open Blockchain allow the deployer of a chaincode to grant (or restrict) access of an entity to any subset of the following parts of a chaincode:

- Chaincode content (i.e., complete code) of the chaincode.
- Chaincode function headers (i.e., the prototypes of the functions included in a chaincode).
- Chaincode invocations and state (i.e., successive updates to the state of a specific chaincode) when one or more functions of its are invoked.
- All of the above.

Note that this design offers to the application the capability to leverage the membership service infrastructure and its public key infrastructure to build their own access control policies and enforcement mechanisms.

At the time of the writing, the confidentiality features of Open Blockchain are restricted to ensuring confidentiality against nonauthorized users of the system. As such, validating entities are currently considered trusted to access the plaintext of all chaincode resources and not to share it with nonauthorized entities.

To support read-access control of the plaintext of a chaincode, Open Blockchain transaction confidentiality protocols leverage the public encryption keys bound to user identities at enrollment time, as well as a chain-specific encryption public key. That is, for confidentiality purposes, a chain is bound to a single long-term encryption key-pair $(\mathrm{pk}_{\mathrm{chain}}, \mathrm{sk}_{\mathrm{chain}})$. This key-pair is generated and maintained within the premises of the membership service infrastructure.

At the deploy time of a confidential chaincode, the payload of the transaction (i.e., chaincode code) as well as the associated metadata are encrypted using a freshly generated chaincode-specific key. The key is passed to the authorized users and validators through messages encrypted with the authorized entities associated with the public encryption key. In addition, the chaincode creator specifies at deployment time the key to be used for the encryption of the chaincode's state each time that chaincode is invoked. Again, access to this key is given to authorized entities using their public keys.

Subsequent invocations of a confidential chaincode are forced to encrypt their payload using the key-material defined at deployment time.

9.3.2.2 Auditing Capabilities

Open Blockchain offers auditability in two layers: (1) at the system level, where auditors are able to monitor all transactions a certain validator is involved in and (2) at chaincode level to allow an auditor to read and access all transactions associated to a specific chaincode. At the same time, an auditor who is authorized to query and get responses back from membership service on a certain user's transactions is able to get proof of a certain user's involvement in the chain. Proper key-hierarchies allow for different keys to be used in each transaction while minimizing the number of keys managed at the client-side.

9.3.3 Possible Extensions

Open Blockchain was recently renamed Hyperledger/fabric as soon as it was shortlisted as one of the candidates of becoming Linux Foundation's Hyperledger.

There are currently a number of discussions within the Hyperledger community on the best ways to evolve Open Blockchain according to the consortium's needs.

Current proposals for extending Open Blockchain discussions include the decentralization of membership services, the separation of the consensus (agreement on the total order of transactions and execution of the included chaincodes), as well as extending the confidentiality guarantees to the endorsing entities.

9.4 RIPPLE

The wide success of Bitcoin has lead to a surge of a large number of alternative cryptocurrencies. These include Litecoin [13], Namecoin [14], and Ripple [15, 16], among others. Most of these currencies simply clone the Bitcoin blockchain and try to address some of the shortcomings of Bitcoin. As described in Chapter 8, Namecoin offers the ability to store data within a PoW blockchain in order to realize a decentralized open-source information registration based on Bitcoin, while Litecoin primarily differs from Bitcoin by having a smaller block generation time and a larger number of coinbases. While most of these digital currencies are based on Bitcoin, Ripple has evolved almost completely independently of Bitcoin (and of its various forks). Currently, Ripple holds the second highest market cap after Bitcoin [17]. This corresponds to almost 20% of the market cap held by Bitcoin. Ripple Labs have additionally finalized the financing of a $30 million funding round to support the growth and development of Ripple [18].

Ripple does not only offer an alternative currency, *XRP*, but also promises to facilitate the exchange between currencies within its network. Although Ripple is built upon an open-source decentralized consensus protocol, the current deployment of Ripple is solely managed by Ripple Labs. At the time of writing, Ripple claims to have a total network value of approximately $ 960 million with an average of almost 170 accounts created per day since the launch of the system [19]. Moreover, there are currently a number of businesses that are built around the Ripple system [20, 21]. For instance, the International Ripple Business Association currently deploys a handful of Ripple gateways [22], market makers [23], exchangers [24], and merchants [25] located around the globe.

In the remainder of this section, we overview the Ripple protocol and discuss the basic differences between the current deployments of Ripple and Bitcoin.

9.4.1 Overview of Ripple

Ripple [15] is a decentralized payment system based on credit networks [26, 27]. The Ripple code is open source and available for the public; this means that anyone can deploy a Ripple instance. Nodes can take up to three different roles in Ripples: *users* that make/receive payments, *market makers* that act as trade enablers in the system, and *validating servers* that execute Ripple's *consensus* protocol in order to check and validate all transactions taking place in the system.

Ripple users are referenced by means of pseudonyms. Users are equipped with a public/private key pair; when a user wishes to send a payment to another user, it cryptographically signs the transfer of money denominated in Ripple's own currency, XRP, or using any other currency. For payments made in non-XRP currencies, Ripple has no way to enforce payments, and only records the amounts owed by one entity to the other. More specifically, in this case, Ripple implements a distributed credit network system.

A non-XRP payment from A to B is only possible if B is willing to accept an *I owe you* (IOU) transaction from A (i.e., B trusts A and gives enough credit to A). Hence, A can only make a successful IOU payment to B if the payment value falls within the credit balance allocated by B to A. This may be the case, for example, if the participants know each other or if the involved amounts are rather marginal; typically however, such transactions require the involvement of market makers who act as intermediaries. In this case, enough credit should be available throughout the payment path for a successful payment.

For example, a trust line can be established between market maker $U1$ and A (see Figure 9.4) by A depositing an amount at $U1$. In this example, A wants to issue a payment to B with the amount of 100 USD. Here, the payment is routed from $A \rightarrow U1 \rightarrow U2 \rightarrow U4 \rightarrow B$. This is possible because available credit lines are larger than the actual payment for every atomic transactions. Note that we did not route through $U3$ as there is not enough credit available between $U1 \rightarrow U3$. However, we note that it is possible to break down the payment amount at $U1$, route a payment below 90 USD through $U1 \rightarrow U3 \rightarrow B$, and transfer the rest through $U1 \rightarrow U2 \rightarrow U4 \rightarrow B$ (extra fee at $U3$ required). In typical cases, Ripple relies on a path-finding algorithm that finds the most suitable payment path from the source to the destination. By implementing credit networks, Ripple can act as an exchange/trade medium between currencies; in case of currency pairs that are traded rarely, XRP can act as a bridge between such currencies.

9.4.1.1 Ripple's Ledger

Ripple maintains a distributed ledger that keeps track of all the exchanged transactions in the system. These ledgers are similar in spirit to Bitcoin blocks but are created every few seconds and contain a list of transactions to which the majority of *validating servers* has agreed. This is achieved by means of Ripple's consensus protocol [15] that is executed among validating servers. This protocol is based on a crash-tolerant consensus protocol and does not deal with Byzantine nodes. This consensus protocol implicitly assumes the presence of rational (but not Byzantine) validating servers. The main intuition is that Ripple captures a closed system of validating servers that are not anonymous. Although Ripple originally had the intention of realizing an open system where any entity can set up and connect their servers, Ripple's use cases involve private deployments within a fixed set of (known) entities (such as banks and financial institutions). In this setting, rational servers would not risk misbehaving since all messages are authenticated and cannot be later refuted; if detected then these servers can simply be banned/sued. More specifically, Ripple claims that, in case of forks, then the entire logs of communication between servers will be exposed/analyzed to identify and isolate malicious servers.

A Ripple ledger consists of the following information: (1) a set of transactions, (2) account-related information such as account settings, total balance, trust relation, (3) a time stamp, (4) a ledger number, and (5) a status bit indicating whether the ledger is validated or not. The most recent validated ledger is referred to as the *last closed ledger*. On the other hand, if the ledger is not validated yet, the ledger is deemed *open*.

9.4.1.2 Consensus and Validating Servers

We now briefly overview the consensus protocol of Ripple.

Each validating server verifies the proposed changes to the last ledger; changes that are agreed by at least 50% of the servers are packaged into a new proposal that is sent to other servers in the network. This process is reiterated with the vote requirements increasing to 60%, 70%, and 80%, after which the server validates the changes and alerts the network of the closure of the last ledger. At this point, any transaction that has been issued in the network but did not appear in the ledger is discarded and can be considered as invalid by Ripple users. Such transactions typically need then to be rebroadcasted in the network in order to be included in subsequent ledgers. Each validating server maintains a list of trusted servers known as Unique Node List (UNL); servers only trust the votes issued by

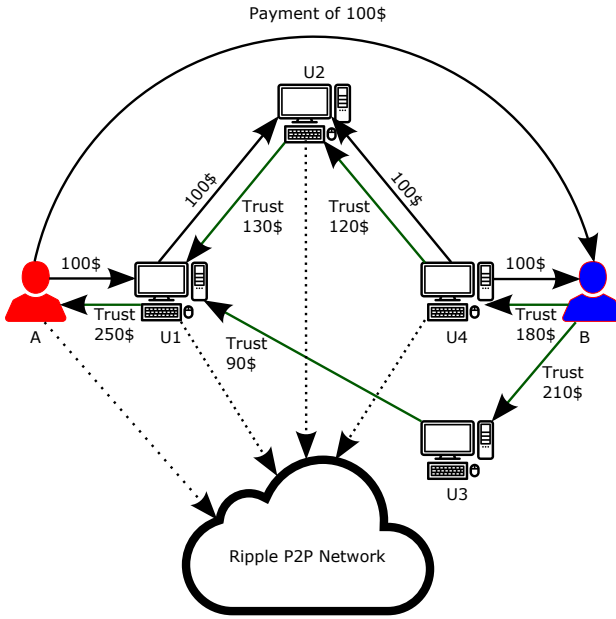

Figure 9.4 Example of IOU payments in Ripple. Here, A wants to pay 100 USD to B.

other servers which are contained in their UNL. More detail on Ripple's consensus protocol can be found in [28].

By doing so, Ripple enables different institutions (e.g., banks that run their own servers) to reach a consensus with respect to the fate of financial transactions. For instance, recently, Ripple Labs sealed a partnership agreement with a number of banks that agreed to adopt Ripple's open-source distributed transaction infrastructure [29].

9.5 COMPARISON BETWEEN BITCOIN, RIPPLE, ETHEREUM, AND OPEN BLOCKCHAIN

In what follows, we briefly discuss the security and privacy provisions of Ripple, Ethereum, and Open Blockchain in relation to the well-investigated Bitcoin system.

9.5.1 Security

Similar to Bitcoin, Ripple, Ethereum, and Open Blockchain relies on ECDSA signatures to ensure the authenticity and nonrepudiation of transactions in the system. Furthermore, since Ripple and Ethereum are open systems (like Bitcoin), all transactions and their orders of execution are publicly available. This ensures the detection of any double-spending attempt (and of malformed transactions). Open Blockchain, on the other hand, is a closed, permissioned enterprise blockchain, where transactions can only be seen by the registered participants.

Consensus in Ripple and Open Blockchain are achieved by requiring that the validating servers check the log of all transactions in order to select and vote for the correct transactions in the system. In this way, these systems adopt a voting scheme across all validating servers (one vote per each validating server). As mentioned earlier, Open Blockchain relies on the PBFT consensus protocol, which ensures security even if 33% of the validators are Byzantine. On the other hand, the consensus layer in Ripple is closer requires that the transactions for which (80% of) the validators agree upon are considered to be valid [30]. Ripple Labs claims that it is easy to identify colluding validators and recommend that users choose a set of heterogenous validators that are unlikely to collude. Note that if validators in Ripple refuse to come to a consensus with each other, this is detectable by other validators, which then pronounce the network broken. In this case, the only way to resolve the problem would be to manually analyze the signed validations and proposals to see which validators were being unreasonable and for all honest participants to remove those validators from the UNLs (i.e., from the lists of validators they try to come to a consensus with). As far as we are aware, there is no formal security treatment of the correctness of Ripple's consensus protocol; this protocol has recently received some criticism [31, 32]. In [28], Armknecht et al. show that the current choice of parameters does not prevent the occurrence of forks in the system.

In contrast, transaction security in Bitcoin and Ethereum is guaranteed by means of proof-of-work which replaces the vote per validating server notion of Ripple and Open Blockchain, with a vote per computing power of the miners that are solving the PoW. Unlike Ripple and Open Blockchain, once transactions are confirmed in the global ledger (i.e., once transactions receive six confirmation blocks), it is computationally infeasible to modify these transactions [33] as long as the majority of the computing power in the network is honest. In contrast, in Ripple and Open Blockchain, if at any instant in time the majority of the validating servers becomes malicious, then they can rewrite the entire history of transactions in the system. For instance, at the time of writing, there are only a handful of Ripple

validating servers that are mostly maintained by the Ripple Labs; if these servers are compromised, then the security of Ripple is at risk.

9.5.2 Consensus Speed

In Bitcoin, payments are confirmed by means of PoW in Bitcoin blocks every 10 minutes on average. A study in [34] has shown that the generation of Bitcoin blocks follows a geometric distribution with parameter 0.19. This means that, since transactions are only confirmed after the generation of six consecutive blocks, then a payment is only confirmed after 1 hour on average. Although Bitcoin still recommends merchants to accept fast payments—where the time between the exchange of currency and goods is short (i.e., in the order of few seconds)—several attacks have been reported against fast payments in Bitcoin (see Chapter 4); a best-effort countermeasure has also been included in the Bitcoin client [34].

Although Ethereum relies on PoW to achieve consensus, blocks are generated in Ethereum every 12 seconds, on average. This ensures considerably faster convergence on consensus when compared to Bitcoin. Recent studies have however shown that by relying on a faster convergence interval, Ethereum offers weaker security guarantees when compared to Bitcoin [35]. On the other hand, Ripple and Open Blockchain inherently support fast consensus; almost all ledgers are closed within few seconds. This also suggests that payments in Ripple can be verified after few seconds from being executed.

9.5.3 Privacy and Anonymity

Ripple, Ethereum, and Bitcoin are instances of open-payment systems. In an open-payment system, *all* transactions that occur in the system are publicly announced. Here, user anonymity is ensured through the reliance on pseudonyms and/or anonymizing networks, such as Tor [36]. Users are also expected to have several accounts (corresponding to different pseudonyms) in order to prevent the leakage of their total account balance. Note that in Bitcoin, transactions can take different inputs, which originate from different accounts. This is not the case in Ripple, in which payments typically have a single account as input.

Although user identities are protected in Ripple, Ethereum, and Bitcoin, the transactional behavior of users (i.e., time and amount of transactions) is leaked in the process since transactions are publicly announced in the system. In this respect, several studies (see Chapter 5) have shown the limits of privacy in open-payment systems [37–39]. There are also several proposals for enhancing user privacy in

these systems; recently, a secure privacy-preserving payment protocol for credit networks that provides transaction obliviousness has been proposed [27].

Open Blockchain, on the other hand, is a permissioned system, where nodes need to register in order to participate in the system. Open Blockchain also relies on an identity manager to authenticate and authorize the participation of nodes and validators in the process. However, Open Blockchain can support encrypted transactions—effectively achieving transaction unlinkability and protecting the privacy of participants from other participants. At the time of writing, Open Blockchain does not offer transaction unlinkability or confidentiality against curious validators who are typically equipped with the appropriate secret material to decrypt all transactions. Moreover, anonymity is not supported in Open Blockchain by design in order to cater for a permissioned Enterprise deployment.

9.5.4 Clients, Protocol Update, and Maintenance

Ripple, Ethereum, Open Blockchain, and Bitcoin are currently open source, which allows any entity to build and release its own software client to interface with either systems. The official clients for Ripple, Ethereum, and Bitcoin are however maintained and regularly updated by the Ethereum foundation, the Bitcoin foundation, and Ripple Labs, respectively. Open Blockchain has been mainly an effort initiated by IBM, but now evolves with the help of the Hyperledger community.

Bitcoin and Ethereum clients can also run on resource-constrained devices such as mobile phones—owing to their support for simple payment verification. As far as we are aware, there exists no secure lightweight version of Ripple and Open Blockchain.

Note that all changes to the official Bitcoin client are publicly discussed in online forums, well justified, and voted among Bitcoin developers [40]. This process is however less transparent in Ripple and Ethereum.

9.5.5 Decentralized Deployment

Ripple, Ethereum, Open Blockchain, and Bitcoin leverage completely decentralized protocols. Similar to Bitcoin, we argue that the current deployment of Ethereum and Ripple is also centralized.

Similar to Bitcoin, only a handful of entities can control the security of all Ethereum transactions. More specifically, a quick look at the distribution of computing power in Ethereum shows that currently the top three (centrally managed) mining pools control more than 55% of the computing power in the network. On

the other hand, Open Blockchain aims for an enterprise deployment and credits control to the operator(s) of the validators.

Finally, in the case of Ripple, most validating servers are run by Ripple Labs at the time of writing. Although there are a few other servers that are run by external entities, the default list of validating servers for all clients points to the ones maintained by Ripple Labs. This also suggests that Ripple Labs can control the security of all transactions that occur in the Ripple system. Moreover, Ripple Labs and its founders retain a considerable fraction of XRPs; this represents the largest holdback of any cryptocurrency [17] and suggests that Ripple Labs can currently effectively control Ripple's economy. We contrast this to Bitcoin, where the current system deployment is not entirely decentralized, yet the entities that control the security of transactions, the protocol maintenance and update, and the creation of new coins are distinct [40]. In Ripple, the same entity, *Ripple Labs*, controls the fate of the entire system.

In [28], it was shown that—although it has been introduced almost 2 years ago—Ripple is still far from being used as a trade platform. Ripple advertises a large number of active accounts [19]. However, there is no strong evidence that users are active in Ripple; most accounts contain a small number of XRPs—which users could have received from one of the many giveaways organized by Ripple Labs [41]. Moreover, although the number of transactions in Ripple seems to be considerably increasing over time, the number of actual payments in the system is only marginally increasing over time and is dominated by direct XRP payments. Finally, although there are a number of currency exchanges performed via Ripple—some of which deal with huge amounts—it is hard to tell whether those transactions have been actually concluded since the Ripple system has no way to enforce IOU transactions.

References

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, June 2014. USENIX Association.

[2] The elements project. available from `https://elementsproject.org/`.

[3] Liquid. `https://elementsproject.org/sidechains/liquid/`.

[4] Blockstream. available from `https://www.blockstream.com/`.

[5] Adam Back, G Maxwell, M Corallo, Mark Friedenbach, and L Dashjr. Enabling blockchain innovations with pegged sidechains, 2014. available from http://cryptolibrary.org/bitstream/handle/21/406/2014_Back_Enabling_blockchain_innovations_with_pegged_sidechains.pdf?sequence=1.

[6] Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification.

[7] Ethereum Homestead Release. available from https://www.ethereum.org/.

[8] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[9] Ethereum homestead 0.1 documentation. http://www.ethdocs.org/en/latest/mining.html. Accessed: 2016-05-11.

[10] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[11] Linux Foundation. available from www.hyperledger.com.

[12] Christian Cachin, Simon Schubert, and Marko Vukolic. Non-determinism in byzantine fault-tolerant replication. *CoRR*, abs/1603.07351, 2016.

[13] Litecoin: Open source P2P internet currency. https://litecoin.org/.

[14] Namecoin: A trust anchor for the internet. https://namecoin.info/.

[15] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple protocol consensus algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf, 2014.

[16] Ripple: Opening access to finance. https://ripple.com/.

[17] Ripple. http://en.wikipedia.org/wiki/Ripple_%28payment_protocol%29.

[18] Ripple labs circling 30m$ in funding. http://www.pymnts.com/news/2015/ripple-labs-circling-30m-in-funding/#.VRLnJfnF98F.

[19] Ripple Labs Inc. Ripple charts. https://www.ripplecharts.com.

[20] Coinist Inc. Ripple gateways. https://coinist.co/ripple/gateways.

[21] International Ripple Business Association. Listed businesses. http://www.xrpga.org/listed-businesses.html.

[22] International Ripple Business Association. Ripple gateways. http://www.xrpga.org/gateways.html.

[23] International Ripple Business Association. Ripple market makers. http://www.xrpga.org/market-makers.html.

[24] International Ripple Business Association. Ripple exchangers. http://www.xrpga.org/exchangers.html.

[25] International Ripple Business Association. Ripple merchants. http://www.xrpga.org/merchants.html.

[26] Arpita Ghosh, Mohammad Mahdian, Daniel M. Reeves, David M. Pennock, and Ryan Fugger. Mechanism design on trust networks. In *Proceedings of the 3rd International Conference on Internet and Network Economics*, WINE'07, pages 257–268, Berlin, Heidelberg, 2007. Springer-Verlag.

[27] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks: Enabling trust with privacy in online marketplaces. In *Network and Distributed System Security (NDSS) Symposium*, 2015.

[28] Frederik Armknecht, Ghassan O. Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, pages 163–180, 2015.

[29] US banks announce Ripple protocol integration. http://www.coindesk.com/us-banks-announce-ripple-protocol-integration/.

[30] Ripple Labs Inc. Why is Ripple not vulnerable to Bitcoin's 51% attack? https://wiki.ripple.com/FAQ#Why_is_Ripple_not_vulnerable_to_Bitcoin.27s_51.25_attack.3F.

[31] Vitalik Buterin. Bitcoin network shaken by blockchain fork. https://bitcoinmagazine.com/3668/bitcoin-network-shaken-by-blockchain-fork/.

[32] Kim Joyes. Safety, liveness and fault tolerance - the consensus choices. available from https://www.stellar.org/blog/safety_liveness_and_fault_tolerance_consensus_choice/.

[33] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[34] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917, New York, 2012. ACM.

[35] Arthur Gervais, Ghassan O. Karame, Karl Wust, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. Cryptology ePrint Archive, Report 2016/555, 2016. http://eprint.iacr.org/2016/555.

[36] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[37] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, pages 34–51, 2013.

[38] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, pages 6–24, 2013.

[39] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the Bitcoin transaction graph. *Future Internet*, 5(2):237–250, 2013.

[40] Arthur Gervais, Ghassan O. Karame, Vedran Capkun, and Srdjan Capkun. Is Bitcoin a decentralized currency? *IEEE Security & Privacy*, 12(3):54–60, 2014.

[41] Ripple Labs Inc. Giveaways - XRPtalk. `https://xrptalk.org/forum/105-giveaways/`.

Chapter 10

Concluding Remarks

In this book, we analyzed in detail the security and privacy provisions of Bitcoin and its underlying blockchain. In addition to discussing existing vulnerabilities of Bitcoin and its various related altcoins, we also discussed and proposed a number of effective countermeasures to deter threats and information leakage within the system—some of which have already been incorporated in Bitcoin client releases. Note that proof-of-work (PoW) powered blockchains currently account for more than 90% of the total market capitalization of existing digital currencies. As far as we are aware, this book offers the most comprehensive and detailed analysis of the security and privacy provisions of existing PoW-based blockchains, and of related clones/variants.

Given that Bitcoin emerges as the most successful PoW blockchain instantiation to date, this book extracts essential lessons learned in security and privacy from eight years of research into the system with the aim to motivate the design of secure and privacy-preserving next-generation blockchain technologies. We now summarize the main observations that we made throughout the book.

SUMMARY

For a long time, the notion of blockchain was tightly coupled with the well-known proof-of-work hash-based mechanism of Bitcoin. For most of its lifetime, it was believed that the security of Bitcoin's blockchain relies on the underlying security of the utilized hash function *and* on the assumption of an honest computing majority. Many users believed that as long as no mining pool operator can harness 50% computing power in the network, then Bitcoin was secure; miners would actively

abandon pools to ensure that this threshold was not reached. Recent research has, however, shown that Bitcoin does not properly incentivize miners to abide by the protocol; selfish mining—in which miners selectively release mined blocks in the network—proves to be a profitable strategy for miners to increase their mining advantage in the system. Even worse, the proof-of-work mechanism of Bitcoin is vulnerable to network-layer attacks, allowing resource-constrained adversaries to selectively eclipse Bitcoin nodes from receiving information from the network. When combined with selfish-mining, such attacks are detrimental for the security of the system; as a result, a mining pool that harnesses as little as 32% of the computing power in the network can effectively control the security of the entire system.

Moreover, securing Bitcoin transactions additionally depends on the ability of users to protect their private keys. Namely, since Bitcoin transactions basically consist of transferring the outputs of unspent previous transactions to a new public key, the compromise or loss of a private key means that peers can no longer redeem any transaction sent to the corresponding public key. Bitcoin stores these private keys in a nonprotected user-specific structure, the wallet. There are a number of proposals/start-ups that offer to secure digital wallets on behalf of Bitcoin users; most of these proposals require users to offload trust to a limited number of entities in order to protect their wallets. Other proposals—such as those requiring support from multisig transactions, external cloud storage, and/or trusted computing—reduce the reliance on such trust assumptions but require support from additional hardware/functionality. These observations motivate the need to understand and analyze the security of blockchains using a holistic approach covering the security of cryptographic primitives, network-layer and system-layer attacks, as well as the storage of private keys and secrets prior to any large-scale deployment.

Note that even if there is a lower bound on the fraction of an honest majority to ensure the security of Bitcoin (which remains unknown up to now), the Bitcoin network requires considerable time to reach consensus. Such a consensus is essential to resist double-spending attacks in the network (and other misbehavior). Namely, Bitcoin requires six block confirmations for each transaction in the network—a process which consumes 60 minutes on average. This forces a number of Bitcoin merchants to bypass the network's consensus protocol and accept unconfirmed payments—a move which clearly weakens the security of payments in the system. A number of studies have shown that unconfirmed transactions can be easily double-spent by resource-constrained adversaries without being noticed in the network. Although there were a number of attempts to secure unconfirmed payments in the system (e.g., Bitcoin XT), there is still no silver-bullet solution that can resist network-layer attacks. It was recently shown that network attacks can

easily circumvent most adopted countermeasures in the system. Up to the time of writing, the best countermeasures to prevent attacks on unconfirmed transactions consist of: (1) waiting a considerable amount of time before accepting the payment, or (2) installing several (e.g., five) machines running the Bitcoin client at various locations across the globe and ensuring that these machines are located behind a NAT or a firewall to prevent targeted eclipse attacks. This shows the need for next-generation blockchains to achieve fast consensus by design and to plan for realistic use cases and deployment settings as most users/vendors expect digital currencies to realize secure and fast payments at low costs.

In terms of privacy and anonymity, studies have shown that Bitcoin leaks considerable information about its users, since all transactions (including the timing and amounts exchanged) are public. As we explained in this book, this is a mandatory requirement to ensure the security of transactions within Bitcoin. This information leakage motivated considerable research to enhance the privacy of the system, and a number of proposals for mixing coins, such as Mixcoin and CoinJoin have been proposed. These proposals offer privacy by offloading trust to one (or more) entities/participants in the system—which suggests a clear departure from the decentralized trust model of Bitcoin. To remedy this, a number of cryptographic extensions of Bitcoin, such as ZeroCoin, Extended ZeroCoin, and ZeroCash, propose the reliance on dynamic accumulators and zero-knowledge proofs of knowledge to enhance user privacy in the system. While some of these proposals can achieve unprecedented levels of privacy in Bitcoin by preventing coin expenditure in the network, and hiding transaction amounts (and address balances), they result in an unacceptable performance penalty that effectively hinders their adoption within the Bitcoin system. This demonstrates the need to incorporate privacy-by-design mechanisms in next-generation blockchain technologies. The Bitcoin experience clearly shows that the sole reliance on pseudonyms and network-based protection is not enough to ensure an acceptable level of user privacy.

The lack of privacy offered by the current Bitcoin system can however be seen as an enabler for accountability measures in the system. Incorporating accountability measures in Bitcoin is essential to deterring misbehavior, especially given the lack of workable mechanisms to ban/punish Byzantine nodes. Recall that, at the time of writing, Bitcoin nodes locally ban the IP address of the misbehaving user for 24 hours. Clearly, such an approach is not sufficient to deter misbehavior, since malicious peers can, for example, modify/spoof their IPs or even try to connect to and attack other peers who have not blacklisted their IP address. We argue that if any blockchain technology is to sustain decades of service, then it must incorporate accountability measures in order to ensure that a misbehaving user is indeed

punished. In this respect, one possible solution would be to enforce Bitcoin *address* blacklisting. Here, the idea would be that those Bitcoin addresses that have been found to misbehave (e.g., double-spend) are added to a public blacklist. Ideally, the BTCs of the blacklisted addresses will not be accepted by Bitcoin peers, and will therefore lose their value. Besides the concerns/issues related to the management and maintenance of such lists, this approach is not sufficient—when used alone—to deter misbehavior since misbehaving users can be equipped with many addresses each containing low balances. Therefore, one obvious question that emerges is whether it is possible to *link* different Bitcoin addresses of the same (misbehaving) user (address linkability). Since such linking is possible, misbehaving users could receive harsher punishment for their misbehavior by not being able to spend a large fraction of their funds.

The security and privacy of lightweight clients (operating under the so-called SPV mode) for blockchains is also of outmost importance. For instance, Bitcoin requires peers in the system to verify all broadcasted transactions and blocks. Clearly, this comes at the expense of storage and computational overhead. To remedy that, most users rely on lightweight client implementations that only perform a limited amount of verifications, such as the verification of block difficulty and the presence of a transaction in the Merkle tree, and offload the verification of all transactions and blocks to the full Bitcoin nodes. Lightweight clients need only to receive a subset of network transactions that are relevant to the user's wallet. This, however, allows Bitcoin nodes to learn information about the addresses owned by the client simply by observing the transactions forwarded to the lightweight clients. Studies show that this information leakage cannot be fully deterred by existing solutions, such as the reliance on anonymizing networks, or leverage false positives of Bloom filters. On the other hand, the reliance on bullet-proof solutions such as private set intersection and/or private information retrieval techniques incur considerable computational overhead that cannot be tolerated by most lightweight clients. Given that most blockchain users no longer run full client implementations, these observations motivate the need to design lightweight and efficient SPV client modes that are privacy-preserving.

Finally, one must pay special attention to the practical deployment of blockchain technologies in the real world. For instance, while the original design of Bitcoin aims at a fully decentralized Bitcoin, recent events in Bitcoin are revealing several aspects of centralization within the system. Namely, a large number of centralized services currently support Bitcoin (e.g., Bitcoin banks, Bitcoin mining pools, Bitcoin market exchanges, Bitcoin online wallets) and control a considerable share in the Bitcoin market. For instance, a quick look at the distribution of

computing power in Bitcoin reveals that the power of dedicated miners far exceeds the power that individual users dedicate to mining, allowing a few parties to effectively control the currency; currently the top three (centrally managed) mining pools control more than 50% of the computing power in Bitcoin. Indeed, while mining and block generation in Bitcoin was originally designed to be decentralized, these processes are currently largely centralized. On the other hand, other Bitcoin operations, like protocol updates and incident resolution, are not designed to be decentralized and are controlled by a small number of administrators whose influence does not depend on the computing power that they control but is rather derived from their function within the system. Bitcoin users do not have any direct influence over the appointment of the administrators—which is somewhat ironic since some of the Bitcoin users opt for Bitcoin in the hope of avoiding the centralized control typically exercised over national currencies.

Furthermore, we note that Bitcoin introduces a level of transparency in terms of coin mining and spending since the transaction logs in Bitcoin are public and available for inspection by any interested party. However, it is not clear how any potential disputes would be resolved in Bitcoin since this would then require appropriate regulatory frameworks—a move that clearly goes against the very nature of Bitcoin.

We further observe that the existence of public logs in Bitcoin can have some negative effects on this currency that extend beyond known privacy concerns. Bitcoin users can, for example, decide not to accept coins that appear to have originated from a particular address (i.e., that were mined by the owner of that address). Since the use of any coin (or its fraction) can be traced back to its origin, this decision by the users will practically devalue these coins because other users will become reluctant to accept these coins as payments. These observations motivate the need to learn from the various caveats in the current deployment of Bitcoin and consider decentralization in all deployment aspects of next-generation blockchains.

OUTLOOK

Irrespective of our opinion of Bitcoin and of speculations on Bitcoin's future, we argue that Bitcoin has provided a considerable number of relevant lessons for system designers, researchers, and blockchain enthusiasts.

In terms of outlook, Bitcoin's blockchain fueled innovation, and a number of innovative applications have already been devised by exploiting the secure

and distributed provisions of the underlying blockchain. Prominent applications include secure time stamping, secure commitment schemes, secure multiparty computations, and smart contracts. Note that some of these extensions cannot be deployed without changing the code base of Bitcoin (i.e., via a hard fork). These are referred to as altcoins and require some measures to initiate currency allocation (e.g., via pegged sidechains) and preserve mining power by leveraging the already established Bitcoin community.

Recently, IBM proposed the notion of Device Democracy with the goal to support consensus across a fully meshed network of IoT devices based on the blockchain technology. Other blockchain technologies were also proposed almost independently from Bitcoin. Many of these propose to replace Bitcoins proof-of-work in order to offset its energy waste and scalability limits. For instance, a number of contributions propose the reliance on memory-based consensus protocols or virtual mining such as proof-of-stake.

Other proposals resort to the classic Byzantine fault-tolerant consensus protocols in the hope of increasing the ledger closure efficiency and achieve high transactional throughput. Moreover, in contrast to the unspent transaction output model used in Bitcoin and its altcoins, some blockchains adopt models such as credit networks or account-based models in which transactions directly link to the issuer accounts instead of pointing to the output of previous transactions.

Among these alternative blockchains, Ripple, the current holder of the second largest market capitalization after Bitcoin, maintains a distributed ledger that keeps track of all the exchanged transactions and account states in the system. Ledgers are created every few seconds and contain a list of transactions to which the majority of validating servers has agreed. This is achieved by means of Ripple's proprietary consensus protocol, which is an iterative process and is executed among validating servers. Ripple has its own currency, called XRP; it also accepts credit-based payments (IOU transaction model) if a trust path between the sender and receiver exists. Ripple has been recently criticized for its centralized deployment (as most of the validation nodes are maintained by Ripple Labs); the underlying consensus protocol of Ripple has also received considerable criticism. Stellar shares a similar model as Ripple, but relies on a federated Byzantine agreement protocol in order to resolve the various issues faced by Ripple. Ethereum brings a new dimension to the blockchain, as it expands the standard application of blockchains from the mere public bulletin board approach to a general-purpose peer-to-peer decentralized platform for executing smart contracts. Namely, Ethereum enables any entity to create and deploy novel applications by writing decentralized contracts. The contract itself is a small program that maintains its own key-value store through transaction

calls. Therefore, multiple application services can run on the shared Ethereum platform, whose role is to maintain consensus in the network. The current consensus protocol used in Ethereum is GHOST, which is a variant of proof-of-work. The next-generation Ethereum release, however, is expected to adopt a more efficient security-deposit proof-of-stake consensus protocol. IBM's Open Blockchain (OBC) is mainly inspired by Ethereum and also provides a general-purpose application platform. In addition, OBC introduces membership services to provide authorization for participation and offers confidentiality for transactions.

Nevertheless, in spite of considerable work in this area, there are still many challenges with respect to system scalability and performance that need to be overcome in order to ensure a large-scale adoption of the blockchain paradigm. More specifically, existing blockchain technologies cannot match the performance or transactional volume of conventional payment methods (e.g., Visa can handle tens of thousands of transactions per second). Moreover, experience from Bitcoin has shown that even the modest currently deployed scalability measures often come at odds with the security of the system. It remains still unclear how to devise blockchain platforms that can effectively scale to a large number of participants without compromising the security and privacy provisions of the system. We only hope that the findings, observations, and lessons contained in this book can solicit further research in this area.

Index

Account state, 182
accountability, 15, 91, 207
accumulator, 35
acquirer, 11
activity unlinkability, 87, 101
addr, 50
addr message, 50
addr messages, 33, 62
address unlinkability game, 88
alert messages, 52
altcoins, 163
anonymity, 101
anonymizing networks, 130
anonymous tax reporting, 26
attack, 64, 73, 79
attacks, 61, 67, 69
authenticated storage, 169
authentication, 15
authorization, 15
availability, 14, 15

balance, 15, 116
bandwidth optimization, 53
BIP, 156
Bitcoin address, 38
Bitcoin addresses, 33
Bitcoin blockchain, 34, 43
Bitcoin blocks, 43
Bitcoin exchanges, 146
Bitcoin Improvement Proposals, 156
Bitcoin P2P, 49
Bitcoin pooled mining, 152
Bitcoin transactions, 33
Bitcoin wallets, 146
BitPay, 145
Bits of Proof, 166
BitStamp, 146
block generation, 69
blockchain, 163
Blockstream, 181
Bloom filter, 128
BTC, 144
BTProof, 169

cash-like payments, 12
centralized payments, 19
Chaincode, 185
chaincodes, 190
change outputs, 42
check-like payments, 12
coin tainting, 91
Coinbase, 146
coinbase transaction, 45
Coinify, 145
coinjoin, 99
CoinKite, 145
collision resistance, 35
commitment schemes, 101
completeness, 105
confidential transactions, 181
confidentiality, 14
confirmations, 47
contract accounts, 182
countermeasures, 63, 64, 74, 79
credit card payments, 23
credit networks, 210
crime, 174

criminal smart contracts, 175
cryptographic accumulators, 103
cryptographic has functions, 35

DAP, 114
deanonymization, 94
decentralized anonymous payment, 115
decentralized identity management, 171
decentralized mining pools, 153
decentralized payments, 19
decentralized storage, 166
dedicated relay networks, 52
default number of connections, 50
denial-of-service, 37
deposit, 12, 172
digital assets, 165
digital cash, 21
dispute mediation, 173
DNS seeds, 49
Dogecoin, 164
double geometric method, 152
double-spending, 60, 62, 64, 69, 73, 79
double-spending attacks, 18

ECDSA, 33, 35, 36
Eclipse attacks, 61
electronic cash, 21
Elements projects, 181
elliptic curve cryptography, 36
Elliptic Curve Digital Signature Algorithm, 36
enrollment identity, 187
entry nodes, 50, 94
EOA, 182
equalized shared maximum pay per share, 152
escrow, 18
Ether, 182
Ethereum, 181, 210
Ethereum transactions, 182
Extended ZeroCoin, 100, 108
EZC, 108

fabric, 192
fairness, 15
false positive rate, 129
false positives, 129
fast payments, 164
fees, 34
figures, 37, 48, 60, 61, 65, 70, 71, 128, 129, 167, 173, 180
Finney attack, 73
fork, 44
forks, 34, 79
full node, 49

gas, 182
genesis block, 51
getaddr message, 51
getdata request, 63
GHOST, 184

hardware wallets, 147
hash functions, 35
headers first synchronization, 51
heading
  chapter, 11, 59, 85, 125, 163, 179, 205
  overview, xiii, 1, 33, 143, 213
  section, 5, 11, 13, 17, 29, 33, 35, 36, 47, 53, 59, 69, 79, 86, 97, 125, 128, 148, 157, 163, 166, 180, 181, 185, 193
  subsection, 5–9, 14–17, 20, 26, 35–38, 43, 48, 53, 55, 56, 60, 61, 63, 65, 69, 74, 79, 80, 91, 93, 98–100, 107, 125, 127–129, 139, 144, 146, 148–151, 154, 155, 164–166, 169, 171, 172, 182–184, 186, 190, 192, 194–196
  subsubsection, 17–20, 23, 25–28, 38, 41–44, 48, 49, 51, 52, 62, 66, 67, 73, 87, 89, 90, 94, 101, 105, 108, 114, 156, 169, 172–174, 189, 191, 192, 195, 197–199
HolyTransactions, 148
Hyperledger, 165, 185, 192

IBM Micropayments, 28
indistinguishability, 116
instant conversion, 144
integrity, 14
interactive payments, 12
internal reputation system, 56
inv message, 53, 63
inv messages, 64

issuer, 11

lightweight clients, 49, 125
linkability, 87
Liquid, 181
listening period, 74
Litecoin, 164
Local Bitcoins, 146
longest chain, 66
loss, 148

M-Pesa, 28
mediator-based payments, 12
mediator-based systems, 18
merged mining, 181
Merkle root, 36
Merkle tree, 35, 126
micropayments, 19, 25
miners, 34, 44, 48
mining, 43
mining pool, 48, 151
minipay, 25, 26
mixers, 85, 97
mixing services, 97
multi-input transactions, 89
multi-sig transactions, 40
multicloud storage, 150
multicurrency wallets, 147
multiple Bloom filters, 137
multipool, 152
multisig, 147, 149, 172

Namecoin, 165
NAT, 94
new tables, 62
nLockTime, 43
nLocktime, 172
node restart, 63
noninteractive payments, 12
nonmalleability, 116
nonoutsourceable proof-of-work, 154
nonrepudiation, 15

OBC, 211
observers, 74
off-line payments, 17
one-wayness, 35
OneName, 171
online payments, 17
online wallets, 147
open blockchain, 185
orphan, 45
orphan blocks, 44

P2P, 33
P2PKH transaction, 39
P2SH transaction, 39
pay on target, 152
pay per last N shares, 152
pay per share, 151
payee, 11
payer, 11
payment processor, 144
payment systems, 11
PayPal, 12
paypal, 27
Paystand Bitcoin Merchants, 145
PBFT, 189
Pedersen commitments, 102
pegged sidechain, 180
penalty, 56
Peppercoin, 28
perfect zero knowledge, 105
Permacoin, 169
person to person payment, 144
point of sale solution, 145
pool hopping, 152
POR, 169
PoW, 34, 46
privacy, 13, 15, 16, 85
privacy quantification, 132
privacy-preserving payments, 17, 85
probability, 68
proof of existence, 169
proof of knowledge, 105
proof of membership, 126
proof of stake, 184, 210
Proof of work, 34, 43
proofs of knowledge, 102
proportional reward, 152
pseudonyms, 33
public randomness beacon, 171

rational adversary, 16
recent shared maximum pay per share, 152
redeemScript, 40
request management system, 63
resistance to impersonation attacks, 15
responsible nodes, 50
Revel Systems, 145
Ripple, 193, 194, 210
Ripple wallet, 148
Ripple's consensus, 195
Ripple's ledger, 195
risks, 93

Satoshi coin, 38
SatoshiDICE, 155
SCORE scheme, 152
script execution, 41
Scripts, 37
scrypt, 164
secp256k1 curve, 36
security, 13, 14, 125, 127
Segregated witnesses, 181
selfish mining, 62, 66
shadow address, 42, 87, 90
shared maximum pay per share, 152
shifted geometric distribution, 69
Sidechains, 180
signature of knowledge, 102
simple payment verification, 125
smart contracts, 172, 181
SNARK, 104
SPV, 49, 125, 147
SPV mining, 127
SPV proof, 180
Stellar, 210
succinctness, 105
supported transaction types, 38
synchronous, 61

tables, 45, 46, 73, 75–77, 137, 151
tamper-resistant hardware, 20
TigerDirect, 143
time-dependent source of randomness, 171
time-out, 64
time-outs, 55
Tor exit node, 95
Tor networks, 95
transaction, 38
transaction anonymity, 16
transaction confidentiality, 191
transaction confirmation, 67
transaction output, 34
transaction unlikability, 16
transaction verification, 65
transactions, 59
trickle node, 51
trickling, 51
tried tables, 62
trusted computing, 150
tumblers, 98
two Bloom filters, 134

uncles, 183
UNL, 195
UTXO, 38

verack message, 51
version messages, 51

wallets, 148

XBTerminal, 145
XRP, 193

zero knowledge proofs of knowledge, 102
zero-confirmation transactions, 69
zero-knowledge systems, 27
ZeroCash, 100, 114
ZeroCoin, 100, 105
ZK-SNARKs, 104, 105, 114
zk-SNARKS, 118