PhD-FSTM-2020-45
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 17/09/2020 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Sergei TIKHOMIROV
Born on 29 May 1991 in Moscow (USSR)

SECURITY AND PRIVACY OF BLOCKCHAIN PROTOCOLS AND APPLICATIONS

Dissertation defence committee

"Imagine there was a base metal as scarce as gold but with the following properties:

- *boring grey in colour*
- *not a good conductor of electricity*
- *not particularly strong, but not ductile or easily malleable either*
- *not useful for any practical or ornamental purpose*

and one special, magical property: can be transported over a communications channel."

Satoshi Nakamoto

UNIVERSITY OF LUXEMBOURG

Abstract

Faculty of Science, Technology and Medicine
Department of Computer Science

Doctor of Philosophy

Security and Privacy of Blockchain Protocols and Applications

by Sergei TIKHOMIROV

Bitcoin is the first digital currency without a trusted third party. This revolutionary protocol allows mutually distrusting participants to agree on a single common history of transactions. Bitcoin nodes pack transactions into blocks and link those in a chain (the *blockchain*). Hash-based *proof-of-work* ensures that the blockchain is computationally infeasible to modify.

Bitcoin has spawned a new area of research at the intersection of computer science and economics. Multiple alternative *cryptocurrencies* and blockchain projects aim to address Bitcoin's limitations. This thesis explores the security and privacy of blockchain systems.

In Part I, we study the privacy of Bitcoin and the major privacy-focused cryptocurrencies. In Chapter 2, we explore the peer-to-peer (P2P) protocols underpinning cryptocurrencies. In Chapter 3, we show how a network adversary can link transactions issued by the same node. We test the efficiency of this novel attack in real networks, successfully linking our own transactions. Chapter 4 studies the privacy characteristics of mobile cryptocurrency wallets. We discover that most wallets do not follow the best practices aimed at protecting users' privacy.

Part II is dedicated to the Lightning Network (LN). Bitcoin's architecture emphasizes security but severely limits transaction throughput. The LN is a prominent Bitcoin-based protocol that aims to alleviate this issue. It performs low-latency transactions *off-chain* but leverages Bitcoin's security guarantees for dispute resolution. We introduce the LN and outline the history of off-chain protocols in Chapter 5. Then, in Chapter 6, we introduce a probing attack that allows an adversary to discover user balances in the LN. Chapter 7 estimates the likelihood of various privacy attacks on the LN. In Chapter 8, we describe a limitation on the number of concurrent LN payments and quantify its effects on transaction throughput.

Part III explores the security and privacy of Ethereum smart contracts. Bitcoin's language for defining spending conditions is intentionally restricted. Ethereum is a blockchain network allowing for more programmability. Ethereum users can write programs in a Turing-complete high-level language called Solidity. These programs, called *smart contracts*, are stored on-chain along with their state. Chapter 9 outlines the history of blockchain-based programming. Chapter 10 describes Findel – a Solidity-based declarative domain-specific language for financial contracts. In Chapter 11, we classify the vulnerabilities in real-world Ethereum contracts. We then present SmartCheck – a static analysis tool for bug detection in Solidity programs. Finally, Chapter 12 introduces an Ethereum-based cryptographic protocol for privacy-preserving regulation compliance.

Acknowledgements

This work would not be possible without the help and support of many people. This is an incomplete list of those to whom I would like to express my gratitude.

First of all, I am deeply grateful to my advisor, Prof. Alex Biryukov, for the opportunity to pursue my research interests and the invaluable guidance throughout this journey. I appreciate Prof. Volker Müller, A-Prof. Andrew Miller, Prof. Matteo Maffei, Dr. Patrick McCorry, and A-Prof. Arthur Gervais agreeing to serve as jury members. I thank my co-authors: Dmitry Khovratovich, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, Yaroslav Alexandrov, Pedro Moreno-Sanchez, Matteo Maffei, René Pickhardt, and Mariusz Nowostawski. I have learned a lot during our fruitful collaborations. A special thanks goes to Pedro Moreno-Sanchez and Matteo Maffei for inviting me to spend three months on a research visit at TU Wien in the beautiful city of Vienna.

I thank my colleagues for the insightful and fun conversations: Aleksei Udovenko, Brian Shaft, Christof Beierle, Dag Arne Osvik, Daniel Dinu, Daniel Feher, Dmitry Khovratovich, Giuseppe Vitto, Johann Großschädl, Luan Cardoso dos Santos, Léo Perrin, Qingju Wang, Ritam Bhaumik, Shange Fu, Vesselin Velichkov, and Yann Le Corre.

The country of Luxembourg and the University of Luxembourg generously provided me with excellent research conditions. I specifically thank Fabienne Schmitz, Ida Ienna, and Catherine Violet for their assistance in administrative matters.

A huge thanks goes to my co-hosts for the Basic Block Radio podcast Ivan Ivanitskiy, Sergei Pavlin, and Alexander Seleznev, and to all our guests and listeners, for helping spread the word about decentralized technologies for the Russian-speaking audience.

My deepest gratitude goes to my family for their endless love and support. I want to specifically thank my parents for encouraging me to pursue education abroad.

Finally, I thank Satoshi Nakamoto. Your ingenious invention continues to amaze and inspire me. I admire your wisdom and humility. It is an honor to be a part of the movement you started.

Contents

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**
- 1.1 Foreword . . . 1
- 1.2 Historical overview . . . 2
  - 1.2.1 Evolution of the Internet . . . 2
  - 1.2.2 Evolution of money . . . 4
- 1.3 Bitcoin . . . 7
  - 1.3.1 Bitcoin architecture . . . 8
  - 1.3.2 Is proof-of-work wasteful? . . . 9
  - 1.3.3 Professionalization of mining . . . 10
- 1.4 Challenges for cryptocurrencies . . . 12
  - 1.4.1 Expressiveness . . . 12
  - 1.4.2 Scalability . . . 13
  - 1.4.3 Privacy . . . 13
- 1.5 Our contributions . . . 15

I Network privacy in Bitcoin and privacy-focused cryptocurrencies **17**

2 P2P protocols in cryptocurrencies **19**
- 2.1 Design goals for a cryptocurrency P2P network . . . 19
- 2.2 P2P protocols in cryptocurrencies . . . 20
  - 2.2.1 Bitcoin P2P protocol . . . 20
  - 2.2.2 Dandelion . . . 21
- 2.3 Taxonomy of nodes . . . 22
- 2.4 Network-level privacy in cryptocurrencies . . . 23
  - 2.4.1 Network-based transaction deanonymization . . . 23

3 Deanonymization of transactions with network analysis **27**
- 3.1 Transaction clustering with timing analysis . . . 27
  - 3.1.1 Weight functions and clustering . . . 28
  - 3.1.2 Measuring clustering quality . . . 30
  - 3.1.3 Measuring the degree of deanonymization . . . 30
- 3.2 Implementation details . . . 31
- 3.3 Experimental evaluation . . . 32
  - 3.3.1 Results for desktop wallets . . . 33
    - 3.3.1.1 Bitcoin testnet . . . 34
    - 3.3.1.2 Bitcoin mainnet . . . 35
    - 3.3.1.3 Zcash . . . 36
    - 3.3.1.4 Dash . . . 37

3.3.1.5 Monero 37
3.3.2 Results for mobile wallets 39
3.4 Attack cost estimation 42
3.5 Discussion and countermeasures 42
3.6 Conclusion 44

4 Privacy of cryptocurrency wallets 45
4.1 Minimal privacy criteria 45
4.2 Analysis of selected wallets 46
4.2.1 Manual inspection 47
4.2.2 Static analysis 50
4.3 Conclusion 52

II Privacy of the Lightning Network 53

5 Introduction to Lightning Network 55
5.1 Evolution of payment channels in Bitcoin 55
5.1.1 Transaction replacement with sequence numbers 55
5.1.2 Unidirectional channels 56
5.1.3 Replace-by-timelock and Duplex channels 56
5.1.4 Poon-Dryja channels (Lightning) 57
5.2 Lightning Network architecture 58
5.2.1 Nodes 58
5.2.2 Channels 58
5.2.3 P2P network and path-finding 61
5.2.4 The future of Lightning 61
5.3 Research directions in payment channel networks 62

6 Probing Lightning channel balances 63
6.1 Probing algorithm 63
6.1.1 Overview 63
6.1.2 Assumptions 64
6.1.3 Selecting channels for probing 65
6.1.4 Probing 66
6.2 Experimental setup 69
6.3 Results 70
6.3.1 Probing times 70
6.3.2 Probing coefficients 71
6.3.3 Distribution of channels in routes 72
6.3.4 How balanced are the channels? 72
6.4 Estimating the attack cost 73
6.5 Limitations 74
6.6 Countermeasures 75
6.7 Conclusion 77

7 Quantitative analysis of Lightning network privacy 79
7.1 Datasets 79
7.2 Security and privacy attacks: background 80
7.3 Methodology 81
7.4 Results and discussion 83
7.5 Countermeasures 85

7.6 Conclusion 85

8 Throughput limitation of the Lightning network 87
8.1 Background 87
8.2 The HTLC limit effect on LN scalability 88
8.3 Depleting the Lightning Network 90
8.4 Discussion 91
8.5 Countermeasures 92
8.6 Conclusion 92

III Security of Ethereum smart contracts 93

9 Introduction to smart contracts 95
9.1 History of smart contracts 95
9.1.1 Early ideas 95
9.1.2 Smart contracts in Bitcoin 95
9.1.3 Smart contracts in Ethereum 96
9.2 Ethereum architecture 96
9.2.1 Accounts 96
9.2.2 Transactions 97
9.2.3 Mining and coin issuance 98
9.2.4 Contracts 98
9.2.5 Applications 99
9.3 Challenges for smart contracts 99

10 Findel: Secure derivative contracts for Ethereum 101
10.1 Findel contracts 101
10.1.1 Execution model 102
10.1.2 Example 103
10.2 Implementation 104
10.2.1 Users and balances 104
10.2.2 Ownership transfer 105
10.2.3 Data sources and gateways 105
10.2.4 Execution 106
10.3 Possible improvements 106
10.3.1 Enforcement 106
10.3.2 Defaulting on debt 107
10.3.3 Modeling balances with Tokens 107
10.3.4 Multi-party contracts 108
10.3.5 Local client 108
10.4 Platform limitations 108
10.5 Gas costs 109
10.5.1 Setup and helper functions 109
10.5.2 Managing common derivatives 110
10.6 Conclusion 110
10.7 Examples 111

11 SmartCheck: Static analysis of Ethereum smart contracts **113**
11.1 Classification of issues in Solidity code 113
11.1.1 Security issues 113
11.1.1.1 Balance equality 113
11.1.1.2 Unchecked external call 115
11.1.1.3 DoS by external contract 115
11.1.1.4 `send` instead of `transfer` 115
11.1.1.5 Re-entrancy 115
11.1.1.6 Malicious libraries 116
11.1.1.7 Using `tx.origin` 116
11.1.1.8 Transfer forwards all gas 117
11.1.2 Functional issues 117
11.1.2.1 Integer division 117
11.1.2.2 Locked money 117
11.1.2.3 Unchecked math 117
11.1.2.4 Timestamp dependence 118
11.1.2.5 Unsafe type inference 118
11.1.3 Operational issues 118
11.1.3.1 Byte array 118
11.1.3.2 Costly loop 118
11.1.4 Developmental issues 119
11.1.4.1 Token API violation 119
11.1.4.2 Compiler version not fixed 119
11.1.4.3 `private` modifier 119
11.1.4.4 Redundant fallback function 119
11.1.4.5 Style guide violation 119
11.1.4.6 Implicit visibility level 120
11.2 SmartCheck architecture 120
11.3 Experimental results 122
11.3.1 Definitions 122
11.3.2 Case studies 122
11.3.3 Testing on a massive sample 124
11.4 Conclusion 126

12 Privacy-preserving KYC on Ethereum **127**
12.1 Identity 127
12.1.1 Centralized identity 128
12.1.2 Decentralized identity and open blockchains 128
12.1.3 Financial and privacy regulation in the EU 129
12.2 KYCE: a decentralized KYC-compliant exchange 129
12.2.1 Definitions and security properties 129
12.2.2 Tokens and exchanges 130
12.2.3 Privacy-preserving KYC 130
12.2.4 Use cases 132
12.3 Implementation details 134
12.4 Related work 134
12.5 Conclusion 135

Bibliography **137**
List of Publications 165

List of Figures

3.1 Weight functions for three timestamp vectors. . . . 29
3.2 Transaction clustering for Bitcoin testnet (listener in California). . . . 34
3.3 Transaction clustering for Bitcoin testnet (listener in Tokyo). . . . 34
3.4 Transaction clustering for Bitcoin testnet (listener in Frankfurt). . . . 35
3.5 Transaction clustering for Bitcoin testnet (combined listeners). . . . 35
3.6 Transaction clustering for Bitcoin mainnet. . . . 36
3.7 Transaction clustering for Zcash. . . . 36
3.8 Free connection slots for Bitcoin testnet. . . . 37
3.9 Free connection slots for Zcash mainnet. . . . 37
3.10 Transaction clustering for Dash (messages and transactions). . . . 38
3.11 Transaction clustering for Dash (transactions only). . . . 38
3.12 Transaction clustering for Monero. . . . 38
3.13 Transaction clustering for mobile wallets. . . . 40

5.1 An HTLC-based payment in the Lightning Network. . . . 59

6.1 The distribution of probes (onions) by response time. . . . 70
6.2 Distribution of channels by total probing time. . . . 71
6.3 Distribution of channels by the obtained information coefficient. . . . 72
6.4 Distribution of relative frequencies of channels in routes. . . . 73
6.5 Distribution of balance coefficients. . . . 74

7.1 Node degree distribution. . . . 80
7.2 Channel capacity distribution. . . . 80
7.3 An illustrative example of value privacy (top), relationship anonymity (middle), and the wormhole attack (bottom). . . . 81
7.4 The share of experiment runs where paths with sufficient capacity exist between sender and receiver. . . . 82
7.5 Share of prone paths for each parameter combination. . . . 84

8.1 Ratio between the current limit on concurrent channel updates and the theoretically possible capacity-based limit. . . . 88
8.2 Share of affected channels for different payment amounts. . . . 89
8.3 Historic share of HTLC-limited channels. . . . 89
8.4 Historic borderline amounts. . . . 89
8.5 Effectiveness of targeting highest-capacity channels. . . . 90
8.6 Number of channels to cut to isolate the largest communities. . . . 91

11.1 Parse tree for the Balance equality code example. . . . 121
11.2 Distribution of non-zero contract balances (ether). . . . 124
11.3 Findings on the big dataset (excluding Implicit visibility level). . . . 125

12.1 KYC-compliant exchange. . . . 133
12.2 KYC-compliant token. . . . 133

List of Tables

1.1 PoW hash functions in selected cryptocurrencies. 11

3.1 Experimental results of transaction clustering for Bitcoin testnet and Zcash. 33

4.1 Minimal privacy criteria for selected wallets. 46
4.2 Alternative installation methods of selected wallets. 47
4.3 Permissions of selected wallets. 48
4.4 Privacy policies of selected wallets: information that the developers may obtain. 49
4.5 Networking characteristics of selected wallets. 50
4.6 Static analysis of selected wallets. 52

10.1 Findel contract primitives. 103
10.2 Examples of custom Findel contracts. 103
10.3 Cost of setup and helper functions (gas units). 109
10.4 Cost of handling derivatives in Findel (gas units). 110

11.1 Code issues detected by SmartCheck. Gray background – false positives possible. 114
11.2 Testing results on three selected projects. 123
11.3 Code issues detected on the big dataset. 126

List of Abbreviations

AMHL	Anonymous multi-hop lock
AML	Anti money laundering
ANTLR	Another tool for language recognition
API	Application programming interface
APK	Android package
ARPANET	Advanced research projects agency network
ASIC	Application-specific integrated circuit
AST	Abstract syntax tree
BFT	Byzantine fault tolerant
BIP	Bitcoin improvement proposal
BOLT	Basics of Lightning technology
BTC	Bitcoin (a currency unit)
CA	Certificate authority
CEI	Checks-effects-interactions
CLTV	Checklocktimeverify
CPU	Central processing unit
CTF	Counter terrorist financing
DAO	Decentralized autonomous organization
DHT	Distributed hash table
DMC	Duplex micropayment channels
DNS	Domain name system
DoS	Denial of service
DSL	Domain-specific language
ECDSA	Elliptic curve digital signature algorithm
EU	European Union
EVM	Ethereum virtual machine
FDR	False discovery rate
FNR	False negative rate
FN	False negative
FPGA	Field-programmable gate array
FP	False positive
FTP	File transfer protocol
GDPR	General data protection regulation
GHOST	Greedy heaviest-observed sub-tree
GPU	Graphics processing unit
GUI	Graphical user interface
HTLC	Hash time-locked contract
HTTPS	Hypertext transfer protocol secure
HTTP	Hypertext transfer protocol
IBD	Initial block download
ICO	Initial coin offering
ID	Identifier
IP	Internet protocol

IR	Intermediate representation
ISP	Internet service provider
JIT	Just in time
JSON	JavaScript object notation
KYC	Know your customer
LN	Lightning network
MPP	Multi-part payment
NAT	Network address translation
NFC	Near-field communication
P2P	Peer-to-peer
PBFT	Practical Byzantine fault tolerance
PKI	Public key infrastructure
PoS	Proof of stake
PoW	Proof of work
PSD	Payment service directive
QR	Quick response
RAM	Random-access memory
RPC	Remote procedure call
SEC	Securities and exchange commission
SHA	Secure hash algorithm
SMS	Short message service
SPV	Simplified payment verification
TCP	Transmission control protocol
TLS	Transport layer security
TN	True negative
TP	True positive
URL	Uniform resource locator
USD	United States dollar
US	United States
UTXO	Unspent transaction output
XML	Extensible markup language
XSS	Cross-site scripting
zk-SNARK	Zero-knowledge succinct non-interactive argument of knowledge

Chapter 1

Introduction

Governments are good at cutting off the heads of a [*sic*] centrally controlled networks like Napster, but pure P2P networks like Gnutella and Tor seem to be holding their own.

Satoshi Nakamoto [273]

If you're not breaking the rules, you're doing it wrong.

Simon Morris [268]

1.1 Foreword

Bitcoin has emerged at the intersection of two secular trends. First, computer networks have enabled nearly-instant global connectivity. The proliferation of the Internet has had a massive economic and societal impact. Second, the world has abandoned the gold standard in favor of *fiat* money. Central banks can arbitrarily inflate the supply of national currencies. The global financial system has become even more interconnected.

Modern finance relies on trust. Trusting one's counterparty differs from trusting the financial system. People are free to choose whom they do business with, and trustworthy organizations prosper. High counterparty trust lowers transaction costs and leads to prosperity. The financial system, on the contrary, demands the trust of all economic actors. This trust concentration puts much power in the administrators' hands. History has shown that they do not always use it responsibly. Governments routinely abuse their influence over money, printing their way out of deficits at savers' expense.

Cryptographers have been working on digital payment systems since the 1980s. However, completely removing a trusted administrator has long seemed unsolvable. The critical challenge is modeling *scarcity*. Money must be costly to produce, but copying digital data is cheap. What prevents a malicious user from spending multiple copies of their digital coin? The traditional solution implies trusting a bank that keeps track of all coins and prevents fraud. Is it possible to achieve the same result without trusting any single entity?

Bitcoin provides an alternative. Announced in 2008 and launched in 2009, it is the first system of its kind. Based on decades of research in cryptography and distributed systems, it models scarcity without a trusted party. Bitcoin's security relies on a combination of cryptographic algorithms and economic incentives. We describe the architecture of Bitcoin in more detail in Section 1.3.

Bitcoin has spawned a new field of study at the intersection of computer science and economics. Thousands of alternative cryptocurrencies are exploring various points in the design space. This thesis attempts to tackle some of the problems in the field, focusing on privacy and security.

In the remainder of this Chapter, we give a more elaborate introduction to cryptocurrencies. First, we outline the relevant historical context and describe the architecture of Bitcoin. Then, we list the challenges it faces and the potential ways to address them. Finally, we outline our original contributions.

1.2 Historical overview

We now provide a historical overview of the two key areas relevant to the development of cryptocurrencies: the Internet and money.

1.2.1 Evolution of the Internet

Information networks developed rapidly in the second half of the XX century. Scientists created the first computer networks in 1960s. ARPANET, the precursor of the Internet, launched in 1969. In 1981, it connected more than 200 computers in US-based research centers.

Early communication networks used circuit switching. Each pair of hosts used a dedicated connection throughout the session. Internet protocols use another approach – packet switching. The sender splits the message into pieces (packets) that travel through the network independently. The receiver reconstructs the message from the packets. The sender re-transmits lost or malformed packets. Packet switching is less reliable but simpler than circuit switching. It has proved indispensable in connecting heterogeneous networks into the global Internet.

Early computer networks lacked security. Protocol designers prioritized simplicity over data confidentiality and integrity. Early Internet users, mostly academics, were not inclined to harm others. Perhaps more importantly, no cryptographic algorithms were suited for the Internet.

Cryptography studies methods to control information flows. For hundreds of years, its primary task was hiding information using *symmetric* encryption. In a classical setting, Alice wants to send a confidential message (the *plaintext*) to Bob. She *encrypts* the plaintext using a secret *key* and transfers the resulting *ciphertext* to Bob. Bob uses the same key to *decrypt* the ciphertext into the original plaintext. An adversary may intercept the ciphertext but cannot decrypt it without the key.

Note that the parties use the same key. Key establishment is a weak spot of symmetric encryption. An adversary who intercepts the key can decrypt all messages. Before the 1970s, two parties could only establish a shared secret by meeting in person or using a physically protected *secure channel*. Both options are expensive and scale poorly.

Moreover, authentication also depended on a shared key. It was impossible to convince the counterparty that the message was authentic without giving them the power to sign messages themselves, which is unacceptable for many Internet use cases. For instance, a company would have to share the signing key with all readers to convince them of the authenticity of a press release. It immediately follows that all subsequent messages signed by this key cannot be trusted.

Whitfield Diffie and Martin Hellman solved both problems. In their breakthrough 1976 paper "New directions in cryptography" [121], they proposed two novel algorithms. First, they introduced a *key establishment* protocol over an insecure channel. This algorithm allows two parties to securely generate a shared secret even if an eavesdropper intercepts all their messages. Second, they described the first *digital signature* algorithm. A digital signature allows a sender to prove the authenticity of their messages without sharing the signing key. A new field of cryptography – *asymmetric* cryptography – was born.

Asymmetric cryptography enabled the widespread deployment and commercialization of the Internet. Users could now establish spontaneous secure connections over insecure channels. Businesses started adopting the Internet in the 1980s. This process accelerated in the early 1990s with the invention of the World Wide Web and web browsers with a graphical user interface. Entrepreneurs started the first Internet companies. Many startups proved nonviable and went bankrupt in the Dot-com crash of 2000. Their early enthusiasm, even if unjustified, attracted talent and capital into the nascent industry. The first two decades of the XXI century saw a rapid expansion of Internet businesses. A new generation of companies built and scaled novel digital services to billions of users.

Modern Internet businesses heavily rely on *networks effects*. Any network is valuable because it allows its members to communicate. Therefore, a new user is more likely to join the social network most of their friends already use. Network effects allow established companies to diminish competition. Internet giants gather vast amounts of user data across all their services. Large-scale data analysis helps them fine-tune their products to attract and retain users more efficiently. This self-reinforcing loop favors the incumbents and concentrates market power.

As a result, the Internet in 2020 is highly concentrated. The five US-based Internet giants – Google, Apple, Facebook, Amazon, and Microsoft (abbreviated as *GAFAM*) – account for 17.5% of the S&P market index [228]. GAFAM plus the China-based Alibaba and Tencent are the most valuable companies in the world by market capitalization. As digital communication now influences most areas of life, Internet giants play an even larger economic and political role.

File-sharing networks

Peer-to-peer file-sharing, which became widespread in the 1990s, foreshadowed cryptocurrencies. At that time, the Internet was gaining adoption in the developed world. Increased bandwidth allowed users to distribute large files over the Internet. P2P file-sharing networks were first to satisfy the demand for fast and convenient content sharing.[1]

File-sharing networks and cryptocurrencies share two crucial attributes. First, networks of both types are driving against the trend towards the centralization of the Internet. Instead of relying on a centralized service provider, they pool resources from users' computers. Second, they demonstrate that given sufficient economic incentives, a decentralized network is impossible to shut down. We explain the differences and similarities between file-sharing and cryptocurrency protocols in Chapter 2.

Napster was the first popular file-sharing network. It launched in 1999. The protocol was only partially decentralized. Users hosted the files, and a central server coordinated the exchange. Napster quickly attracted millions of users. Widespread

[1]Client-server file-sharing predates P2P file-sharing by at least two decades: the File Transfer Protocol (FTP) was introduced in 1971.

sharing of copyrighted content drew the attention of law enforcement. The administrators shut down the service in 2001 to comply with a court order. Napster shows how centralization harms resilience. Without central coordination, Napster users could not locate the files. The existence of the central server made it *possible* to shut the network down.

Gnutella, introduced in 2000, took another approach to content addressing.[2] In Gnutella, users forward queries to all their neighbors. Each neighbor either replies with the requested content or forwards the query further. This "flooding" approach has no single point of failure but is inefficient.

Distributed hash tables (DHT) offered a compromise by storing the content *index* in a distributed manner. This approach proved to be resilient and efficient. A DHT randomly distributes content among nodes. A searching node forwards the query to the node "closest" to the required file. DHT allows for efficient querying and minimal network restructuring when nodes leave or join. Kademlia [249] is a popular DHT implementation.

BitTorrent [304], launched in 2001, is arguably the most successful file-sharing protocol. It strikes a balance between efficiency and resilience by allowing users to locate files via either specialized websites (*torrent trackers*) or a Kademlia-like DHT.

File-sharing networks demonstrate the importance of economic incentives – the central tenet of cryptocurrencies. Users of file-sharing networks download content from other users' computers. A protocol without an identity system cannot force users to upload content. What motivates uploaders to provide the files for free?

BitTorrent implements measures against free-riding. First, downloaders receive file chunks from peers who do not have the full file. In turn, they upload parts of the file back while waiting for their download to complete. On top of the protocol-based measures, some torrent trackers also account for how much their users upload and download. The combination of these measures makes BitTorrent sufficiently reliable but not too difficult to use. The file-sharing ecosystem has attracted both altruistic [318] and profit-driven [333] content distributors.

In 2010s, file-sharing declined in popularity. However, it applied intense competitive pressure on the entertainment industry. Streaming services emerged, offering unlimited access to content for a fixed monthly price.[3]

File-sharing demonstrated the resiliency of Internet protocols. Despite copyright infringement lawsuits against torrent trackers and their users, law enforcement could not fully shut down file-sharing networks. One may argue that this is impossible in principle. As long as at least two computers are willing to communicate according to the protocol rules, the network lives on.

Resilience is crucial for cryptocurrencies. As file-sharing networks opposed a powerful entertainment industry oligopoly,[4] cryptocurrencies compete with central banks. If cryptocurrencies live up to their promise, attempts to shut them down are inevitable.

1.2.2 Evolution of money

Money is a form of language to convey value – a particular type of information. For example, it conveys that the payer performed some valuable work in the past and

[2]See an overview and comparison of Napster and Gnutella in [337].

[3]BitTorrent usage rose in the late 2010s because of the fragmentation of content among streaming services [54].

[4]For example, three major corporations dominate the music industry: Universal Music Group, Sony Music Entertainment, and Warner Music Group.

wishes to receive something in return now. Money separates the labor from enjoying its fruit. Nothing is a universal store of value, because value is subjective. A future payee may reject payment for myriads of unpredictable reasons.

Throughout history, people used various types of money. Some goods make better money than others and do so on longer time frames. Gold is arguably the longest widely recognized money. It possesses the essential properties of money: recognizability, divisibility, portability, durability, fungibility, and scarcity.

Gold is burdensome to handle directly. The growing speed of commerce demanded easier methods of payment, such as *representative money* (gold certificates) and, eventually, *fiat money*, disconnected from physical commodities.

Fiat money is the basis of the modern financial system. In 1971, the US stopped converting dollars to gold, ending the Bretton Woods global monetary system. The currency exchange rates are now determined by supply and demand. Governments and central banks strongly influence the market. They change interest rates, perform market interventions, and enforce capital controls.

This transition has deprived money of its fundamental property – trustlessness. Gold is a *bearer asset*. It is not anyone's liability. In contrast, a gold certificate holder must trust the issuer to exchange it to gold, and a holder of fiat money must trust the issuer not to dilute its value with excessive issuance.

Network effects in money

Similar to information networks, money exhibits network effects. People are likely to demand widely accepted currencies for their work. The world economy tends to converge onto a single currency. The US dollar already plays this role to a large extent. It is the most popular reserve currency and the currency of international trade. One can argue that without legal restrictions (such as the need to pay taxes in local currencies) the US dollar would dominate the global economy.

The effects of centralization induced by network effects are especially adverse in monetary networks. For example, censorship has more severe consequences: a frozen bank account causes more problems than a blocked social media account. The issue is even more concerning if physical cash is unavailable. In many developed countries, such as Sweden and the Netherlands, banks are phasing out cash to combat money laundering. Without cash, a person banned from the banking system cannot buy necessities. Money administrators can also change the rules on short notice. A recent example is the 2016 demonetization in India. High-denomination banknotes were declared invalid in an attempt to fight black markets. This sudden move caused severe economic disruption throughout the world's second-most populous country.

A money network has a unique property: all users value the content that it helps exchange. Financial administrators may abuse this property and print themselves money – a privilege that their information network counterparts do not enjoy. A social network administrator can push their writings into everyone's news feeds or inflate the reported number of views but cannot make people perceive their content as universally valuable.[5]

Switching monetary systems is hard. People cannot easily "vote with their feet", especially if the administrators abuse their position. First, it is not always apparent that abuse takes place. For instance, moderate money printing can long go unnoticed, slowly diluting savings. Second, it is hard to coordinate which another

[5]For example, as of 2020, "only" 116 million out of 2.5 billion users of Facebook follow its creator Mark Zuckerberg.

system to switch to. Uncoordinated exodus destroys the benefits of network effects. Finally, monetary administrators deliberately impede exit with legal action. Multiple independent centralized payment systems were shut down. Examples include Liberty Reserve, Liberty Dollar, and e-gold [404, 385]. Others, such as PayPal, were forced to give up their initial vision and merge with the existing financial system [204]. This state of affairs inclines rational actors to accept the corrupt status quo.

Key challenges for digital currencies

The first digital cash protocols were proposed in the early 1980s, but nearly three decades passed before the first viable solution – Bitcoin – was introduced. Why did designing a decentralized digital currency take so long?

Digital signatures provide only a part of the solution. Senders sign transactions to reliably prove their intent to spend their money. The receiver can verify the signature without relying on any authority.[6] However, asymmetric cryptography is not sufficient. Two crucial challenges have been hindering the deployment of digital cash protocols for decades.

Double-spending One cannot easily copy a physical object. A metal coin is either in the sender's or the receiver's hand. In contrast, one can effortlessly copy digital information. A malicious user can duplicate their "coin" and spend it twice. This problem is known as *double-spending*.

Balances must be stored on multiple computers to mitigate centralization risks. However, it is unclear how to ensure consistency. If two computers report different balances for the same account, how to agree on the right one?

Voting is a questionable solution in this context. Without an *identity management* system, an adversary can launch a *Sybil attack* and vote multiple times. One way to combat Sybils implies maintaining a list of all voters and only allowing each of them to vote once.[7] Nevertheless, contrary to the design goals, a party who controls the voter list becomes the central point of control. To prevent censorship, the network must allow new users to join unconditionally.

How can a network defend against Sybil attacks while providing free access?

Fair emission A digital currency must come into circulation somehow. Who should get the newly created money?

On the one hand, the system should reward the users who help maintain it. Transaction processing requires resources. If no central party allocates these resources, users must provide them. However, without strong identities, economically rational agents would not contribute. The system needs economic incentives, or it collapses under the burden of free-riders.

On the other hand, users should perceive the currency distribution as fair. Otherwise, they will not join. Unlike the fiat system, no one is forcing them to. The currency distribution must also be objectively verifiable. All users should be able to independently check that everyone else follows the rules.

How can a network automatically reward anonymous participants proportionally to their contributions?

[6] Assuming that the receiver knows the sender's public key.

[7] This class of problems is called *Byzantine fault tolerant* consensus. A prominent protocol of this class is *Practical Byzantine fault tolerance* (PBFT) [79]. Cryptocurrencies such as Ripple [81] and Stellar [250] use BFT-like protocols.

Early digital currencies

Let us mention notable pre-Bitcoin proposals of digital currency systems.

David Chaum introduced *ecash*, an anonymous digital cash protocol, in 1982 [84] and further enhanced it in 1988 [86]. Ecash users would exchange digital coins issued by a bank. A receiver would consult the bank to verify that the incoming coins had not been spent. However, the participants' identities would remain hidden from the bank due to *blind signatures*. In 1989, Chaum founded a company called Digicash to commercialize his invention. While gaining some traction in mid-1990s,[8] the company declared bankruptcy in 1998.

In 1998, Wei Dai proposed *b-money* [105]. B-money, predating Bitcoin by a decade, was in many ways similar to it. Users, identified by public keys, would independently maintain a list of all current balances. Any user would be able to generate coins by performing otherwise useless computations.[9]

A crucial piece of the Bitcoin's puzzle is *proof-of-work* (PoW). Cynthia Dwork and Moni Naor proposed PoW in 1992 as an anti-spam mechanism [128]. A sender of an electronic message would need to perform computational work. A *proof* allows anyone to verify the number of computations performed. The puzzle depends on the message to prevent re-using one solution for different messages or recipients. PoW would incur a negligible delay for regular users while deferring spammers.

Adam Back suggested using cryptographic hash functions for PoW in his 1997 Hashcash proposal [19]. The *work* in Hashcash means finding partial collisions of a cryptographic hash function. Such functions simulate random oracles. Thus it is computationally hard to find preimages or partial preimages for them. One cannot predict whether a function output satisfies a given property without calculating it. Hence PoW solutions can only be found by trial and error. Hashcash uses this property as a puzzle with an adjustable level of hardness.

In 2005, Nick Szabo proposed Bitgold [367]. His idea was to represent digital coins as "a string of bits [computed] from a string of challenge bits." The solutions to such puzzles would be linked in a chain using multiple timestamping servers to preserve integrity.

1.3 Bitcoin

Bitcoin was the first decentralized digital currency to solve the double-spending problem without a trusted third party. An unknown person under a pseudonym Satoshi Nakamoto announced Bitcoin in October 2008.[10] Shortly after, he published the source code. The original code repository was later renamed to *Bitcoin Core* – the Bitcoin's *reference implementation*. Bitcoin launched on 3 January 2009 and started slowly gaining traction in the technology community.

[8]For instance, the Dutch authorities considered using Digicash for road toll payments [85].

[9]"Anyone can create money by broadcasting the solution to a previously unsolved computational problem. The only conditions are that it must be easy to determine how much computing effort it took to solve the problem and the solution must otherwise have no value, either practical or intellectual."

[10]Nakamoto might have deliberately chosen dates with a symbolic meaning while constructing their pseudonymous identity. For instance, Nakamoto claimed to have been born on 5 April 1975. The Executive Order 6102, issued on 5 April 1933, banned private gold ownership in the US. The ban was repealed on 31 December 1974. The date for the first public announcement of Bitcoin – 31 October – may have also been chosen deliberately. On that day in 1517, Martin Luther nailed his Ninety-five Theses on the door of a church in Wittenberg, starting the European Reformation. It has been argued that Bitcoin may lead to similarly wide-reaching societal shifts [118].

Nakamoto's insight was in the way he combined Bitcoin's components. All the necessary ingredients had already been proposed, but never connected in the right way.

1.3.1 Bitcoin architecture

Let us now briefly describe the architecture of Bitcoin. We refer the reader to [274] for a historical review of Bitcoin's building blocks, to [62] and [388] for an overview of the field, and to [275] and [12] for a comprehensive technical introduction.

Nodes and P2P network The Bitcoin network consists of *nodes*, or *peers*. Each node maintains a few connections to other nodes – *neighbors*, or *entry nodes*. Nodes exchange messages via unencrypted TCP connections. They forward transactions and other protocol data to other nodes following a *gossip* protocol. Eventually, every node becomes aware of every transaction.

Each node maintains a database of all transactions that have ever taken place. Transactions are grouped into blocks. Each block contains a hash of the previous block. Hence, the blocks form a chain (the *blockchain*). A node that validates all blocks is called a *full node*.

Keys and transactions A *wallet* is a piece of software that stores cryptographic keys. Users create public-private key pairs locally. The number of possible key pairs is practically unlimited. To accept coins, the receiver generates an address from a public key. To send coins, the sender signs a *transaction* with a private key.

Internally, Bitcoin represents the state of the system as *unspent transaction outputs* (UTXO). Each UTXO specifies the amount of coins and their spending conditions. A Bitcoin transaction *consumes* UTXOs as *inputs* and creates new UTXOs. To spend a UTXO, the sender must provide a valid signature.[11] The sum of the outputs must be less than the sum of the inputs. The difference is the fee (paid to *miners*).

Mining Some nodes choose to *mine*. Mining is creating new *blocks* of transactions. A block contains the hash of the previous block, the Merkle root of new transactions, and a *nonce*. A valid block must only include valid transactions and contain a PoW solution. The solution is sufficient if the double SHA-256 hash of the block header is smaller than some target value. Miners achieve this by modifying the nonce in a trial-and-error process.

Bitcoin produces a block every 10 minutes on average. Automatic *difficulty* adjustment every 2 016 blocks ensures the constant rate of block production. If blocks were produced too quickly during a 2 016 block period, the difficulty increases; otherwise, it decreases.[12]

A miner who generates a block gets rewarded. The block reward consists of the *block subsidy* and the sum of the fees of all included transactions. Block subsidy is cut in half every 210 thousand blocks. It decreased from 50 bitcoins to 25 in 2012, 12.5 in 2016, and 6.25 in 2020. The total number of bitcoins will never exceed 21 million.

[11]To give more detail, spending conditions are defined in Bitcoin script – a Forth-like stack-based non-Turing-complete language. Spending a UTXO requires submitting the arguments such that the script evaluates to `true`, which usually involves providing digital signatures.

[12]Due to a bug, only 2 015 last blocks are accounted for difficulty re-adjustments.

Different miners may produce two valid but conflicting blocks that link to the same parent block. This situation is called a *fork*. Bitcoin nodes apply the *fork choice rule* to resolve the conflict. They compare the cumulative amount of work put into the two branches. The *heaviest* branch is considered valid. This objective criterion allows nodes to converge on a single chain without a central authority.

Bitcoin assumes that no more than half of the mining power is under adversarial control. Otherwise, a colluding majority can perform a *51% attack*, which allows the adversary to re-write blocks and potentially double-spend coins.

Bitcoin's PoW solves the double-spending problem in a Sybil-resistant way. Miners are not inclined to include conflicting transactions in the same blockchain branch. An invalid block would make the branch invalid and nullify miners' rewards.[13] Including a conflicting transaction in another branch requires controlling more hash power then the rest of the network combined. If the attacker does not control the majority of hash power, the honest branch would accumulate more work and be deemed valid according to the fork choice rule. In any case, Sybil attacks on Bitcoin are expensive. An attacker can only increase their influence in the network by committing more *physical* resources – hardware and energy.

Bitcoin's emission mechanism also elegantly solves the fair emission problem. Coins only come into existence as miners' rewards. The more hashing power is committed to Bitcoin, the harder it is to attack. Therefore, Bitcoin automatically rewards miners in proportion to their contribution to the network's security. Miners get revenue in bitcoins, but bear expenses in fiat currencies. Fierce competition forces them to sell their coins. Bitcoins "percolate" to non-mining users, which stimulates adoption and prevents capital concentration.

1.3.2 Is proof-of-work wasteful?

A standard critique of PoW is that it is "wasteful." Arrays of computers burning energy to solve a seemingly arbitrary mathematical equation may indeed look uneconomical. However, we believe this assessment is not accurate.

Markets show that Bitcoin provides value to some people. PoW is crucial for Bitcoin's properties that its users value. One such property is predictable issuance. To guarantee it in a permissionless system, producing new bitcoins must incur a cost. Energy is arguably *the* universal form of value. PoW serves as a proxy for the amount of energy committed, ensuring that producing bitcoins is costly.

The influence of Bitcoin on energy markets is non-obvious [77]. Bitcoin mining does consume a significant amount of energy (64 TWh annualized as of September 2020 [315]). However, miners rarely compete for energy with other forms of human activity. Electricity price is the most critical competition factor for miners. Thus they move to places with the cheapest energy. Low prices often indicate that the energy would otherwise have been wasted. For instance, the output of hydroelectric power plants is seasonal. In high season, it may be uneconomical to transfer the excess energy to population centers. Bitcoin utilizes this otherwise wasted power. We should note that not all Bitcoin miners use renewable energy. For instance, coal-based mining is gaining popularity in Central Asia [3].

[13] In the words of Satoshi Nakamoto, "If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth." [272]

Useful proof-of-work

One may wonder whether miners can extract more value from mining if a given amount of energy is spent regardless. Proof-of-work algorithms that allow this are known as *useful proof-of-work*, or *proof of useful work*.[14] Primecoin is an early example of a cryptocurrency with useful PoW. Instead of hash collisions, Primecoin miners search for Cunningham and bi-twin chains of prime numbers. The extra value is advancing mathematical science. Primecoin miners have found multiple long Cunningham chains.

Two main lines of reasoning oppose useful PoW. First, it complicates the security model. Miners have *two* ways to extract value from the PoW solutions they find: release them to the network and sell them elsewhere. As revenues from supporting the network only form a part of the miners' income, they become more susceptible to bribery and less committed to the network's success. It is hard to reason about such unintended consequences, which depend on unpredictable factors. Second, real-world problems are rarely perfectly tunable. Recall that PoW rewards miners in proportion to the committed energy. The network cannot measure energy expenditures directly. For PoW solutions act as a proxy, the committed amount of energy, as estimated from the PoW solutions, should predictably depend on the real committed energy. A miner should get roughly $x\%$ more solutions for $x\%$ more energy spent.

SHA-256 produces a uniformly distributed output in the range from 0 to $H_{max} = 2^{256} - 1$.[15] A solution is valid if the hash is smaller than the target t. For every t in the range from 0 to H_{max}, and for a random value r, the probability $P(h(r) < t) = \frac{t}{H_{max}}$. Therefore, the probability of finding a solution is proportional to the number of guesses. This property of cryptographic hash functions allows for fine-tuning the difficulty of PoW.

In contrast, no one knows the distribution of solutions to most real-world problems. As an example, consider protein folding. At first glance, it is a proper candidate for a useful PoW. Similarly to hash-based PoW, it involves searching for solutions in a large space by trial and error. It was also used in volunteer distributed computation projects such as Folding@Home [26]. However, we do not know how a unit increase in committed resources affects the probability of finding a solution per unit time. Therefore, we cannot parameterize the folding-based puzzle to make it X times harder for an arbitrary coefficient X.

1.3.3 Professionalization of mining

Bitcoin mining has evolved into a highly specialized industry. As mentioned earlier, Bitcoin miners search for partial collisions of SHA-256 – a general-purpose cryptographic hash function. They iterate over nonce values[16] in block headers until the hash of the header is below the target. Mining is a perfectly parallelizable task because candidate nonces are hashed independently.

Satoshi Nakamoto originally envisioned a network where every node could generate coins. Early versions of the software allowed users to generate coins using regular processors (CPUs). It quickly turned out that graphical processing units (GPUs) are more suitable for mining due to hardware parallelization. In 2011-2012,

[14] See [21] for an overview of proofs of useful work.

[15] This is a *cryptographic assumption*. It cannot be proved rigorously. No attacks have *severely* contradicted the cryptographic properties of SHA-256.

[16] Due to high mining difficulty, miners also modify other parts of a candidate block.

TABLE 1.1: PoW hash functions in selected cryptocurrencies.

Cryptocurrency	**Hash function**	**Memory-hard?**
Bitcoin	SHA-256	No
Ethereum	Ethash	Yes
Litecoin	scrypt	Yes
Monero	RandomX, CryptoNight (before 2019)	Yes
Zcash	Equihash	Yes

GPUs and configurable integrated circuits (field-programmable gate arrays, FPGA) became the primary mining equipment. In 2013, China-based manufacturer Canaan Creative announced the first specialized devices for Bitcoin mining (application-specific integrated circuits, ASIC) [219]. Fueled by competition and the rising price of bitcoin, ASICs improved rapidly. All non-specialized mining hardware quickly became unprofitable.

Mining is a highly competitive business with large capital requirements [224]. Large miners take advantage of economies of scale. They buy devices in bulk and negotiate low electricity and rent prices. Bitcoin mining is geographically concentrated. Two thirds of the mining power originate from China [315].

Mining professionalization has upsides and downsides. On the one hand, specialized mining discourages 51% attacks. To control the majority of hash power, an attacker would have to obtain specialized equipment in a large-scale data center in a region with competitive energy prices. This commitment makes miners less interested in disrupting the network. In case of a successful attack, the price of bitcoin is likely to drop. Unlike general-purpose hardware, Bitcoin ASICs cannot be repurposed. With the trust in Bitcoin undermined, the ASICs would be hard to sell, which decreases the attacker's potential profits. On the other hand, mining concentration increases the risks of collusion or government intervention. The key security assumption in Bitcoin is the absence of a colluding majority. Even a sizable colluding minority can perform attacks such as selfish mining [155]. Therefore, mining power should be under the control of a diverse group of participants.

ASIC-resistant proof-of-work

Multiple alternatives cryptocurrencies aim to discourage mining specialization. This design goal is known as *ASIC resistance*. A common path towards ASIC resistance implies using *memory-hard hash functions* for PoW (see Table 1.1). Such functions discourage parallelization by requiring frequent access to random memory regions. Unlike computation, memory access cannot be substantially optimized in custom hardware. ASIC-resistant cryptocurrencies can be mined using off-the-shelf equipment, primarily GPUs.

ASIC resistance may raise the probability of 51% attacks. GPUs, unlike ASICs, are widely available and used for gaming and scientific computing. Therefore, an attacker can sell the GPUs after the attack, at least partially recouping the initial investment.[17]

It is not clear if a cryptocurrency can retain ASIC-resistance in the long run. An economic argument suggests that mining will be professionalized for any PoW,

[17]The attacker can also *rent* hashing power only for the duration of the attack using hash-rate markets. Multiple cryptocurrencies (Ethereum Classic, Bitcoin Gold) have been 51%-attacked in practice [412].

given sufficient incentives. ASICs have been developed for memory-hard hash functions used in prominent cryptocurrencies such as Ethereum [283] and Zcash [167].

Cryptocurrency developers can discourage ASIC manufacturing by regularly and unpredictably changing the PoW hash function. This countermeasure is based on the assumption that specialized hardware takes at least a few months to develop, manufacture, and deploy. The developers of Monero, a privacy-focused cryptocurrency, used to change its hash function every six months [218].[18] Ethereum developers consider a similar strategy known as ProgPoW [284]. Changing the hash function for PoW is a breaking protocol change. Such updates require coordination and a degree of trust in the cryptocurrency developers.

Proof-of-stake

An alternative approach for Sybil protection in permissionless networks is to require the commitment of another scarce resource instead of energy. *Proof-of-stake* (PoS) is a family of designs that use the units of cryptocurrency for this purpose [22]. The key idea behind PoS is that the probability to mine a block must be proportional to the number of coins a miner holds (the *stake*). Misbehaving miners can be punished (*slashed*) by destroying or re-distributing their stake.

PoS designs include Algorand [87], Ouroboros [217], and SnowWhite [30]. Novel security issues have been identified in PoS protocols [159, 174, 65, 89]. Some authors argue that PoW is the only viable Sybil protection mechanism in Bitcoin's security model [9, 369, 298]. However, it remains to be seen if a PoS system, albeit with weaker security guarantees than PoW, proves to be beneficial. We refer the reader to [29] for a review of cryptocurrencies without PoW.

1.4 Challenges for cryptocurrencies

We identify three key challenges that Bitcoin faces: expressiveness, scalability, and privacy. These challenges are being addressed both by Bitcoin and alternative cryptocurrencies.

1.4.1 Expressiveness

In Bitcoin, spending conditions are defined in Bitcoin script. It is a simple non-Turing complete language. The simplicity of Bitcoin script makes it easier to analyze. In particular, miners can put an upper bound on the number of computational steps each transaction execution would take.

On the other hand, complex financial contracts are hard or impossible to express in Bitcoin script. Bitcoin developers address this issue by introducing a more efficient transaction structure [410], new opcodes [331], support for external data oracles [127], and high-level programming languages [281, 411].

Ethereum [70, 407] is a cryptocurrency that supports more expressive programs. It incorporates a virtual machine that executes code in a Turing complete language. Such programs are called *smart contracts*.[19] Each contract is stored at a unique address along with its *state*. Users issue transactions to call contracts, which may, in

[18]In 2019, Monero changed the strategy and switched to a new hash function without plans for further modifications [113].

[19]The term "smart contract" dates back to 1997 and refers to digitally encoded and automatically executed agreements. This term is widely used to refer to Ethereum programs and sometimes used to refer to Bitcoin scripts. We give a more elaborate introduction to smart contracts in Chapter 9.

turn, call other contracts. Interoperability and rich programming environment allow for more flexibility but introduce new security risks.[20]

1.4.2 Scalability

We observe a trade-off between transaction throughput and security of blockchain networks. In Bitcoin's security model, users must be able to validate all transactions using universally available hardware. Therefore, the network as a whole can only process as many transactions as one node. This design choice limits Bitcoin's throughput to tens of transactions per second [102].

The simplest way to address this issue is by increasing the block size or decreasing the time between blocks. Bitcoin Cash [225] takes this approach. However, scaling the network for widespread global adoption requires a throughput increase by many orders of magnitude. Validating all transactions in real time would require resources unavailable to most users. Tweaking constants for scalability is an interim solution at best.[21]

Another approach is *sharding* [175, 238]. This term is borrowed from database design, where transactions are split between parts of a database (*shards*). The key challenge in blockchain sharding is *cross-shard communication*: nodes in one shard must ensure that transactions in other shards are valid. The upcoming[22] Ethereum 2.0 is built on a sharded architecture [348].

Finally, *off-chain protocols*, also known as *layer two* (*L2*) protocols, move the bulk of the transactions off the blockchain. In an off-chain protocol, users exchange signed transactions without submitting them to the blockchain but resolve disputes using the underlying blockchain protocol (referred to in this context as *layer one*, or *L1*). Bitcoin's Lightning Network [300] takes this approach. Ethereum-based L2 protocols include state channels [129, 313, 258], refereed computation protocols [372, 211], Plasma [299], and rollups [165, 181]. We refer the reader to [188] for a comprehensive overview of off-chain protocols.

1.4.3 Privacy

Privacy is essential in money for both ethical and technological reasons.

From an ethical standpoint, people should have the right to choose whom they disclose their activity. Transactions in the current financial system are linked to peoples' identities and closely monitored. Undocumented people cannot get access to basic finance altogether. Banks freeze accounts in case of "suspicious" behavior. Modern digital technologies facilitate data collection on a massive scale. The concentration of power over money in the hands of governments and corporations creates a breeding ground for human rights abuse. Eradication of cash exacerbates the issue [64].

From a technical standpoint, a digital currency should be *fungible*. Fungibility means that units of a currency of equal value are indistinguishable. Non-fungible currency fails to act as a unit of account. Merchants may instigate blacklisting of currency units with a "bad" transaction history. This practice incurs a tax on everyone as merchants start discounting incoming transactions based on which particular currency units they contain.

[20] We refer the reader to [24] for an overview of smart contract platforms.

[21] Not to mention coordination issues: raising the block size is a non-compatible upgrade.

[22] Note though that the developers repeatedly shifted the launch date.

Bitcoins are not entirely fungible. Each coin has a unique history that can be tracked up until its creation as a miner reward. (Note that such histories have multiple threads because values are split and merged in transactions.) Multiple companies provide the service of blockchain analytics to enable blacklisting "tainted" coins [139, 306].

We can classify privacy attacks on cryptocurrencies into *transaction graph analysis* and *network analysis*. In transaction graph analysis, the attacker studies the graph built from the publicly available blockchain data [319, 11, 254, 280, 328]. The simplest countermeasure is to generate a new address for each transaction.[23] However, this is insufficient to protect against modern blockchain analysis methods. *Mixing* is a more involved technique. In a mixing protocol, a group of users collaboratively create a transaction with multiple inputs and multiple outputs. Each user gets the same amount of coins as they put in, minus fee. The links between the inputs and the outputs of the same user are entangled. The key challenge is trustless mixing coordination. Multiple mixing protocols have been proposed [248, 61, 332, 393]. In network analysis, the attacker participates in the cryptocurrency P2P network to track or influence data propagation. Network-level attacks include eclipse attacks [245, 193], global network disruption [14], and transaction deanonymization [37].

Privacy-focused cryptocurrencies

Multiple privacy-focused cryptocurrencies have been developed. Dash [111] coordinates mixing using a network of *masternodes*. Monero [261] implements the CryptoNote protocol [336]. It uses ring signatures to entangle the transaction graph and Bulletproofs [68] to hide transaction amounts. Zcash [415] implements the Zerocash protocol [28, 201] – an improvement of an earlier Zerocoin protocol [256]. It uses zk-SNARKs [27] to hide transaction data.[24] Grin [186] and BEAM [25] implement the MimbleWimble protocol [206]. They hide transaction amounts using Pedersen commitments.

Privacy-focused cryptocurrencies face not only technological but also economic hurdles. Privacy technologies work best with a large number of users (the *anonymity set*). However, privacy-focused cryptocurrencies may struggle to get enough users. Exchanges are less incentivized to support privacy-focused cryptocurrencies. Offering them brings little trading volume, but incurs technical costs and legal risks. Privacy-focused cryptocurrencies may also be harder to use due to the computational requirements of advanced cryptography. For example, most Zcash transactions do not use its zero-knowledge cryptography [311, 34], likely because of its high computation requirements and the difficulty of integrating it into third-party wallets. Attacks on privacy-focused cryptocurrencies have also been described [311, 269, 35, 36, 384].

In the meantime, the two most popular cryptocurrencies improve their privacy technologies. Privacy solutions based on zero-knowledge proofs are being developed on Ethereum. Bitcoin's privacy is also set to improve with better mixing.[25] It remains to be seen which approach provides more robust privacy – improving the privacy of existing cryptocurrencies or developing new privacy-focused ones.

[23]This complicates a widespread use case – collecting donations to a static Bitcoin address published on a website or a social network page. A safer way to receive donations would be to deterministically generate a new address for each donation from a master key.

[24]Zero-knowledge proofs are also used to improve blockchain scalability [60].

[25]Some argue that for these reasons privacy-focused cryptocurrencies are not a viable market niche [178].

1.5 Our contributions

This thesis is structured as follows.

- Part I focuses on the privacy of Bitcoin and privacy-focused cryptocurrencies.
  - In Chapter 2, we introduce P2P networking in cryptocurrencies.
  - In Chapter 3, we describe and evaluate a network-level privacy attack on Bitcoin and three privacy-focused cryptocurrencies. We describe a method that allows an adversary to link transactions issued from the same node based on propagation timings.
  - In Chapter 4, we study the privacy of mobile cryptocurrency wallets. We show that few wallets satisfy our minimal privacy criteria.
- Part II is dedicated to security and privacy of the *Lightning Network* (LN) – a Bitcoin-based off-chain protocol.
  - In Chapter 5, we outline the history of layer-two protocols in Bitcoin and provide a technical introduction to the Lightning Network.
  - In Chapter 6, we introduce a *probing* attack on the LN. Our method lets an adversary accurately reveal users' balances, assumed to be private. We implement and evaluate the attack on the Bitcoin testnet.
  - In Chapter 7, we quantitatively assess the probability of three privacy attacks on the LN. From a simulation based on an LN snapshot, we conclude that compromising a few influential nodes significantly raises the attack success rates.
  - In Chapter 8, we quantify the effects of a known limitation on concurrent payments in the LN on its throughput.
- Finally, Part III explores the security and privacy of Ethereum smart contracts.
  - In Chapter 9, we provide the necessary background on Ethereum.
  - In Chapter 10, we propose Findel – a declarative language for financial contracts – and implement it in Solidity, Ethereum's main contract language.
  - In Chapter 11, we introduce SmartCheck – a static analysis tool for Solidity. We classify and codify common Solidity bugs and evaluate our tool on a large sample of real-world Ethereum contracts.
  - Finally, in Chapter 12, we propose a cryptographic scheme for a more privacy-preserving know-your-customer (KYC) procedure.

Part I

Network privacy in Bitcoin and privacy-focused cryptocurrencies

Chapter 2

P2P protocols in cryptocurrencies

Each cryptocurrency relies on a P2P network to disseminate transactions and other messages. Ensuring that all nodes reliably obtain the relevant data is a prerequisite for reaching consensus. Resilience, privacy, and performance are crucial design considerations for the P2P layer.

This introductory Chapter provides the background on the network-layer aspects of cryptocurrencies. First, we outline the design goals for a cryptocurrency P2P network. Then, we describe the technical details of the P2P protocol in Bitcoin[1] and selected alternative cryptocurrencies. Finally, we classify cryptocurrency nodes from a networking perspective and provide a review of network-level attacks on cryptocurrency privacy.

2.1 Design goals for a cryptocurrency P2P network

Unlike client-server protocols, all nodes in a P2P network have equal roles. A cryptocurrency P2P protocol has two major tasks. First, it provides new peers with the entire set of existing blocks. Second, it continuously disseminates new blocks and transactions as they appear.

While adhering to the same general philosophy as file-sharing (discussed in Section 1.2.1), cryptocurrency P2P protocols aim at slightly different goals [125]. First, a cryptocurrency P2P protocol is tailored to disseminating one specific dataset, whereas file-sharing is content-agnostic. Moreover, each block in the blockchain depends on the previous one. This logical structure leads to a trade-off: downloading blocks out of order in parallel is faster but may lead to bandwidth waste if some of them turn out to be invalid.

Second, content addressing is not as essential in cryptocurrencies as in file-sharing, where locating the peer that hosts the required file is the key challenge. All cryptocurrency peers store the same set of blocks (except possibly a few latest blocks).

Third, cryptocurrencies demand stringent security guarantees. Cryptocurrency P2P protocols prioritize resilience over efficiency to defend against denial-of-service attacks, eclipse attacks, and network data analysis. For instance, Bitcoin peers deliberately connect to peers from a diverse set of IP regions and autonomous systems [276]. File-sharing protocols, on the contrary, prioritize local connections to utilize the ISP's local networks instead of the global Internet [414, 402].

[1]We refer the reader to [48, 173] for more comprehensive documentation.

2.2 P2P protocols in cryptocurrencies

Let us now describe the P2P protocol used in Bitcoin and an alternative P2P protocol for cryptocurrencies – Dandelion. Cryptocurrencies based on a modified Bitcoin Core codebase, such as Dash and Zcash, inherit its P2P protocol. Others, such as Ethereum, implement their own P2P protocol [193].

2.2.1 Bitcoin P2P protocol

We now describe the technical details of the P2P protocol in Bitcoin.

Bootstrapping Any P2P network faces the bootstrapping problem: a new peer does not know any other peers. Bitcoin Core provides two bootstrapping methods: *DNS bootstrapping* and *seed nodes*. The code contains hard-coded DNS records that resolve into IP addresses of stable peers. If DNS bootstrapping fails, the new peer connects to some of the *seed nodes* whose IP addresses are also hard-coded.

Connections and peer discovery Bitcoin peers establish unencrypted TCP connections.[2] A peer tries to maintain 10 outgoing connections: eight connections for relaying all types of messages and two dedicated connections for block propagation [104]. A peer allows up to 117 incoming connections by default.

Peers exchange information about the IP addresses of known peers. A new peer advertises its IP address to its neighbors in an `addr` message. Upon receiving an `addr` message, a peer may decide to relay it to some of its neighbors. A peer can ask its neighbor which peers it is aware of using a `getaddr` message. The neighbor responds with an `addr` message containing up to 1 000 addresses of recently seen peers.

After establishing the initial connections, a new peer asks the bootstrapping peers about other peers and connects to those. It then disconnects from the bootstrapping peers to keep them available for new joining peers. Each peer maintains a persistent database of IP addresses of known peers. Ideally, this database should suffice for all subsequent connections to the network. DNS bootstrapping and seed nodes remain available as a fallback mechanism.

Initial block download After connecting to the network, a new peer downloads and validates all previous blocks. This process is known as the *initial block download* (IBD). A peer may decide to delete most blocks after validating them (*pruning*). It may also enable *indexing*, allowing for fast querying of transactions in the local database.

Bitcoin Core supports two IBD modes: *blocks-first* and *headers-first*. In blocks-first IBD, a peer downloads the blocks sequentially from a single peer. This process ensures that the peer downloads the next block only if the previous block is valid. Blocks-first IBD is slow and depends on a single peer. Headers-first IBD mode, introduced in 2015 [98], addresses these shortcomings. A peer first downloads all block headers (using `getheaders` and `headers` messages) and ensures that they are well-formed and contain the necessary PoW. The peer then downloads the full blocks from multiple peers in parallel. Note that a peer cannot validate a block by only

[2] Different networks use different default ports: 8333 for Bitcoin, 18333 for the Bitcoin testnet, 8233 for Zcash, 18080 for Monero, 9999 for Dash.

looking at its header. A block may contain a valid header with sufficient PoW despite containing an invalid transaction. However, headers-first IBD is beneficial in most practical scenarios.

Propagation of transactions and blocks Bitcoin peers exchange blocks and transactions in a three-step message exchange. A sending peer first announces the hash of a new object with an inventory (`inv`) message. Upon receiving an `inv`, another peer may reply with `getdata` to receive the full data. The sending peer replies with a `block` or a `tx` message for blocks and transactions, respectively.

Developers have introduced multiple improvements to Bitcoin's P2P protocol. In the initial protocol, each transaction is propagated twice: first on its own, then as a part of a block. An optimized block propagation protocol called *compact blocks* [99] eliminates this redundancy. Peers share block *sketches* that describe its contents using short transaction identifiers. A sketch also contains the transactions that the receiving peer lacks (as *predicted* by the sending peer). The receiving peer reconstructs the block from the sketch and requests the missing transactions with additional `getblocktxn` queries if necessary. Erlay [277] is another P2P optimization that reduces the number of redundant `inv` messages that peers exchange. Miners have special requirements for the P2P protocol and use dedicated networks [157, 164] for fast block dissemination.

Broadcast randomization A gossip-based propagation in a P2P network may reveal private information such as the original message sender. An attacker may analyze the timestamps of messages received from different peers. Bitcoin uses *broadcast randomization* to protect against network attacks. A peer introduces a random delay before announcing a transaction to each neighbor. This mechanism, called *diffusion*, replaced [409] another randomization technique called *trickling*. With trickling, a peer announces a transaction to a random subset of neighbors. Such subsets are chosen once in a fixed-length interval. Replacing trickling with diffusion provided only a modest privacy improvement [158].

2.2.2 Dandelion

Dandelion [396, 160] is an alternative P2P protocol for cryptocurrencies designed for stronger privacy. It addresses the key issue with gossip protocols – the symmetry of message propagation. Dandelion peers propagate a message in two stages: the *stem phase* and the *fluff phase*. On the stem phase, a peer only relays a message to one randomly selected neighbor. The peer that receives the message randomly chooses whether to continue the stem phase (forward the message to one random neighbor) or start the fluff phase (relay the message to multiple neighbors). Dandelion makes a network adversary confuse the original message sender with the initiator of the fluff phase. The authors show that the protocol achieves much stronger anonymity than Bitcoin's current P2P protocol. The drawbacks of Dandelion include increased propagation delays and sensitivity to DoS attacks at the stem phase [361]. As of 2020, Dandelion is not introduced in Bitcoin but used in privacy-focused cryptocurrencies Monero, Grin, and Beam.

2.3 Taxonomy of nodes

Full Bitcoin nodes download, validate, and share the whole blockchain. These activities consume bandwidth, storage, and processing power. Alternative types of nodes offer ways to use Bitcoin with more lenient requirements.

Pruned nodes The simplest protocol change to decrease storage requirements is to discard (*prune*) old blocks. A pruned node downloads and validates all blocks, and then deletes the old blocks. Pruning reduces storage requirements significantly. Bitcoin Core allows allocating as little as 550 MB for the most recent blocks (the full Bitcoin blockchain requires 300 GB as of September 2020). Pruned nodes do not support transaction indexing and cannot serve old blocks to others.

SPV nodes *Simplified payment verification* (SPV) is another approach, which is outlined in the original Bitcoin paper [273]. Instead of downloading full blocks, an SPV node only asks for block headers and transactions of interest. The peers respond with the requested transactions and a Merkle proof that they are included in some block. SPV is based on weaker security assumptions than the full protocol. Recall that a peer cannot verify a block based only on its header. A malicious peer may provide a Merkle proof that a transaction is included in an invalid block with sufficient PoW. SPV nodes must therefore ensure they are not eclipse-attacked and query block headers from multiple independent sources.

Moreover, SPV provides weaker privacy. In the naive implementation, the full node learns the addresses that belong to the SPV node. To mitigate this threat, modern SPV uses *Bloom filters* – probabilistic data structures that allow checking whether an element belongs to a set. A Bloom filter may produce false positives, wrongly reporting that an element belongs to a set, but never produces false negatives. If an element is in the set, the filter always indicates that. An SPV node submits a Bloom filter to a full node to specify the addresses it is interested in. The full node replies with the transactions involving all addresses that pass the filter. The SPV node discards the false positives locally. The privacy guarantees of Bloom filters have also been questioned [179].

Wallets with trusted remote nodes Finally, one may use Bitcoin without directly connecting to its P2P network. This mode of operation, implemented by many mobile wallets, requires a *trusted node*. A wallet stores the keys and signs transactions locally but can only publish them through a trusted node. The trusted node can lie about the blockchain state, deny service, learn what addresses a user controls, and link them to their IP address. The user may gain partial protection by connecting to the trusted node through Tor – an anonymization overlay network [122]. However, the administrator of the trusted node can use other methods to associate user's transactions (e.g., by making the wallet send a cookie along with transactions). On the other hand, it gets harder for a global attacker to distinguish users that broadcast transactions from the same trusted node.

2.4 Network-level privacy in cryptocurrencies

We now outline various types of network-level attacks on the privacy of cryptocurrency users.

First, an adversary can perform a *denial-of-service* attack by flooding the network with messages to overwhelm honest nodes. The Bitcoin's P2P protocol addresses this threat: peers do not re-broadcast messages and ban peers that send invalid data or exhibit other unexpected behavior.

Second, in an *eclipse attack*, an adversary takes control of all connections between the victim and the rest of the network. Controlling the victim's view of the network allows the attacker to provide incorrect data, censor transactions, and collect network traffic for future analysis. Eclipse attacks may have severe consequences for miners, as the attacker inhibits or slows down block propagation. In layer-two protocols such as Lightning, network-level attacks can lead to direct loss of funds [323].

Third, in a *network data analysis attack*, an adversary extracts information from the P2P messages and infers private information about the victim. For instance, timing differences in transaction announcements from different peers may reveal its original sender. An adversary can establish multiple connections and collect traffic from many vantage points for more accurate results.

Fourth, the adversary can infer the *network topology*, i.e., which pairs of nodes are connected, and use this information in further attacks. Multiple topology estimation attacks have been described.

One attack [257] exploits some peculiarities in the update mechanism for the address database (`addrMan`) in Bitcoin Core. Each Bitcoin peer maintains a database of known IP addresses, along with corresponding "freshness" timestamps. Counterintuitively, Bitcoin peers only update timestamps for peers that they maintain outgoing connections with.[3] For incoming connections, the peer preserves the first timestamp relayed along with the address. The authors implement a tool that exploits these rules to estimate the network topology.

Another method [278] infers the network topology by analyzing transaction propagation timing. The authors test the method on the real-world Bitcoin network with high recall and precision. They also show that an inappropriately parameterized trickling mechanism can reduce the network's resilience against topology discovery.

Finally, a measurement study [403] of Bitcoin's *unreachable* peers (i.e., peers behind NATs and firewalls) reports, among other findings, that a large share of Bitcoin transactions originate from only two mobile applications.

2.4.1 Network-based transaction deanonymization

In *transaction deanonymization* attacks, an adversary is trying to associate the victim's identities used inside and outside of the cryptocurrency protocol. A more concrete task towards this goal is to reveal the victim's IP address, which is often linked to their geographical location and real-world identity. A related task is transaction clustering, whereby an attacker reveals the hidden relationships between transactions.

Heuristics based on relaying behavior The first work on deanonymization based on network analysis [222] proposes the following technique. An adversary connects to all listening Bitcoin nodes and logs the traffic. The attacker then hypothesizes, for each transaction, that one of the IP addresses that has relayed it "owns" it (i.e., to

[3]After an update of Bitcoin Core in March 2015, the attack is no longer possible.

possess the corresponding private key). The method considers multiple cases of relaying behavior. More than 90% of transactions conform to the "normal" relaying pattern, where each node relays a transaction at most once. In that case, the method uses the *first relayer heuristic*: the first IP to relay a transaction is assumed to own its inputs and outputs. This correspondence is not guaranteed due to network delays and broadcast randomization. If a transaction is relayed from one IP address only and not re-relayed by anyone else, the single relayer is assumed to be the owner. If a single IP addressed relayed the same transaction twice (while many others relayed it once), the single re-relayer is assumed to be the owner.[4] Other types of anomalous relaying behavior are excluded from consideration.

The data is then converted into the form of *tuples* that include the transaction identifier, the Bitcoin address, and the IP address assumed to own the transaction. A separate tuple is created for each input and each output of each considered transaction. Tuples are then interpreted as "votes" in favor of a hypothesis that an IP address owns a Bitcoin address. For each Bitcoin address, the authors consider separately the tuples where it appears as an input and as an output. If the confidence is high for inputs, the IP address is assumed to own the Bitcoin address. If the confidence is high for outputs but low for inputs, the relationship cannot be identified. If both metrics are low, there is likely no association between the IP address and the Bitcoin address in question.

The authors apply the method to 5.6 million Bitcoin transactions recorded in 2012-2013. They only consider 3.9 million transactions with a single input. With highly conservative constraints, they map "between 252 and 1 162 Bitcoin addresses to the IPs that very likely own them." The "vast majority" of the final mappings are obtained from anomalous relaying behavior. The first relayer heuristic yields poor results on transactions relayed normally. The authors provide two explanations. First, the users may follow the recommendation to generate a new Bitcoin address for each transaction. Second, the first relayer heuristic itself may be "uneffective at best and invalid at worst."

Deanonymization based on entry sets A similar attack [37] correlates Bitcoin transactions with IP addresses. It exploits the fact that each Bitcoin node connects to a random set of *entry nodes*, which is used as a fingerprint. Entry sets allow the attacker to differentiate nodes behind NAT that use the same public IP address. The paper distinguishes between Bitcoin *servers* (nodes that accept incoming connections) and *clients* (nodes that do not). The authors also describe an attack that prevents the victim from using Tor by abusing the Bitcoin's anti-DoS mechanism [40].

Assuming no usage of Tor, the attack proceeds as follows. The attacker is expected to know the list of IP addresses of Bitcoin clients whose transactions it wants to deanonymize. First, the attacker establishes multiple *parallel* connections to all nodes that accept incoming connections. Then, the attacker learns the entry nodes of the victim clients from the `addr` messages they advertise upon connecting.[5] Finally, for each new transaction, the attacker considers the first 10 IP addresses that have relayed it. Based on this set's intersection with known entry sets, the attacker estimates the probability that the transaction originates from a given victim node.

[4]The relaying rules in Bitcoin Core that generally disallow nodes to re-relay transactions, but make an exception for the sender and the receiver.

[5]Note that the method assumes that the attacker is capturing the traffic when the victim is connecting to the network.

Combining blockchain analysis with network analysis Another attack [279] clusters Bitcoin addresses and associates them with IP addresses by combining transaction graph analysis and network analysis. The attacker determines the transaction originator using the first relayer heuristic. It uses two listening nodes and applies multiple heuristics to discard the "obviously false mappings" between transactions and their first relayers. The authors show that most clusters obtained through blockchain analysis cannot be directly associated with a single IP address. Likewise, most IP addresses cannot be associated with one cluster. However, a small number of participants exhibit correlations that make them prone to such attacks.

Network-level attacks on privacy-focused cryptocurrencies Network-level attacks on privacy-focused cryptocurrencies have also been described [384]. The time it takes for a remote node to reply to a specific request reveals whether that node is the sender of a given transaction. The implementation of zero-knowledge proofs in Zcash allows for timing side-channel attacks. Grin allows for linking transaction senders and receivers [55].

Chapter 3

Deanonymization of transactions with network analysis

In this Chapter, we describe a novel network-level deanonymization attack on cryptocurrencies.[1] We show how a global passive adversary can cluster transactions issued from the same node and correlate the clusters with IP addresses.

Our method is based on analyzing the timings of transaction announcements. We, as the authors of [37], go beyond the first relayer heuristic [222] and consider multiple announcements for each transaction. First, we collect the data using geographically distributed listening nodes. We then apply carefully chosen weight functions to transaction announcement timestamps. We observe that the weight vectors of transactions issued from the same node exhibit stronger correlations. This technique allows us to cluster such transactions with high accuracy. We also unpack address advertisement messages (`addr`), which may help link transaction clusters to IP addresses of the corresponding nodes under certain assumptions.

We test our method on Bitcoin and three privacy-focused cryptocurrencies. We cluster our own transactions in the Bitcoin testnet and Zcash with high levels of precision and recall. In particular, we can cluster Zcash transactions that involve both transparent and shielded addresses. We estimate the cost of a full-scale attack on the Bitcoin mainnet at hundreds of US dollars, feasible for a low budget adversary. We also prove the applicability of our technique to Dash and Monero.

Our clustering method may act as a complement to transaction graph analysis. The network-based analysis may reveal a relationship between transactions with no common public keys used in their inputs and outputs. Analogously, transactions sharing some addresses may be announced though different sets of nodes.

3.1 Transaction clustering with timing analysis

Each transaction is initially introduced by a single peer – the *source*. The source first announces a transaction to its immediate neighbors (*entry nodes*). They, in turn, announce the transaction to their neighbors, and so forth.

Each peer is typically receiving announcements for the same transaction from multiple neighbors. Intuitively, the first peer to announce a given transaction is likely to be close to its source. Earlier network-based deanonymization attacks [222] are based on the *first relayer heuristic* – an assumption that the first peer to announce a transaction *is* the source. An attacker connects to all nodes and derives the correspondence between transactions and their first relayer′ IP addresses. This heuristic

[1]This Chapter is based on [41, 43]. This work was partially supported by the Zcash Foundation grant 2017Q4-24 [161].

assumes that the attacker establishes connections to all nodes and that all nodes announce a new transaction to all neighbors as soon as they become aware of it. Both assumptions do not fully hold in practice. First, some nodes decline incoming connections. The attacker cannot connect to such nodes directly and learns about their transactions from other nodes. Second, cryptocurrencies use *broadcast randomization* methods to break the latter assumption. In particular, Bitcoin uses *diffusion*, whereas Zcash uses *trickling* (see Section 2.2.1 for details).

Our goal is to divide all transactions into clusters, where each cluster corresponds to one source. Compared to [222], we go beyond the first relayer heuristic by considering multiple IP addresses for each transaction. We hypothesize that propagation patterns of transactions issued from the same source are similar because they are announced through the same set of entry nodes. To exploit this similarity, we first connect to all nodes and log the timestamps of transaction announcements. We then characterize transaction propagation patterns with *weight vectors*. This technique distinguishes our method from [37], where the attacker correlates transactions with clients based on how many of the transaction's first relayers fall into a known entry set. Each element of a vector corresponds to a node[2] that has announced at least one transaction to us. For each transaction, we assign decreasing non-zero weights to the first N nodes that have announced it and a zero weight to all others. We expect transactions from the same source to have a stronger correlation between the weight vectors compared to transactions from different sources.

As an example, consider a source that announces three transactions tx_1, tx_2, and tx_3 via eight entry nodes $p_1, \dots, p_8$. If all transactions are broadcast via the same subset of the entry nodes, for instance, p_1, p_2, and p_3, the transactions would be easy to correlate: their weight vectors would have non-zero elements at positions 1, 2, and 3. But due to broadcast randomization, the following scenario is more typical: tx_1 is relayed via $p_{\{1,2,3\}}$, tx_2 via $p_{\{3,4,5\}}$, and tx_3 via $p_{\{5,6,7\}}$. With our technique, the correlation between tx_1 and tx_2 and between tx_2 and tx_3 would be noticeable, considering that weight vectors are sparse. This observation allows us to reveal not only the relationship between these two transaction pairs but also among all three transactions. The technique is also applicable for transactions initially broadcast from behind a trusted node (see Section 2.3 for the taxonomy of nodes). In that case, a cluster represents transactions from multiple clients connected to the same full node.

3.1.1 Weight functions and clustering

Let tx be a transaction. Let $p^{tx} = [p_1^{tx}, p_2^{tx}, \dots, p_N^{tx}]$ be the first N IP addresses that have announced tx to us. Let $t^{tx} = [t_1^{tx}, t_2^{tx}, \dots, t_N^{tx}]$ be the vector of the corresponding *relative* announcement timestamps. A *relative* timestamp is defined as $t = t_i^{abs} - t_0^{abs}$, where t_{abs}^i is the Unix timestamp of the announcement of tx_i. In other words, we subtract the timestamp of the first announcement of each transaction from the timestamps of all its announcements. For each $p_i^{tx} \in p^{tx}$, we assign a parameterized weight:

$$w_k(p_i^{tx}) = e^{-(t_i^{tx}/k)^2}$$

The weight function reflects the decreasing importance of every next announcement. p_1 is assigned the maximum weight of 1, other nodes receive lower weights. Our experiments show that this function family yields better clustering (compared

[2]We identify nodes by their IP addresses.

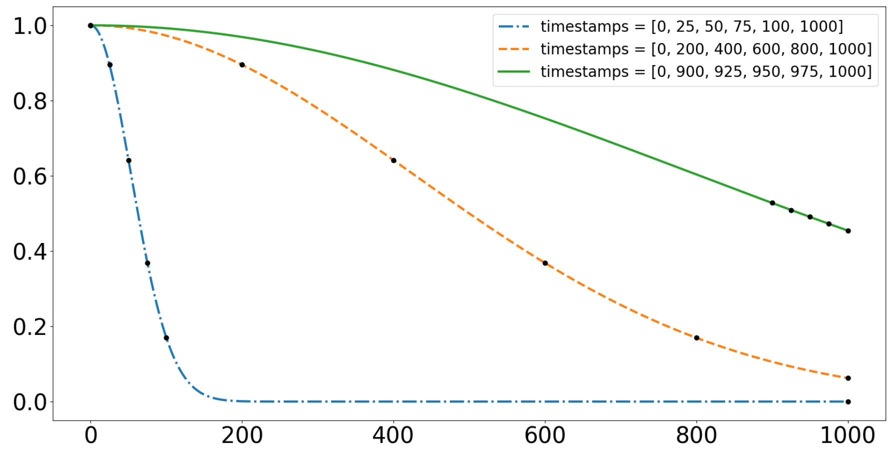


FIGURE 3.1: Weight functions for three timestamp vectors.

to $1/(kt)$ and e^{-kt}). The intuition is that it gives greater weights to a certain window depending on k, while exponentially decreasing outside of it. Moreover, the window size is adjusted for each vector.

For each p^{tx}, we want to use such w_k that gives sufficient variance among the weight values. Weights quickly fall to nearly zero if k is too low and stay close to one if k is high. We choose k^{tx}_{opt} such that the weight of the median value t^{tx}_{med} in t^{tx} is equal to 0.5:

$$k^{tx}_{opt} = \frac{t^{tx}_{med}}{\sqrt{-\ln(0.5)}}$$

This choice of k distributes the weights for any t^{tx}: they neither stay close to one nor quickly fall to zero (Figure 3.1).

For each transaction, we evaluate the vector of weights:

$$w^{tx} = w_{k^{tx}_{opt}}(t^{tx})$$

Let X be the set of all transactions we consider. Let P be the set of IP addresses of nodes that appear in at least one of p vectors in X:

$$P = \bigcup_{tx \in X} p^{tx}$$

We define an extended weight vector v_{tx} for each tx by setting the weight of nodes in $P \backslash p^{tx}$ to zero and sorting the values in the weight vectors w. r. t. the alphabetical order of P. We then calculate a matrix where an element in i-th row and j-th column is the Pearson correlation of the extended weight vectors v_{tx_i} and v_{tx_j}. This matrix can supposedly be transformed into a block-diagonal matrix with blocks (clusters) corresponding to transaction sources. To reveal the clusters, we use spectral co-clustering [119] implemented in the Python `sklearn.cluster.bicluster` module [290, 227]. Given an input matrix A, the algorithm forms A_n as follows:

$$A_n = R^{-1/2} A C^{-1/2}$$

R is the diagonal matrix with entry i equal to $\sum_j A_{ij}$, and C is the diagonal matrix with entry j equal to $\sum_i A_{ij}$. This defines the singular value decomposition (SVD) of the normalized matrix A_n. The $l = \lceil \log_2 k \rceil$ singular vectors $u_2, \ldots, u_{l+1}$ and $v_2, \ldots, v_{l+1}$ of A_n may be used to solve a real approximation of the minimal cut problem. Let U be a matrix with columns $u_2, \ldots, u_{l+1}$, and similarly for V and $v_2, \ldots, v_{l+1}$. Then Z is defined as:

$$Z = \begin{bmatrix} R^{-1/2} & U \\ C^{-1/2} & V \end{bmatrix}$$

The rows of Z are then clustered using the k-means algorithm to obtain the desired partitioning.

3.1.2 Measuring clustering quality

We use the Rand score as an external metric of clustering quality (see [8], Section 4.2). A clustering algorithm decides for each pair of elements, whether to put it in the same cluster or different clusters. Let SS, SD, DS, and DD be the numbers of transaction pairs defined as follows:

- SS: same cluster, same category[3] (our transactions in one cluster);
- SD: same cluster, different category (our and foreign transactions in one cluster);
- DS: different cluster, same category (our transactions in different clusters);
- DD: different cluster, different category (our and foreign transactions in different clusters).

The Rand score reflects the proportion of correct decisions:

$$R = \frac{SS + DD}{SS + SD + DS + DD}$$

Note that this assessment only considers clusters with "our" transactions, because we do not know whether any two "foreign" transactions should have been assigned to the same cluster.

We parameterize this metric with the minimal number of our transactions in a cluster required to consider this cluster in the calculation. In our experiments, we only consider clusters with at least two of our transactions. With no such threshold, large clusters with one of our transactions disproportionately increase DD and bring the score close to 1, which does not reflect the subjective amount of information an adversary acquires.

3.1.3 Measuring the degree of deanonymization

We measure the success of the attack using a quality score based on the *anonymity degree* [120]. The goal is to assign to each transaction a probability that it originates from $S_{control}$. Initially, all transactions have equal probabilities. The attacker adjusts the probabilities based on clustering results. The anonymity degree measures the amount of information the attacker gains.

Let p_i be the probability that a transaction i originates from $S_{control}$. K is the total number of transactions. The entropy is calculated as:

[3] Here we consider two categories: "our" and "foreign" transactions.

$$H = -\sum_{i=1}^{K} p_i log_2(p_i)$$

The maximum entropy is:

$$H_{max} = log_2(K)$$

The anonymity degree is defined as:

$$d = \frac{H}{H_{max}}$$

Our goal is to put transactions that originate from one target source $S_{control}$ into one cluster. Out of K captured transactions, k are issued from $S_{control}$. For each transaction i, the a priori probability of it having originated from $S_{control}$ is $p_i = k/K$.

We then divide all transactions into clusters. However, multiple clusters may correspond to $S_{control}$. To account for this, each cluster is assigned a *weight* that reflects how likely this cluster represents $S_{control}$. Consider an example. The attacker captures 10 transactions $t_0, \ldots, t_9$. Five of them, $t_0, \ldots, t_4$, originate from the target source $S_{control}$. The clustering algorithm yields three clusters: $c_a = \{t_0, t_1, t_2, t_3\}, c_b = \{t_4, t_5, t_6, t_7\}, c_c = \{t_8, t_9\}$. The attacker knows that t_0 and t_4 originate from $S_{control}$ and assigns a weight of 0.5 to clusters c_a and c_b, and a weight of 0 to cluster c_c. Therefore, the total "probability weight" of transactions from $S_{control}$ is distributed evenly among c_a and c_b ($t_0, \ldots, t_7$). Note that the true distribution is 1 for transactions $t_0, \ldots, t_4$ and 0 for all others.

Finally, we calculate the *adjusted* anonymity degree accounting for cluster weights. We calculate the median square error e between the vectors of probabilities p_i derived by the attacker and the actual probabilities. The adjusted anonymity degree is defined as follows:

$$d_{adj} = 1 - (1 - e) \times (1 - d)$$

Consider two edge case examples. If $e = 0$ (the attacker correctly guessed the $S_{control}$ cluster), $d_{adj} = d$. If $e = 1$ (the attacker's cluster weights do not at all reflect the reality), $d_{adj} = 1$ (the system retains full anonymity).

3.2 Implementation details

We use a modified Bitcoin network probing tool `bcclient` [309] to maintain parallel connections to peers and log incoming messages. Multiple parallel connections increase our chance to be among the first peers to learn about a new transaction despite broadcast randomization. The tool is relatively easy to adapt for usage with Dash and Zcash, which mostly inherit the networking layer from Bitcoin Core. We re-compile `bcclient` with modified constants (port numbers, protocol magic bytes, DNS seeder addresses). For Monero, which is not based on the Bitcoin Core codebase, we modify the reference implementation (`monerod`). We add the required logging and disable the built-in limits on the total bandwidth and artificial delays between network requests.

First, we collect a fresh snapshot of the network. We start by resolving the bootstrapping DNS seeds. We try to connect to all known peers and ask them for peers they know using `getaddr` command. We recursively repeat the process three times.

For each transaction announcement, we log the transaction hash, the IP that has announced it, and the timestamp of this event. We only log `inv` messages, and never continue with the `getdata` – `tx` exchange. For selected experiments on the Bitcoin testnet, we also log `addr` messages. Address announcements allow us to infer the set of most probable IP addresses that correspond to each transaction cluster.

We use additional Python scripts to issue series of transactions from two different nodes. For each transaction set (learning set and control set) for each experiment, we emit a series of transactions to our own newly generated addresses. We use the command line interface of the underlying full node (such as `bitcoind` for Bitcoin). To send a transaction, we use `sendtoaddress` command. We generate a new address for each transaction with `getnewaddress`. The payment amounts are generated uniformly at random from an interval between the minimum and the maximum allowed amount. The maximum amount is 10 times higher than the minimum amount. The minimum amounts for Bitcoin and Zcash are 0.00001 BTC and 0.0005 ZEC, respectively.

Experiments on the cryptocurrencies based on the Bitcoin Core codebase are implemented similarly. For Zcash, we issue both transactions that involve shielded addresses (we call such transactions shielded for short) and transactions between transparent addresses (i.e., transparent transactions). Issuing transparent transactions is similar to issuing transactions in Bitcoin. For shielded transactions, we use `z_getnewaddress` to generate a new address and `z_sendmany` to send a transaction.

We use Python scripts to process the log. We parse the log, extract the relevant information (transaction hashed and timestamps), and save the data in a more compact JSON format. We then analyze the data, perform the clustering, and visualize the results.

The number of transactions we issue per experiment is limited due to the behavior of Bitcoin Core when spending unconfirmed UTXOs. It does not allow creating chains of unconfirmed transactions of length 25 or more. Therefore, issuing more transactions within a short time frame (shorter than the block generation interval) demands creating multiple confirmed UTXOs in advance at both issuing nodes, which would require additional time and effort during the setup phase of each experiment. Splitting the transactions in multiple chains and increasing the length of the experiment would lead to difficulties in parsing and processing the logs.

3.3 Experimental evaluation

The outline of our experiment is as follows:

1. collect a fresh list of live peers;
2. establish multiple parallel connections to them;
3. launch the listening nodes and start logging `inv` and `addr` messages;
4. launch two nodes S_{learn} and $S_{control}$;
5. issue two series of transactions: the learning set from S_{learn} and the control set from $S_{control}$;
6. for each considered number N of first propagations, calculate the transaction correlation matrix;

TABLE 3.1: Experimental results of transaction clustering for Bitcoin testnet and Zcash.

Network	Listener	Anon. deg.	Servers	Avg free slots	Tx `invs`	`addrs`
Bitcoin test	California	0.83	1 141	64	139	402
Bitcoin test	Tokyo	0.80	1 128	64	193	414
Bitcoin test	Frankfurt	0.72	1 137	64	172	403
Bitcoin test	combined	0.63	1 154	63	250	1 321
Bitcoin main	Frankfurt	0.88	1 000*	25*	3 238	11 300
Zcash	Frankfurt	0.86	206	36	62	1 086

7. run the clustering algorithm with various assumed average numbers of transactions per cluster;
8. choose the best clustering by Rand score based on the “learning” set;
9. in the best clustering, assign the cluster weights proportionally to the distribution of known transactions from $S_{control}$;
10. assign zero probability of being in $S_{control}$ to transactions from S_{learn};
11. re-distribute the probability weight among transactions in each cluster;
12. calculate the final adjusted anonymity score;
13. re-arrange the clusters such that high correlation values are close to the main diagonal;
14. visualize the results.

We visualize the results with heatmaps. We assign a color to each element of the correlation matrix. A darker color at the intersection of the i-th row and j-th column represents a stronger correlation of weight vectors of i-th and j-th transactions. The heatmap is diagonally symmetric by definition: each vector is perfectly correlated with itself. We permute rows and columns such that the highly correlated elements are close to the main diagonal. We expect such permutation to reveal the block-diagonal structure of the matrix.

3.3.1 Results for desktop wallets

We evaluate our method by clustering our own transactions in Bitcoin (testnet and mainnet) and Zcash. For these experiments, we log the traffic for 15 minutes. For Dash and Monero, we run the clustering algorithm without calculating the anonymity degree. We obtain clearly visible clusters, which indicates that our approach is applicable to these cryptocurrencies as well. In the experiments on the Bitcoin mainnet, we deliberately do not attempt to occupy all connection slots and operate only on a subset of 1 000 nodes (out of approximately 10 000 nodes reachable at any given time [52]). In this section, we refer to the node that logs the incoming messages as the *listener*. Following the terminology of [37], *servers* are peers that accept incoming connections.

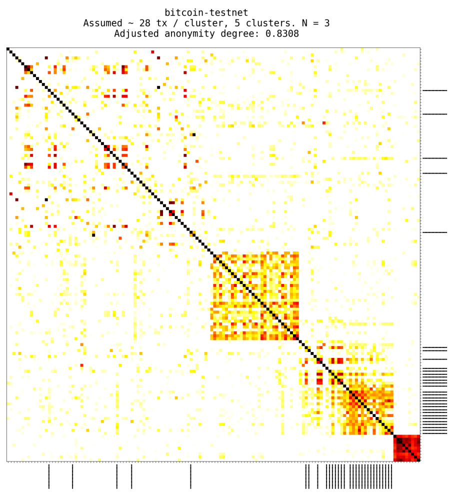


FIGURE 3.2: Transaction clustering for Bitcoin testnet (listener in California).

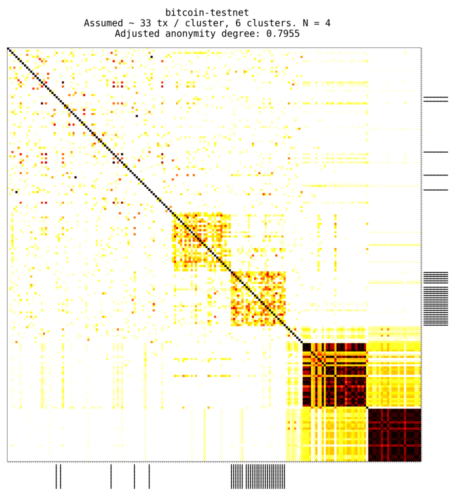


FIGURE 3.3: Transaction clustering for Bitcoin testnet (listener in Tokyo).

3.3.1.1 Bitcoin testnet

We perform four experiments on the Bitcoin testnet using listeners in different geographical locations: Frankfurt (Germany), Tokyo (Japan), and North California (the US). We conduct three experiments with each of the listeners and the fourth experiment using all listeners simultaneously. To use multiple listeners in one experiment, we distribute live peers equally among the three listeners. Each listener connects only to the peers from its chunk of the list. We then merge the log files. The fourth experiment measures the advantage an adversary gains from using geographically distributed listeners. Each listener attempts to establish 117 connections to each assigned peer.

In each experiment, we issue two sets of test transactions (the learning and the control sets) containing 30 transactions each from computers located in Luxembourg. We denote 10 transactions from the control set as "known" to estimate the anonymity degree.

The number of live peers collected by each of the listeners is close, which indicates that we obtain a complete view of the network. The number of received transactions varies little between experiments, whereas the number of `addr` messages is significantly higher in the multi-listener experiment: `addr` messages propagate through the network more slowly than transactions. The average number of available connection slots is independent of the location of the listener.

The anonymity degree calculated on our own transactions indicates a substantial loss of privacy (Table 3.1). The "*" sign in the table indicates the experiments where we only connected to a subset of available nodes. The joint experiment with three geographically distributed listeners gained the best results with an anonymity degree of 0.63. Out of the three single-listener experiments, only the Frankfurt experiment shows a lower anonymity degree. We explain it by a small geographical distance between the listener and the transaction source. The results are presented in Figures 3.2, 3.3, 3.4, and 3.5 (ticks along the axes denote our transaction from the control set).

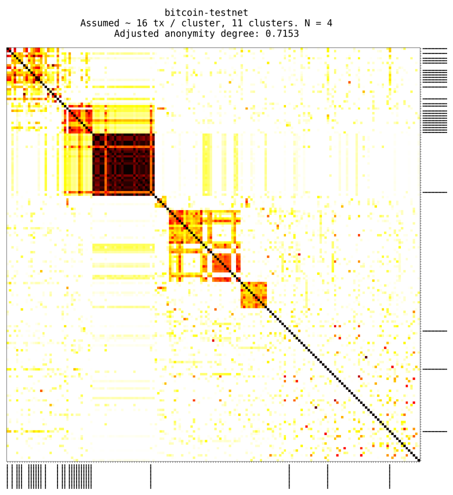


FIGURE 3.4: Transaction clustering for Bitcoin testnet (listener in Frankfurt).

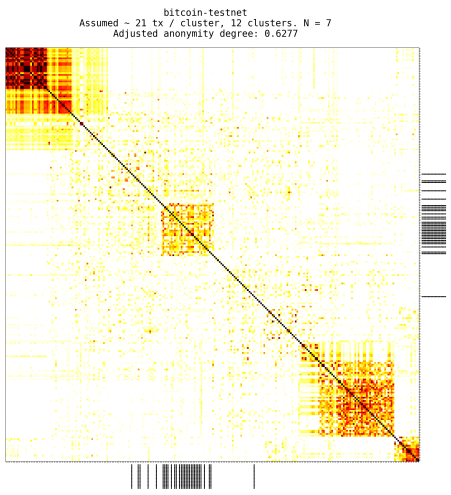


FIGURE 3.5: Transaction clustering for Bitcoin testnet (combined listeners).

Estimating the original IP We use the addr messages to determine (with some level of precision) the IP address of the transaction source. In our experiments, we first launch the listener and only then launch the issuing nodes. This allows the listener to capture the addr messages issued by the issuing nodes during bootstrapping. Address messages propagate through the network more slowly than transactions and are periodically re-broadcast. A listener can distinguish between addr messages of recently joined nodes and re-broadcasts of older addr messages. If only one or two nodes announce an IP address, we assume it is a re-broadcast. If more nodes announce an IP address, we assume that the node has just joined the network or is re-advertising its IP address.

We leverage the addr messages as follows. For each cluster, we determine the IPs of the most "important" nodes, i.e., nodes we assume might be the source or its entry nodes. For each transaction in the cluster, we sum up the weights of all IPs that have relayed it to us. We assume the top 10% of most weighted IPs to be the entry nodes. The intuition is that the entry nodes are among the first to relay addr messages from the source. Therefore, we assume that an addr message relayed by a set of IPs that substantially intersects with the entry nodes contains the IP address of the source of the cluster.

We apply this heuristic to the Bitcoin testnet experiments. We consider the clusters that mostly consist of control transactions. In three out of four experiments, the actual IP address is among the top five most "important" IPs. This result indicates that an adversary can narrow down the search for the source IP address to only a handful of IPs.

3.3.1.2 Bitcoin mainnet

We perform one experiment on the Bitcoin mainnet with a listener located in Frankfurt. The learning and control sets consist of 20 transactions each. Five transactions from the control set are assumed "known" for anonymity degree calculation.

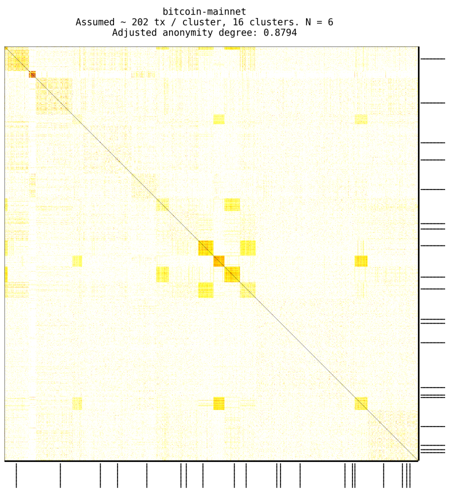


FIGURE 3.6: Transaction clustering for Bitcoin mainnet.

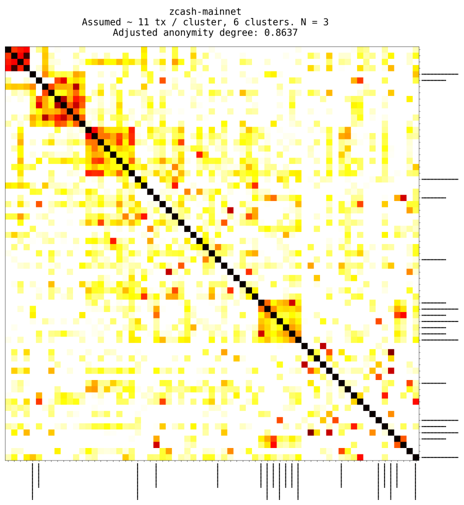


FIGURE 3.7: Transaction clustering for Zcash.

The results are presented in Table 3.1 and Figures 3.6 and 3.7. The correlation matrix also exhibits the "clustering" behavior, though the anonymity loss is smaller than for the Bitcoin testnet and Zcash. We explain these weaker results by multiple factors. First, the Bitcoin mainnet is substantially larger than other networks that we consider. More transactions in Bitcoin constitute a larger anonymity set. Second, Bitcoin nodes provide fewer connection slots and often limit the number of slots one IP can occupy. A low number of parallel connections decreases the probability that our listener learns about a new transaction quickly. Third, we only establish up to 50 connections to 1 000 peers to avoid disrupting the network, and due to resource constraints.

3.3.1.3 Zcash

Zcash is based on the Bitcoin Core codebase.[4] As of the time of our experiments (mid-2018), Zcash uses trickling for broadcast randomization, while Bitcoin uses diffusion. Zcash does not provide privacy by default: zero-knowledge proofs are used only in transactions involving *shielded* addresses [213]. Most transactions happen between *transparent* addresses and have no added privacy-preserving mechanisms compared to Bitcoin.

We perform one experiment on the Zcash mainnet with a listener located in Frankfurt. The learning and control sets consist of 20 and 18 transactions, respectively. Eight out of 18 control transactions use shielded addresses (transactions from a t-address to a z-address, also known as t-to-z). We use 6 control transactions as "known" for anonymity degree estimation.

The results are presented in Figures 3.6 and 3.7. T-to-z transactions from the control set are marked with longer ticks. Note that our method clusters transactions involving both transparent and shielded addresses.

We notice that Zcash peers offer far fewer connection slots on average: 36 compared to 64 on the Bitcoin testnet (Figures 3.8 and 3.9). Many servers only accept fewer than ten connections. These results may reflect a larger share of *protected* nodes

[4]Bitcoin Core version 0.11.2 (commit 7e27892), November 2015.

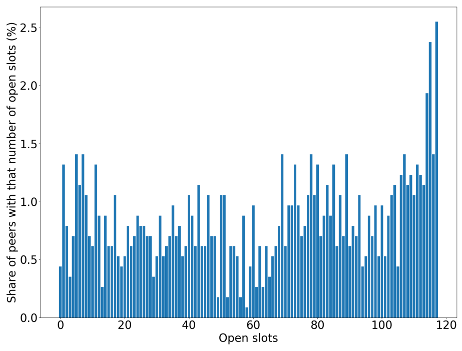


FIGURE 3.8: Free connection slots for Bitcoin testnet.

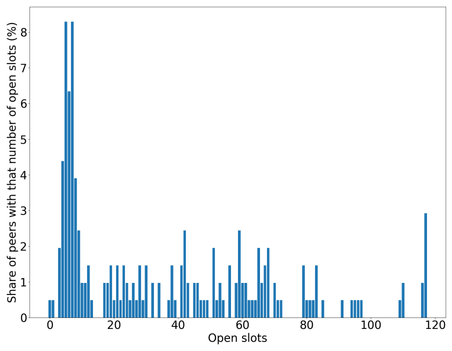


FIGURE 3.9: Free connection slots for Zcash mainnet.

in Zcash. Protected nodes use firewalls or other network-level mechanisms to limit the number of connections from each IP. However, an adversary can overcome this limitation by purchasing IP addresses from a cloud provider.

3.3.1.4 Dash

Dash is also based on the Bitcoin Core codebase and inherits the basics of its networking protocol. Dash uses diffusion for broadcast randomization. The Dash networking protocol is substantially more complex compared with Bitcoin. Dash contains 22 new message types for masternode management [339]. Masternode-related tasks include periodic pings to check whether masternodes are online, coin mixing and voting for governance proposals. Our tool does not handle Dash-specific messages.

In our experiment, we connect to 500 out of 3 065 randomly chosen Dash nodes and ask for 30 connection slots. During 15-minutes, we receive 12 transaction inventory messages and 396 Dash-specific messages.

We run the clustering algorithm twice: accounting for the Dash-specific messages (Figure 3.10), and considering only standard transaction inventory messages (Figure 3.11). In both cases, we obtain clearly visible clusters. These preliminary results show a privacy concern, especially if combined with transaction graph analysis [212].

3.3.1.5 Monero

Monero is a privacy-focused cryptocurrency not based on the Bitcoin Core codebase. Monero provides privacy by default: users do not have to explicitly choose the "private" option, contrary to Zcash (shielded transactions) and Dash (*PrivateSend*).

The Monero community recognizes the threat of deanonymization through network analysis [392, 243, 154, 76]. This has motivated the integration of Dandelion++ networking protocol in 2020 [145] (see Section 2.2.2). Moreover, the Kovri project [223] aims to add an I2P router into Monero. As of the time of our experiments (mid-2018), Monero uses no broadcast randomization.

Monero does not allow creating and spending a transaction output in the same block. A new output appears as "locked" until the transaction that creates it receives 10 confirmations (20 minutes at the target block time of 2 minutes) [126]. Though this

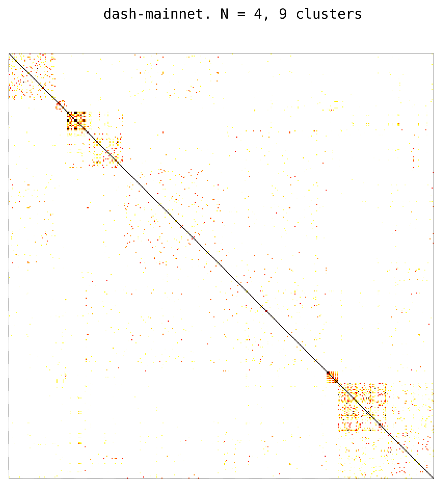


FIGURE 3.10: Transaction clustering for Dash (messages and transactions).

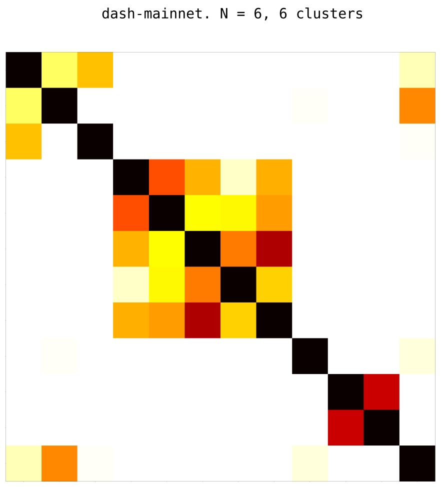


FIGURE 3.11: Transaction clustering for Dash (transactions only).

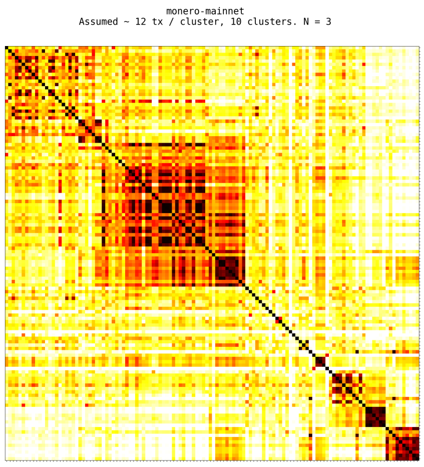


FIGURE 3.12: Transaction clustering for Monero.

is a wallet-level and not a protocol-level restriction, the official desktop wallet and Monerujo, a popular Monero wallet for Android, enforce it. Therefore, the scenario of our earlier experiments is somewhat unrealistic. For example, to issue 20 transactions within a 20 minute period, a user must have 20 "unlocked" transaction outputs (each takes 20 minutes to create).

We conduct an experiment on Monero without issuing transactions. This experiment aims to detect a block-diagonal structure in the correlation matrix derived from real-world transactions. We connect to 200 nodes and receive 124 transactions in a 38 minute window (Figure 3.12).

We explain a less clear picture compared to the Bitcoin testnet as follows. First, we connect to only 200 out of 1 700–1 800 nodes (as of the time of the experiment [262]). Second, Monero does not use the three-step transaction propagation (`inv` – `getdata`

– `tx`), which may have opposite effects on clustering quality. On the one hand, relaying a transaction unconditionally to all neighbors increases the probability that we receive a new transaction from a node other than the source of one of its entry nodes. On the other hand, this broadcast type may provide a near-perfect insight into transaction sources, if the attacker connects to all or nearly all nodes. Third, `monerod` connects to nodes slower than `bcclient`, which impedes data collection. In our experiment, while trying to connect to 200 nodes, we obtain 150 connections after approximately 2 hours, 175 connections after 3 hours, 200 connections after nearly 8 hours.

Monero uses hard-coded DNS seeds and seed IP addresses for bootstrapping (see Chapter 2.2.1). As of July 2018, all DNS seeds fail to resolve. The official client falls back to seed IP addresses.

3.3.2 Results for mobile wallets

We perform analogous experiments on Bitcoin (testnet and mainnet) and Zcash issuing transactions from selected mobile wallets for Android. Most mobile wallets connect to the P2P network via a trusted server maintained by the wallet's developers. We refer to wallets that connect to the P2P network directly as *P2P wallets*. Most P2P wallets rely on the BitcoinJ library for networking. None of them uses broadcast randomization. Unlike Bitcoin Core, BitcoinJ sends `tx` unconditionally.[5] The BitcoinJ developers acknowledge that the three-step `inv` – `getdata` – `tx` exchange in Bitcoin Core improves privacy, but argue that since SPV nodes have a weaker privacy model, the three-step broadcast would only decrease efficiency.

We choose a diverse selection of mobile wallets for our experiments: Bitcoin-only and multi-coin, with centralized and P2P networking. We perform experiments on the Bitcoin testnet (Bitcoin wallet), the Bitcoin mainnet (Bitcoin wallet, BRD, Coinomi, Mycelium), and Zcash (Coinomi). Bitcoin wallet and BRD[6] use P2P broadcast. Coinomi and Mycelium use centralized broadcast. For wallets with centralized broadcast, we only issue one set of transactions, using it as a "label" for a presumed wallet cluster. If our transactions form a visible cluster, we inspect the IP addresses of nodes among the first ones to broadcast them. Thus we infer the IP addresses of nodes likely to be used for transaction broadcasts for this wallet. This information allows us to associate subsequent transactions with popular wallets.

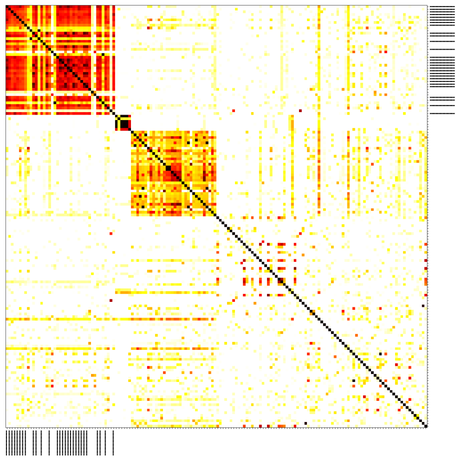
The correlation matrices exhibit a block-diagonal structure (Figure 3.13), as expected. The clusters are clearly visible for the Bitcoin wallet on testnet, for which we obtained the anonymity degree of 0.5089. The adjusted anonymity degree for the Bitcoin mainnet is 0.8646 for Bitcoin wallet, 0.8413 for BRD, and 0.9117 for Coinomi. Similar to the results with a desktop node as a transaction source, the picture for the Bitcoin mainnet is much less clear because of more transactions and nodes overall and a smaller number of connections that we establish.

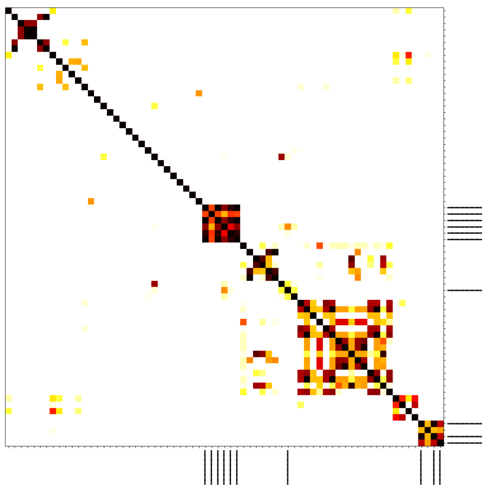
Estimating the IP addresses of wallet's nodes

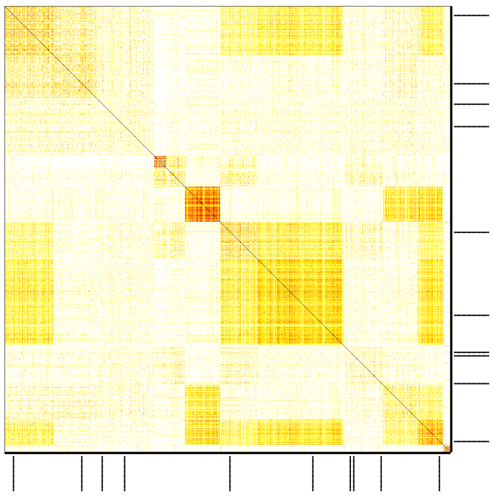
Apart from clustering transactions, an adversary might be interested in obtaining the IP address of the nodes that a centralized wallet uses for transaction broadcast. The

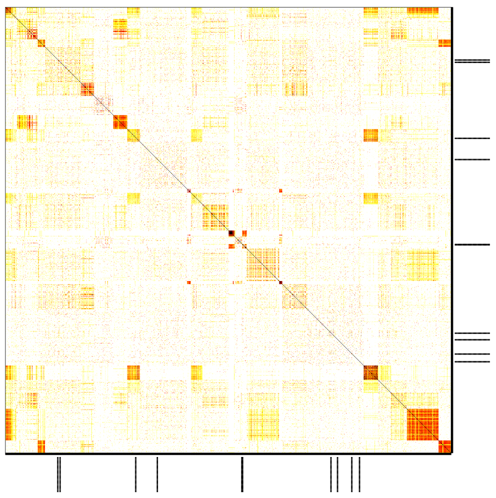
[5]Note that in this case a full node receiving a `tx` message from an SPV node can be sure that the transaction originates at that SPV node, whereas receiving an `inv` announcement from a full node may also be a re-broadcast. This demonstrates the privacy enhancement of exclusively connecting to a trusted node for SPV.

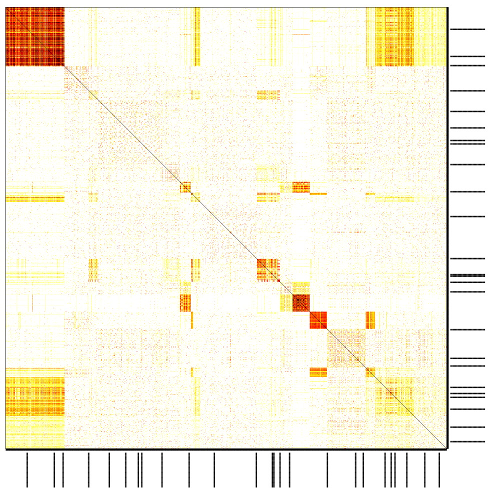
[6]Bitcoin wallet and BRD are the two most popular Bitcoin clients [403].


(A) Bitcoin testnet, Mycelium.


(B) Bitcoin testnet, Bitcoin wallet.


(C) Bitcoin mainnet, Bitcoin wallet.


(D) Bitcoin mainnet, BRD.


(E) Bitcoin mainnet, Coinomi.

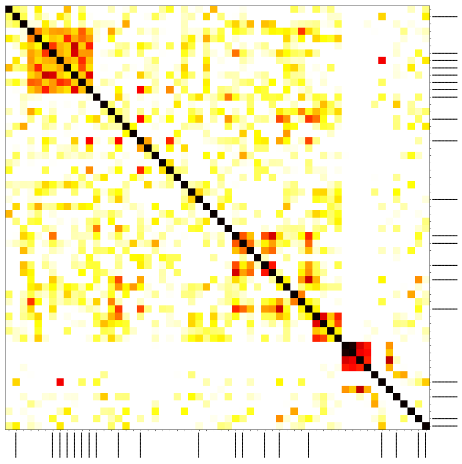

(F) Zcash, Coinomi.

FIGURE 3.13: Transaction clustering for mobile wallets.

IP address of a centralized wallet's node may not necessarily be secret. Still, linking Bitcoin transactions with IP addresses may reveal which wallet a victim is using. The adversary can later leverage this information for social engineering attacks.

We test this attack scenario in two experiments (Figure 3.13). In the first experiment, we consider the Bitcoin testnet and Mycelium wallet. The Mycelium transactions exhibit a clearly visible cluster: they are quickly announced from the same two IP addresses. The time difference between these announcements is in single milliseconds. Other nodes re-broadcast then only after tens or hundreds of milliseconds. According to IP geolocation services, the two nodes are located in Germany (`2a01:4f9:2b:4ca::2`) and in Helsinki, Finland (`95.216.68.181`). A reverse DNS lookup service `robtex.com` suggests that one of these IP addresses corresponds to a URL `electrumx-b.mycelium.com`. Both IP addresses belong to Hetzner (a cloud provider) and host Bitcoin nodes with a latency of 25 ms [52]. We estimate that each of these nodes offers more than 700 connection slots in a separate experiment. The second experiment considers Zcash and Coinomi wallet. Though the Coinomi transactions do not form a clear cluster, we observe that some of them (in the second cluster) are quickly announced from the same IP (`5.79.123.194`), which, we assume, is one of Coinomi's nodes.

Comparison with prior work

Multiple research works propose deanonymization techniques for cryptocurrencies. However, few of them quantify their results based on ground truth data, i.e., by deanonymizing transactions known to have originated from one source. For instance, one of the earliest papers [254] clusters Bitcoin transactions using transaction graph heuristics and known deanonymized addresses as a starting point. However, it does not provide a way to verify how many of the addresses assigned, for example, to the cluster of an exchange, actually belong to it.

The same is mostly true for related work on network analysis. As outlined in Section 2.4.1, the most closely related prior work is [222, 37, 279]. In [222], the first relayer heuristic is used. However, after refining the results to eliminate likely false positives, the authors conclude that most results are based on anomalous propagation and have limited applicability.

In [279], two clustering methods are compared: based on transaction graph analysis and networking analysis (first relayer heuristic with optimizations). The authors find little correlation between the two types of clusters. Less than 8% of clusters from transaction graph analysis correspond to a single IP address. No evaluation based on the ground truth has been performed.

Only in [37] do the authors estimate the success using their own transactions, in addition to theoretical calculations. They achieve a success rate of nearly 60% on the Bitcoin testnet, over a set of 424 transactions.[7] This means that 60% of the testnet transaction issued from a given entry set are correctly identified as such.

Our method seems more promising since it tries to use all the available information, including timing. In the experiment on the Bitcoin testnet (Figure 3.5), 24 out of 30 control set transaction are assigned to one cluster. Additionally, 7 transactions

[7]The experiments described in [37] were conducted in 2014. Since then, Bitcoin Core implemented stricter rules on chains of unconfirmed transactions. The length of such chains is now restricted to 25 by default [103, 266, 247]. This limits our ability to emit a large number of transactions during one experiment. This issue does not apply to Zcash, as its source code was forked from Bitcoin Core before the relevant changes were introduced. However, the limitation in the Zcash experiments is the time required to generate a shielded transaction (around 30 seconds on a consumer laptop as of 2018). As of 2020, after a series of protocol optimizations, generating a shielded Zcash transaction takes 2-3 seconds.

not from the control set have been put into this cluster. Therefore, we achieve a false positive rate of 23% and a false negative rate of 20%. This corresponds to the success rate of 80%, which is higher than 60% reported in [37].

We should note however that while our method is promising, more statistical evidence is needed to quantify its advantage over previous methods precisely.

3.4 Attack cost estimation

We now estimate the resources required for a full-scale attack on the Bitcoin mainnet. The Bitcoin mainnet consists of approximately 10 000 nodes reachable at any given time [52]. Bitcoin nodes provide 43 connections slots on average (measured on 1 000 random nodes). The size of an `inv` message is "36x + const for message with x objects" [48]. We assume that an `inv` for a single transaction requires 40 bytes. Bitcoin processes around 250 000 transactions per day (as of November 2018), or 2.89 transactions per second. Assuming each connection eventually relays each transaction, we arrive at the required bandwidth for one connection slot: $2.89 \times 40 = 115.6$ B/s. A full-scale attack on the Bitcoin mainnet would require maintaining an average of 43 connections to 10 000 nodes, i.e., a total bandwidth of $115.6 \times 10\,000 \times 43 = 49\,708\,000$ B/s = 47.4 MB/s = 379 Mbit/s. An hour-long attack at this bandwidth will require receiving approximately 167 GB of incoming traffic.

We estimate the attack cost based on the cost of running a full Bitcoin node. Various estimations put that cost between 3 and 20 US dollars per month [416, 95]. By default, Bitcoin nodes relay transactions through 8 outgoing connections and accepts up to 117 incoming connections.[8] Assuming an average node has a total of 125 connection slots, $125 - 43 = 82$ slots eventually get occupied. An adversary needs to maintain $10\,000 \times 43 = 430\,000$ connections, or approximately 5 244 times more than a regular node. Considering that one month (30-days) is 720 hours, we conclude that an estimated cost of an hour-long attack is approximately $5244 \div 720 = 7.3$ times the monthly cost of running a regular full node. That leads to an estimation of bandwidth costs between 20 and 150 USD. The total cost of the attack is on the order of hundreds of US dollars (taking into account the cost of computation and storage). We conclude that the attack is well within reach of even low-budget adversaries. All our experiments on the Bitcoin testnet and Zcash mainnet cost 35 USD, which can likely be decreased by optimizing the scripts and storing data locally.

3.5 Discussion and countermeasures

Our technique performs well on relatively small networks (Bitcoin testnet, Zcash) and works to some extent on Bitcoin. We expect a resourceful attacker to achieve better results on the Bitcoin mainnet by establishing more connections.

Application-level cryptographic countermeasures, such as zero-knowledge proofs in Zcash, cannot defend against our attack. We only consider transaction hashes and their announcement times, ignoring their content. Overlay networks such as Tor [381] are a popular mitigation for deanonymization attacks. In our case, transactions announced from the same Bitcoin node would form a cluster, even if they are sent to this node through Tor. Moreover, broadcasting transactions via Tor may introduce man-in-the-middle vulnerabilities [40].

[8]The experiments were performed before Bitcoin Core introduced two more connections for block propagation. In any case, blocks are outside of the scope of our technique [104].

Our method's main limitation is the assumption that a user issues multiple transactions during a relatively short time frame through the same set of entry nodes (i.e., the same session). Transactions issued from different sessions would not be linkable by our technique.

Practical countermeasures against transaction clustering depend on the wallet type. For a full node that accepts incoming connections (a server):

- run the node with more outgoing connections to dilute the quality of the topological fingerprint;
- introduce random delays on top of those implemented in the node software;
- drop connections to randomly chosen entry nodes and establish new ones, constantly altering the set of entry nodes;
- advise users not to broadcast sensitive transactions within a short period (if the node is used for broadcasting users' transactions).

Users of full nodes without incoming connections (e.g., behind NAT) may wish to re-launch the software to issue each transaction through a new set of entry nodes.

Proposed countermeasures for SPV nodes would be:

- use wallets with P2P broadcast (e.g., Bitcoin wallet for Android [44]);
- if using wallets with centralized broadcast, use different wallets for transactions not meant to be linkable;
- connect to a trusted full node.

All users should avoid sending multiple transactions within a short time frame.

Note that an attacker may leverage external information to increase clustering accuracy, such as known addresses of exchanges and other service providers [400].

Dandelion and Erlay The key property of the currently used cryptocurrency P2P protocols that we exploit is that nodes do not distinguish between incoming and outgoing connections. A node announces transactions to a random subset of connections. This allows a well-connected listener to receive more information by initiating more connections. For instance, by saturating 50% of a node's connection slots, a listener has a 50% chance to be the first to receive a new transaction from it.

Dandelion [160] is a P2P protocol for cryptocurrencies (see Section 2.2.2). It is an effective countermeasure against our attack.[9] In Dandelion, nodes choose neighbors for the stem phase from their outgoing connections only. An attacker has no easy way to force a remote peer to initiate a connection. Therefore, a malicious node with many outgoing connections does not have an advantage in the stem phase. It can only aggregate incoming information while acting as a regular relay, gaining some but not much insight into possible transaction clusters. The same applies to Erlay [277] – a P2P protocol for a more efficient transaction broadcast in Bitcoin.

[9]The authors mention ([160], Section 4.2) that some configurations of the protocol may be prone to transaction correlation attacks.

3.6 Conclusion

We have studied the state of anonymity of cryptocurrencies on the network level. We have described and implemented a novel transaction clustering method based on the analysis of transaction announcements. We have implemented and tested our technique on four popular cryptocurrencies using a variety of wallets.

Our results show that Bitcoin and the major privacy-focused cryptocurrencies do not sufficiently defend against network-based transaction clustering. A low budget adversary can accurately link transactions that originate from the same node. A similar technique allows an attacker to infer the IP addresses of nodes used for transaction broadcast by mobile wallets. Cryptocurrencies should defend against network analysis to provide stronger privacy guarantees.

Chapter 4

Privacy of cryptocurrency wallets

Smartphones have become the primary computing device for many millions of people and play an increasingly important role in the cryptocurrency ecosystem. In this Chapter, we study the privacy of mobile cryptocurrency wallets for Android – the most prevalent mobile operating system.[1] We systematize the privacy-related characteristics of popular wallets and study 23 selected wallets in more detail. We compare their privacy-related properties and analyze their source code, both manually and using static analysis tools. Our results show that most mobile wallets do not follow privacy guidelines. Privacy-conscious users are quite limited in choosing a mobile wallet for Bitcoin, and even more so – for privacy-focused cryptocurrencies.

4.1 Minimal privacy criteria

We argue that the following privacy criteria are minimally necessary for a mobile wallet:

1. No registration required. Otherwise, the wallet provider can link together and deanonymize user's transactions.
2. Open-sourced code. A malicious closed-source application can track users or steal their funds. Open-sourced code decreases the risk of backdoors or other unintended functionality.
3. Private keys generated and stored locally. Otherwise, the server that stores the keys requires full trust.
4. Direct connection to the P2P network. The wallet should query blockchain data and broadcast transactions to peers directly and not through a server. Otherwise, the server can deanonymize and censor transactions.

We check the first criterion by attempting to generate a receiving address in the wallet without registering or providing any personal information. If we succeed, we proceed to check the next criteria. We consider the second criterion met if we find a publicly available code repository with a fully-fledged Android application that the wallet's official website links to.[2] We check the third and the fourth criteria based on the official documentation and the source code.

We compile a list of wallets recommended on the official websites of the considered cryptocurrencies: Bitcoin, Dash, Monero, and Zcash. We also consider the most popular wallets (based on the publicly available approximate number of Play Store

[1]This Chapter is based on [42].

[2]Strictly speaking, *reproducible builds* are required to assure that the file in the Play Store is not modified compared to the repository. We do not check for build reproducibility in this work.

TABLE 4.1: Minimal privacy criteria for selected wallets.

Wallet	Coin support				No registration	Open-source	Local private keys	P2P networking
	Bitcoin	Dash	Monero	Zcash				
Abra	+	+	+	+	-	-	+	?
Airbitz	+	-	-	-	-	+	+	?
ArcBit	+	-	-	-	+	+	+	-
Bitcoin.com	+	-	-	-	+	+	+	-
Bitcoin wallet	+	-	-	-	+	+	+	+
Bither	+	-	-	-	+	-	+	+
BTC.com	+	-	-	-	-	+	+	-
BRD	+	-	-	-	+	+	+	+
Coin.space	+	-	-	-	+	+	+	-
Coinomi	+	+	-	+	+	+	+	-
Copay	+	-	-	-	+	+	+	-
Dash wallet	-	+	-	-	+	+	+	+
Edge	+	+	+	-	-	-	+	-
Electrum	+	-	-	-	+	+	+	+
Ethos	+	+	+	+	-	-	?	?
GreenBits	+	-	-	-	+	+	+	-
Jaxx	+	+	-	+	+	-	+	-
Mobi.me	+	+	+	+	-	-	?	?
Monerujo	-	-	+	-	+	+	+	+
Mycelium	+	-	-	-	+	+	+	-
Samourai	+	-	-	-	+	+	+	+*
Simple Bitcoin	+	-	-	-	+	+	+	+
Zelcore	+	-	-	+	-	-	+	-

downloads). The following wallets satisfy these criteria (Bitcoin unless specified): Bitcoin wallet, Bither, BRD, Dash wallet (Dash), Electrum,[3] Monerujo (Monero), Simple Bitcoin (marked in bold in Table 4.1). No multi-currency wallets and no Zcash wallets satisfy these criteria.

4.2 Analysis of selected wallets

The following wallets have passed the minimal privacy criteria: Bitcoin Wallet, Bither, BRD, Dash Wallet, Electrum, Monerujo, and Simple Bitcoin. We add the Samourai

[3]Electrum is originally a desktop application; the official GitHub repository gives instructions on how to generate an APK file using Kivy GUI. Electrum wallet relies on an independent network of nodes (Electrum servers) to receive blockchain data and broadcast transactions. Though Electrum servers are not genuine cryptocurrency P2P nodes, we consider Electrum satisfy the P2P criteria, as a user can technically choose which Electrum servers to connect to, including their own trusted server.

TABLE 4.2: Alternative installation methods of selected wallets.

	Bitcoin Wallet	Bither	BRD	Dash wallet	Electrum	Monerujo	Simple Bitcoin	Bitcoin.com	Mycelium	Coinomi	Jaxx	Copay	Airbitz	Samourai
F-Droid	+	-	-	-	-	+	+	-	-	-	-	-	-	-
APK	+	+	-	-	+	+	+	+	+	+	-	-	-	-

wallet to our list due to its heavy emphasis on privacy.[4] We now study these wallets in more detail.

4.2.1 Manual inspection

Independent installation Users usually install Android applications from the Play Store, which requires a Google account. A privacy-conscious user may want to avoid linking their cryptocurrency activity with their Google profile. They may use F-Droid (an independent application store for free and open-source applications [156]) or install the application manually from an APK file. Out of the seven wallets we considered, three are available in F-Droid, and five can be installed from an APK file (see Table 4.2).

Permissions Android *permissions* restrict access to sensitive user information and device functionality. Application developers declare the necessary permissions in the application's *manifest file*. The user grants permissions at installation time or run-time.[5] Permissions that give access to critical functionality or personal information are referred to as *dangerous* [10].

Wallets vary in the number and importance of permissions they require (Table 4.3, dangerous permissions in bold). Airbitz requires the highest number of permissions (15). Two applications require access to coarse and fine location (Airbitz, BRD) and sending SMS (BRD, Samourai). Electrum requests the lowest number of permissions – four (three of them dangerous). All wallets require at least one dangerous permission – camera access, required to scan cryptocurrency addresses presented as QR codes.

Privacy policies Google Play store rules prescribe all Android applications that handle "personal or sensitive user data" to declare a privacy policy. We compare the privacy policies of the selected wallets (Table 4.4).

Privacy policies of some wallets (Bitcoin wallet [45], Dash wallet [112], Bither [49], Monerujo [263]) are relatively concise and only justify the use of some of the required permissions. Privacy policies of BRD [63] and Electrum [137] are more elaborate. Bither privacy policy provides the rationale behind requiring permission to record audio: it allows for "collecting ambience entropy for XRandom." BRD uses cookies, trackers, and third-party providers for analytics: Google Analytics and Firebase. These tools allow wallet developers to analyze crash reports and collect application usage patterns. Such data may be used to link users' activity with unique identifiers

[4]Strictly speaking, Samourai did not pass our initial test: it connects to a remote node via RPC, not P2P, and requires full control over it [389]. We mark it with "+*" in Table 4.1.

[5]Starting from Android 6.0.

TABLE 4.3: Permissions of selected wallets.

	Bitcoin Wallet	Bither	BRD	Dash wallet	Electrum	Monerujo	Simple Bitcoin	Bitcoin.com	Mycelium	Coinomi	Jaxx	Copay	Airbitz	Samourai
read storage	+	+	+	+	+	+	+	+	+	+	+	+	+	+
modify storage	-	+	+	+	+	+	+	+	+	+	+	+	+	+
take pictures	+	+	+	+	+	+	+	+	+	+	+	+	+	+
coarse location	-	-	+	-	-	-	-	-	+	-	-	-	+	-
fine location	-	-	+	-	-	-	-	-	-	-	-	-	+	-
send SMS	-	-	+	-	-	-	-	-	-	-	-	-	-	+
view connections	+	+	+	+	-	-	-	+	+	+	+	+	+	+
Bluetooth	+	+	-	+	-	-	-	-	-	+	-	-	+	-
full network access	+	+	+	+	+	+	+	+	+	+	+	+	+	+
control NFC	+	-	-	+	-	+	-	-	+	+	-	-	+	-
run at startup	+	+	-	+	-	-	-	-	-	+	-	-	+	+
control vibration	+	+	-	+	-	-	+	-	+	+	-	+	-	+
prevent sleeping	+	+	+	+	-	+	-	+	+	+	-	+	+	-
upd component usg	-	-	+	-	-	-	-	-	-	-	-	-	-	-
receive data	-	-	+	-	-	-	-	+	+	+	-	+	-	-
background work	-	-	+	-	-	-	-	-	-	-	-	-	-	-
display settings	-	-	+	-	-	-	-	-	-	-	-	-	-	-
disable screen lock	-	-	+	-	-	-	-	-	-	-	-	-	-	-
retrieve running apps	-	+	-	-	-	-	-	-	-	-	-	-	-	-
record audio	-	+	-	-	-	-	-	-	-	-	-	-	-	-
view Wi-Fi	-	+	-	-	-	-	-	-	-	-	-	-	-	-
send sticky broadcast	-	+	-	-	-	-	-	-	-	-	-	-	-	-
read phone status	-	-	-	-	-	-	-	+	+	-	-	+	-	+
read service config	-	-	-	-	-	-	-	+	-	-	-	-	-	-
license check	-	-	-	-	-	-	-	-	+	-	-	-	-	-
find accounts	-	-	-	-	-	-	-	-	-	-	-	-	+	-
read contact card	-	-	-	-	-	-	-	-	-	-	-	-	+	-
Bluetooth settings	-	-	-	-	-	-	-	-	-	-	-	-	+	-
use accounts	-	-	-	-	-	-	-	-	-	-	-	-	+	-
receive SMS	-	-	-	-	-	-	-	-	-	-	-	-	-	+
reroute out calls	-	-	-	-	-	-	-	-	-	-	-	-	-	+
Total permissions	9	13	14	10	4	6	5	9	12	11	5	9	15	11

TABLE 4.4: Privacy policies of selected wallets: information that the developers may obtain.

	Bitcoin Wallet	Bither	BRD	Dash wallet	Electrum	Monerujo	Simple Bitcoin	Bitcoin.com	Mycelium	Coinomi	Jaxx	Copay	Airbitz	Samourai
IP address	-	-	+	-	+	(+)	?	-	?	+	-	+	+	+*
browser version	-	-	+	-	-	(+)	?	-	?	+	-	?	+	+
pages visited	-	-	+	-	-	(+)	?	-	?	+	-	?	+	+
time of visit	-	-	+	-	+	(+)	?	-	?	+	-	?	+	+
unique device ID	-	-	+	-	-	(+)	?	-	?	-	-	?	?	-
other diagnostics	-	-	+	-	+	(+)	?	-	?	+	+	?	+	+
type of device	-	-	+	-	-	(+)	?	-	?	-	-	?	+	+
OS type	-	-	+	-	+	(+)	?	-	?	+	-	?	+	+
location	-	-	+	-	-	(+)	?	-	?	+	-	?	-	-
device name	-	-	-	-	+	-	?	-	?	-	-	?	?	-
app configuration	-	-	-	-	+	-	?	-	?	-	-	?	-	-
pages visited before	-	-	-	-	-	(+)	?	-	?	+	-	?	-	-
browser plug-ins	-	-	-	-	-	(+)	?	-	?	-	-	?	-	-
time zone	-	-	-	-	-	(+)	?	-	?	-	-	?	-	-
"clickstream"	-	-	-	-	-	(+)	?	-	?	-	-	?	-	-
cookies	-	-	+	-	-	-	?	-	?	-	+	?	+	+
analytics	-	-	+	-	+	-	?	-	-	-	-	?	+	+

of their devices. Simple Bitcoin has no privacy policy. The link from Google Play refers to the wallet's official website [349], which does not specify whether the application collects, stores, or transmits the users' data. Monerujo privacy policy notes that the application uses the exchange rates from the public API of `kraken.com`, and an exchange service `xmr.to`, which are subjects to their privacy policies (marked with (+) in Table 4.4). Mycelium transmits a report to the developers' server in case of a crash. The developers claim they "took care that it does not contain unnecessary privacy relevant information." The Samourai wallet collects users' IP addresses "with replaced last byte," which can hardly be considered anonymization: one may still infer the approximate location from the first three bytes of the IP address (marked with "+*" in Table 4.4). Airbitz broadcasts transactions through Electrum servers; a user may choose a server. In many cases, the link to the privacy policy from the wallet's Play Store page leads to the privacy policy of the corresponding *website*, not the *wallet*.

Networking All wallets with P2P transaction broadcasting except Monerujo use hard-coded DNS seeds to bootstrap. Simple Bitcoin adds one random node from a hard-coded list to a list of peers obtained via bootstrapping.[6] Electrum connects to two random servers from a hard-coded list of 52 servers. It requests the transaction history from a single server and checks it against block headers sent by other servers. Monerujo lets the user either choose from three hard-coded URLs that resolve to a

[6] `5.9.104.252`, `213.133.103.56`, `213.133.99.89`; all unreachable as of 2018.

TABLE 4.5: Networking characteristics of selected wallets.

	Bitcoin Wallet	Bither	BRD	Dash wallet	Electrum	Monerujo	Simple Bitcoin	Bitcoin.com	Mycelium	Coinomi	Jaxx	Copay	Airbitz	Samourai
Trusted node	+	-	-	+	+	+	+	-	-	-	-	-	+	+
F-Droid download	+	-	-	-	-	+	+	-	-	-	-	-	-	-
APK download	+	+	-	-	+	+	+	+	+	+	-	-	-	-
Uses BitcoinJ	+	-	-	+	-	-	+	-	+	+	-	-	-	+
Net monitor	+	+	±	+	±	+	-							
Connections	4-6	6	3	4-6	2	1	10							

list of publicly available nodes [7] or provide the credentials for connecting to a custom node.

We tried to connect Bitcoin wallets with P2P broadcast to our own full node. Bitcoin wallet and Simple Bitcoin did connect,[8] BRD did not. Bither did not provide this option.

Wallets with P2P networking have a *network monitor* that displays the IP addresses of connected nodes. BRD shows only the IP of the "primary node" (without specifying what it means). Electrum shows the address of the server used to query the transaction history (other servers are used to check it). Simple Bitcoin only shows the number of connected peers. The number of established connections varies from wallet to wallet.

The summary of the networking characteristics is presented in Table 4.5.

4.2.2 Static analysis

We analyzed the source code of the selected wallets using two tools: FlowDroid and SmartDec Scanner.

FlowDroid

FlowDroid [15] is an open-source static analysis tool.[9] It uses data flow analysis to detect execution paths that transfer data from *sources* (functions that may return sensitive data) into *sinks* (functions that send data elsewhere). FlowDroid fails to scan Bitcoin wallet, Bither, and Monerujo (stopped after a 2 hour timeout). Samourai has not been scanned because of the unavailability of the APK file.[10] FlowDroid detected potential data leaks in 8 out of 14 applications (see Table 4.6).

SmartDec Scanner

We scan the wallets with a proprietary static analysis tool SmartDec Scanner [354]. We manually inspect the results and summarize the most prevalent privacy-related

[7] `node.moneroworld.com:18089`, `node.xmrbackb.one`, `node.xmr.be`

[8] Version strings: `/bitcoinj:0.14.7/Bitcoin Wallet:6.29/`, `/bitcoinj:0.14.4/Bitcoin:1.075/`.

[9] The source code is available at [166].

[10] The application was marked as "unreleased" in the Play Store at the time of the experiment, which prevented us from obtaining the APK.

issues, in a roughly decreasing order of potential threat. Note that these issues do not directly lead to exploits.

Leak to external storage Android provides internal and external storage.[11] An application can only access its own directory in the internal storage. External storage is available to other applications.

Sensitive data should only be kept in internal storage. Android automatically backs up data and settings of applications that did not opt out.[12] Automatic backups should be disabled for privacy-critical applications.

XSS attacks via Javascript in WebView One method of developing dynamic user interfaces on Android is using JavaScript inside a `WebView` – an Android component that displays web pages. By default, execution of JavaScript code in `WebView` is disabled, but a developer can override this setting (`setJavaScriptEnabled(true)`). Executing malicious JavaScript code may lead to cross-site scripting (XSS) and other attacks.[13] We detect two instances of this issue in BRD (in `FragmentSupport` and `WebViewActivity` classes). In both cases, the warning from Android Lint – a static analyzer built into the standard Android development environment – is suppressed.

Insecure connection In Java, the `X509TrustManager` class specifies the parameters of a TLS connection. A developer can override its methods to accept all certificates. Accepting certificates that are not authenticated by a chain of signatures up to a trusted root CA may lead to a man-in-the-middle attack. Connections between Electrum servers, unlike the Bitcoin protocol, are encrypted and authenticated with TLS. Bitcoin wallet and Dash wallet, in addition to their respective P2P protocols, use Electrum servers for querying the balance when sweeping a paper wallet. Bither defines HTTP URLs of its own API (`bither.net`) and a block explorer `blockchain.info` (class `BitherUrl`), which can lead to displaying incorrect balances or fake transactions in case of a man-in-the-middle attack. Simple Bitcoin uses five hard-coded URLs to query current fees. One of them[14] uses an unencrypted connection. A man-in-the-middle attack of a fee estimator may lead to a denial of service attack (transactions with low fees may never be confirmed), though this scenario requires four other APIs served over HTTPS to fail simultaneously.

Leak into logs Each Android application can write to its log. Applications can also read their logs with the `READ_LOGS` permission. Before Android 4.0, this permission also granted access to logs of other applications.

It is possible to access logs on a rooted device or using developer tools. All wallets log details about their operation, including error messages, which may include sensitive data (e.g., the IP address of a trusted node). This issue is present in all wallets, as all wallets use logging in exception handlers. One may find all occurrences of this issue by searching for the methods of the `Log` class, `print`, `println`, and exception handlers with `ex.printStackTrace()`. Further investigation is needed to determine the impact and probability of data leaks.

[11] Historically, external storage was assumed to be on a removable memory card, which now may not be the case.

[12] By setting `android:allowBackup=“false”` in the Manifest file.

[13] Even trusted code, e.g., from the application's resources, may contain unintended side effects or bugs, and implicitly leak information.

[14] `http://api.blockcypher.com/v1/btc/main`.

TABLE 4.6: Static analysis of selected wallets.

	Bitcoin Wallet	Bither	BRD	Dash wallet	Electrum	Monerujo	Simple Bitcoin	Bitcoin.com	Mycelium	Coinomi	Jaxx	Copay	Airbitz	Samourai
Leaks (FlowDroid)	0	4	3	1	0	2	1	0	6	4	0	0	4	?
Leak to ext. storage	-	-	-	-	+	-	+	-	-	-	+	+	+	-
XSS WebView	-	-	+	-	-	-	-	+	-	+	+	+	+	-
Insecure conn.	+	+	-	+	-	-	+	-	-	-	-	-	-	-
Leak into logs	+	+	+	+	+	+	+	+	+	+	+	+	+	+

The summary of the results obtained with static analysis (both FlowDroid and SmartDec Scanner) is presented in Table 4.6.

4.3 Conclusion

We have studied Android wallets for Bitcoin and the major privacy-focused cryptocurrencies. Most wallets do not satisfy our minimal privacy criteria. Many wallets obtain dangerous permissions and potentially leak users' private information. Static analysis reveals defects in their source code.

Secure development practices are especially relevant for application targeting mobile devices, which store lots of personal data and can be easily lost or stolen. Mobile developers should require as few permissions as possible, open-source the code, provide alternative installation methods (F-Droid, direct APK download), and implement other privacy-preserving measures to protect their users' privacy.

Part II

Privacy of the Lightning Network

Chapter 5

Introduction to Lightning Network

Bitcoin's security model assumes that every node must be able to validate every transaction, which severely limits transaction throughput. *Layer-two*, or *off-chain* protocols provide a solution. They allow the participants to exchange value without broadcasting every transaction to the blockchain. Conflicts can be resolved on the blockchain, preserving some of its security guarantees. Moving most transactions off the blockchain increases the throughput without modifying the base protocol (also referred to as *layer-one* in this context).

Most Bitcoin-based layer-two protocols implement on the concept of *payment channels*. A *payment channel* is a protocol for off-chain payments.[1] It allows two parties to continuously update the distribution of the initially committed funds. This Chapter provides the background on the evolution of payment channels and the most prominent implementation of this idea – the Lightning Network.

5.1 Evolution of payment channels in Bitcoin

To put our work in a historical context, we now describe the evolution of Bitcoin-based payment channels.[2]

5.1.1 Transaction replacement with sequence numbers

The first protocol for re-negotiating unconfirmed transactions, proposed by Satoshi Nakamoto [190], uses two transaction fields: sequence number (`nSequence`) and timelock (`nLockTime`). `nSequence` acts as a counter. `nLockTime` mandates a time before which a transaction cannot be included in a block. In the transaction replacement protocol, two parties sign a series of transactions with increasing sequence numbers and a timelock set to a point in the future. The final state is confirmed after the timelock expires. The major drawback of this approach is that sequence numbers are not enforceable. Miners have no economic incentives to prioritize a transaction with a higher sequence number if it offers a lower fee than a conflicting transaction with a lower sequence number. They even enjoy a degree of plausible deniability: they may claim that they have not heard of the later transaction versions, which may happen without malicious intent due to delays in the P2P network.

[1]We follow the terminology of [13]: a transaction is a data structure that records the transfer of control over bitcoins, whereas a payment is a process of moving value across one or multiple LN channels. Each LN payment implies signing and exchanging multiple Bitcoin transactions.

[2]See [253] for an overview of Bitcoin payment channel designs.

5.1.2 Unidirectional channels

Spillman channels, introduced in 2013, is the first version of unidirectional channels [360]. This protocol is a modified implementation of Nakamoto's transaction replacement protocol. BitcoinJ, a popular Bitcoin library written in Java, supports this protocol [47].

Consider two channel participants: a customer and a merchant. Initially, they lock coins into a multi-signature output and create a time-locked refund transaction. The customer can thus withdraw all funds in case the merchant goes offline. Then the customer signs a transaction that distributes the coins from the funding transaction in a new proportion, allocating more funds to the merchant, and sends the new transaction to the merchant. The merchant either co-signs and broadcasts it, closing the channel, or waits for the next version of the transaction. Shortly before the timelock of the refund transaction expires, the merchant broadcasts the latest transaction. The last transaction closes the channel and confirms the latest agreed-upon balances.

This protocol has two major drawbacks. First, it only supports unidirectional channels. The customer can pay the merchant but not vice versa. Second, the timelock of the refund transaction limits the lifetime of a channel.

A similar unidirectional payment channel design uses `CHECKLOCKTIMEVERIFY` – an opcode added to Bitcoin in 2015 [380]. Contrary to `nLockTime`, which specifies transaction-level timelocks, CLTV allows specifying absolute timelocks for each transaction output.

State replacement Let us explain why the protocols described so far do not support bidirectional payments. Consider a channel between Alice and Bob. Initially, Alice commits 10 coins to a multi-signature address. Thus, she owns 10 coins, and Bob has none. Alice sends 2 coins to Bob by signing a new transaction that spends the multi-signature output and sends this transaction to Bob. The update distributes the coins as follows: 8 coins to Alice, 2 coins to Bob. However, Bob cannot send 1 coin back to Alice. If he signs a new transaction that assigns 1 coin to him and 9 coins to Alice, she would not accept it. She knows that Bob has another valid transaction that gives him 2 coins. Therefore, he can fraudulently broadcast that older transaction and effectively cancel his payment.

This issue is called the *state replacement* problem and is the key challenge in payment channel design. On the one hand, each channel state transition should be represented with signed Bitcoin transactions. On the other hand, only one (latest) channel state should be enforceable on-chain. All previous transactions, representing old channel states, must be provably invalidated.

The state replacement mechanism in unidirectional channels is called *revocation by incentive* [188]. Bob is *incentivized* to close the channel using the last transaction because it gives him the largest amount of money. Bi-directional channels require other state replacement mechanisms.

5.1.3 Replace-by-timelock and Duplex channels

Bidirectional channels can be implemented using timelocks. The two parties exchange a series of transactions. Each of them becomes valid at a point in time closer to the present than the previous transaction. The transaction with the lowest timelock represents the latest state. Any channel party can close the channel by submitting the latest state to the blockchain before other states become valid. This protocol has two shortcomings. First, similar to BitcoinJ's unidirectional channels, timelock-based bidirectional channels have a limited lifetime. The timelock of the first transaction determines when the parties must close the channel. Second, such channels only support a limited number of updates. The difference between the subsequent timelocks must be higher than a security margin. Each party must have sufficient time to close the channel with the latest state before other states become valid. Therefore, the first timelock and the minimal safe difference between timelocks determine the maximum number of channel updates. This state replacement mechanism is known as *replace by timelock*.

Duplex micropayment channels [116] (DMC) implement bidirectional channels as pairs of unidirectional channels. The protocol combines *replace by timelock* and *replace by incentive* state replacement techniques. DMC's key concept is *invalidation tree* – a hierarchical transaction structure for invalidating old channel states. The construction leverages the fact that the first transaction's timelock defines the validity of all subsequent transactions in a chain. A follow-up paper [69] describes a way to share the cost of opening and closing channels among multiple parties.

5.1.4 Poon-Dryja channels (Lightning)

The Lightning Network (LN) [300] overcomes the limitations of earlier payment channel designs. Lightning channels are *bidirectional* and have an *unlimited lifetime*.

Lightning is based on a novel *revocation-based state replacement*. Each payment in a channel *invalidates* the previous one. Though all intermediate states are represented by valid Bitcoin transactions, broadcasting any of them except the latest one leads to economic loss: the other party can then withdraw *all* funds from the channel, punishing the cheater.

The development of the LN is guided by a set of documents called "Basics of Lightning Technology" (BOLTs) [59], followed by several implementation teams. The three most advanced implementations available in 2020 are LND [235] (written in go), c-lightning [73] (written in C), and Eclair [132] (written in Scala). Implementations at earlier stages of development include Electrum [135, 136], lit [232], lpd [237], ptarmigan [308], and rust-lightning [335]. As of September 2020, the LN facilitates the off-chain exchange of more than 1 000 BTC. A separate Lightning Network operates on top of Litecoin [2] – a cryptocurrency similar to Bitcoin.

Lightning takes advantage of two relatively recent updates in Bitcoin: *relative timelocks* and *segregated witness*. A relative timelock makes a UTXO valid only after a specified time has passed after the transaction that created this UTXO is confirmed. This functionality was implemented in BIP-112 [66] (`CHECKSEQUENCEVERIFY`) and activated in 2016. Relative timelocks allow Lightning channels to be left open indefinitely.

Segregated witness Transaction *malleability* was a critical roadblock preventing the development of L2 protocols for Bitcoin. ECDSA signatures used in Bitcoin are malleable, i.e., multiple valid signatures exist for the same message. Therefore, one can create different transactions with the same semantics but different hashes. Recall that a payment channel is initiated in three steps. First, the parties co-sign the funding transaction that creates a multi-signature output. Second, they co-sign the refund transaction that spends the multi-signature output and distributes the funds back in the original proportion. Third, they confirm the funding transaction on the blockchain.

Note that the refund transaction spends the output of the unconfirmed funding transaction. Transaction malleability makes this step unreliable. One of the parties may invalidate the funding transaction by broadcasting a modified version of the funding transaction with the same semantics but a different hash [189].

Transaction malleability also complicates fraud prevention. L2 protocols assume that the parties react to broadcasts of old channel states to the blockchain. With transaction malleability, a hash does not uniquely identify a transaction, which complicates watching the blockchain for relevant events.

Segregated Witness, or *SegWit* was introduced in 2017 and mitigated transaction malleability. Originally, the transaction hash was calculated based on all transaction data, including the signature. SegWit introduced a new category of transaction outputs with the *witness* (i.e., the signature) *segregated* from other components and no longer affecting the transaction hash. SegWit opened the way for the practical implementation and deployment of more advanced Bitcoin-based L2 protocols such as the Lightning Network.

5.2 Lightning Network architecture

We now describe the key details of the Lightning Network protocol.

5.2.1 Nodes

Each *LN node* is defined by an ECDSA private-public key pair. A persistent *node identifier* is derived from the hash of the public key. A user can add a human-readable alias to their node. Operations from a node are authorized with a digital signature created with the corresponding signing key. One user can potentially own several nodes.

Nodes connect to each other in the P2P network identifying themselves by the IP address and the node ID. Revealing the IP address is optional but is required to accept incoming connections. Nodes exchange information about the currently open channels and their fee policies. Nodes communicate with an underlying Bitcoin node (such as Bitcoin Core) to receive information on the confirmed transactions.[3]

5.2.2 Channels

A Lightning channel operates in three stages: opening (locking the coins), operating (performing off-chain payments), and closing (broadcasting the most recent channel state to the blockchain).

Channel opening

Opening a channel consists of several steps. To open a channel to Bob, Alice establishes a connection to Bob in the P2P network and issues a request to open a channel. If the parties agree on channel parameters, they co-sign a *funding transaction* that establishes the initial distribution of funds.[4]. The funding transaction creates a 2-of-2 multi-signature output that can be spent by Alice and Bob together if they agree to do so.

[3]Some LN implementations partially support *pruned* nodes [236].

[4]While in the initial specifications [300] it was assumed that both parties could fund a channel, the current LN channels are single-funded: Alice provides all funds and may optionally "push" some funds to Bob as a gift.

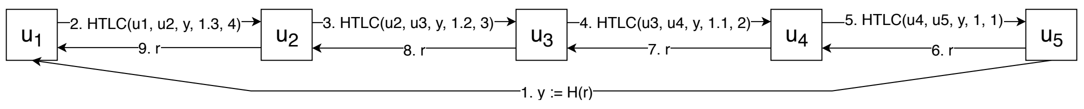

u1
2. HTLC(u1, u2, y, 1.3, 4)
9. r
u2
3. HTLC(u2, u3, y, 1.2, 3)
8. r
u3
4. HTLC(u3, u4, y, 1.1, 2)
7. r
u4
5. HTLC(u4, u5, y, 1, 1)
6. r
u5
1. y := H(r)

FIGURE 5.1: An HTLC-based payment in the Lightning Network.

The channel is open when the funding transaction gets a sufficient number of confirmations (usually 3 to 6). The capacity of the channel stays constant during its lifetime.

Channel updates

An *LN payment* is an atomic update of one or multiple channels. In single-channel payments, two users agree on an updated balance. In multi-hop payments, the balances of several channels forming a path are simultaneously updated.

Single-channel payments To send a payment to Bob, Alice negotiates a new channel state. Each channel state is reflected in a *commitment transaction*. A commitment transaction spends the output of the funding transaction and re-distributes the coins between Alice and Bob.

More precisely, each channel state is encoded in a *pair* of commitment transactions: one for Alice and one for Bob. These transactions are symmetric: they enforce a timelock on the party that holds the transaction. In particular, Alice's version of a commitment transaction allows her to redeem her output only after a timeout. Bob's version imposes analogous restrictions on his output. The timelocks allow the counterparty to dispute an incorrect channel closure. This mechanism provides economic security guarantees to LN channels, assuming the parties are watching the blockchain sufficiently often.

Outputs of commitment transactions are called *hash time-locked contracts* (*HTLC*s). An HTLC allows a node (u_1) to lock x coins in a channel between u_1 and u_2 and release them according to the encoded conditions. The terms for the HTLC(u_1, u_2, y, x, t) are defined with a hash value $y := H(r)$, an amount x of coins, and a timeout t, as follows. If u_2 reveals a value r such that $H(r) = y$ before t expires, u_1 pays x to u_2. If t expires, u_1 receives x back.

A simple LN payment proceeds as follows. If Alice wants to send x coins to Bob, she first asks him for a *payment hash*. Bob generates r uniformly at random and sends its hash $H(r)$ to Alice in an *invoice* message. Alice then *offers* Bob an HTLC that can be *resolved* in one of two ways. Either Bob reveals r and *redeems* the coins before time t, or Alice gets the coins back. A payment channel can keep track of multiple concurrent unresolved, or *in-flight* HTLCs.

Multi-channel payments A multi-channel payment leverages a path of channels between a sender and a receiver, who might not share a channel between them. To initiate a multi-channel payment, the receiver generates a random r and sends its hash $H(r)$ to the sender. The sender then constructs a path to the receiver and sets up an HTLC with the next node in the path. The second node sets up an HTLC with the *same* hash value with the third node, and so on. Finally, the receiver redeems the payment from the last channel by revealing r, which allows all involved channels to be updated.

Note that all HTLCs along the path use the same hash value $y = H(r)$ to achieve atomicity. If the receiver reveals r, all channels are updated. Otherwise, none of them are.

An illustrative example of an HTLC-based payment is depicted in Figure 5.1. Here, the user u_1 transfers 1 coin to u_5 using u_2, u_3, and u_4 as intermediaries. For that, u_5 locally chooses a value r uniformly at random, computes the cryptographic challenge for the HTLC as $y := H(r)$, and sends y to the sender in an *invoice* (step 1). Then, the payment starts with a commit phase (steps 2-5) where every pair of nodes, starting from the sender, establishes an HTLC using y. After the commit phase is finished, the payment enters the release phase. Here, the receiver reveals r to u_4 to fulfill the contract (step 6), triggering the release phase where every pair of nodes fulfills their contract from the receiver to the sender (steps 6-9).

Intermediaries may charge fees for their forwarding service. For instance, u_2 receives 1.3 coins but only forwards 1.2 coins, getting a fee of 0.1 coins. No fees are collected if a payment fails, as all pending balance updates roll back. In the LN implementations, the fee consists of two parts: a constant *base fee* for each payment and a *fee rate* proportional to the payment value.[5]

The time parameter of the HTLCs along the path decreases to guarantee a safety margin between the timeouts. For example, the HTLC between u_1 and u_2 sets a timeout of four days, whereas the timeout in the HTLC between u_2 and u_3 is only three days. The difference between the timelocks ensures that u_2 has enough time to settle the contract with u_1 after receiving r from u_3, even if u_3 reveals the preimage at the last moment. We observe an inherent trade-off regarding timeout lengths. If timeouts are short, a victim of a malicious channel closure may be unable to dispute it if the blockchain is congested at that time. If timeouts are long, an attacker can route many unsettled payments through a channel and effectively block it until the timelock expires.

Multi-path payments use onion routing to enforce the order of intermediary nodes. Each intermediary node only knows the immediately previous and next nodes, but not the final sender or receiver and not its position in the path.

Channel closure

A channel is closed when a transaction that spends the output of the funding transaction gets confirmed on-chain. Channel closure may happen in one of three ways:

- Collaborative closure. Alice signals the intent to close the channel, and Bob cooperates in signing the transaction that reflects the latest channel state. Collaborative closure results in one on-chain transaction and imposes no delays. Both parties must be online and cooperating.

- Non-cooperative closure without breach. Alice signals the intent to close the channel, but Bob does not respond. In this case, Alice publishes the latest commitment transaction on the blockchain. Bob can redeem his output immediately, but Alice has to wait until a timeout expires. This safety measure allows Bob to broadcast a *justice transaction* if Alice were trying to broadcast an old commitment transaction.

[5]The LN fee structure is different from Bitcoin fees, where the fee is proportional to transaction weight (roughly speaking, size in bytes) but does not account for the transaction value. As of 2019, LN fees are largely non-economical [32].

- Non-cooperative closure with a breach. Alice broadcasts an old commitment transaction, thus potentially stealing from Bob. If Bob does not react before the timelock on Alice's output expires, Alice can redeem her output, and the channel is closed.[6] If Bob is online and notices the breach before the timeout, he broadcasts the justice transaction that closes the channel and spends Alice's output before she can spend it, punishing her for the cheating attempt.

This state replacement mechanism makes Lightning superior to earlier payment channel designs. Lightning channels have an unlimited lifetime (the parties do not have to close the channel if none of them wants to) and support bi-directional payments. However, the parties must be online to notice potential malicious channel closures and broadcast justice transactions. LN users may outsource this function to *watchtowers* – entities that watch the blockchain for malicious channel closures and dispute them on the user's behalf. Implementing effective, economically incentivized, and privacy-preserving watchtowers in an active area of research [252].

The LN's revocation technique has proved useful as deterrence against malicious channel closures. As of July 2019, only 241 channel closures have been followed by a justice transaction. This constitutes only 0.7% of channels at that time [50].[7]

5.2.3 P2P network and path-finding

Lightning network is *source-routed*. The sender determines the path to the receiver. LN nodes gossip about new channels available for routing. Based on this information, each node maintains a local model of the network graph and uses it to generate routes to the receiver. The total capacities of public channels are known. The sender only considers channels with the capacity larger than x for a payment of amount x.

However, this is insufficient to prevent routing failures. The ability of channel parties to send or forward payments is limited by their *local* channel balances. Consider an example. After Alice opens a channel with Bob, all funds are initially on her side. She can send up to the total capacity, but she cannot receive payments. As the local balances change, the routing capabilities of the channel in both directions also change.

The lack of information about local channel balances makes routing unreliable, especially for larger amounts. If a payment fails, an error message notifies the sender which channel has failed. The sender then tries a different route. The process repeats until the payment succeeds.

5.2.4 The future of Lightning

The future of the LN depends in part on the planned modifications in the Bitcoin protocol. Such changes usually take years to implement, test, and roll out. Let us outline some of the directions for future developments of Lightning.

[6]From the on-chain point of view, a non-disputed non-cooperative close is indistinguishable from a non-cooperative close without a breach. Layer-one does not know whether a commitment transaction is the last one, unless this fact is disputed.

[7]The research is based on the LN's on-chain footprint and may not show the full picture. A non-cooperative closure followed by a justice transaction may also happen without malicious intent, for example, if a channel is incorrectly restored from backup (where Alice's node "forgets" about the latest state and broadcasts an earlier state assuming it is the latest one).

Schnorr-Taproot *Schnorr signatures* is a digital signature algorithm that is considered preferable to ECDSA currently used in Bitcoin.[8] In particular, this signature scheme allows for arithmetic on signatures. In the context of Bitcoin, this makes multi-signatures indistinguishable from signatures with a single signer. Schnorr signatures are being integrated in Bitcoin as part of a complex *Schnorr-Tapscript-Taproot* update [195]. In the LN context, Schnorr signatures allow for privacy improvements, making Lightning-related transactions indistinguishable from other transaction types to an external observer.

Eltoo *Eltoo*[9] is an alternative payment channel proposal proposed in 2018 [115]. In Eltoo, intermediary transactions are linked linearly, unlike the LN, where they spend the same funding transaction output. On channel closure, the final transaction is "re-attached" to the funding transaction's output allowing for more efficient state replacement.

Eltoo requires a new *signature flag* – `SIGHASH_NOINPUT` [114]. A signature flag specifies whether a transaction signature commits to all or only some of the inputs. Allowing a transaction to *not* commit to any input allows for "re-attaching" it to any compatible output. If `SIGHASH_NOINPUT` is deployed, this replacement mechanism can be used in the LN or a separate payment channel network.

5.3 Research directions in payment channel networks

Multiple research works have shed light on various aspects of payment-channel networks, such as security [240, 216], liquidity [109, 267, 96], topology [246, 345], and routing [142, 305, 242, 187, 287, 294, 329, 351, 20, 295, 297, 420, 419, 418]. Security and privacy are the directions most relevant to our work.

Security Payment channel networks operate in a permissionless environment and should be resilient to attacks. For instance, in the current LN design, no fees are charged if a payment fails, but failed payments consume network resources. An attacker may leverage this issue to launch a DoS attack. Various DoS attacks have been described based on route hijacking [379], depleting channel capacity [291], and exceeding the supported number of concurrent in-flight HTLCs [260].

Privacy From the privacy point of view, layer-two protocols improve upon layer-one. L2 protocols do not store transactions in a globally distributed open database available for analysis. However, multiple weaknesses in LN privacy have been identified. Payment privacy can be breached due to short routes and strong statistical hints [32]. A potential countermeasure – adding noise to channel balances – has been shown ineffective [370]. An attack that allows determining channel balances, similar to our contribution presented in Chapter 6, has also been proposed [194]. A payment channel protocol called Bolt[10] has also been proposed for a privacy-focused cryptocurrency Zcash [184] and later modified for compatibility with Bitcoin and renamed to zkChannels [5]. Lightning network privacy remains an active research area [241, 221, 371, 327, 214].

[8]US patent [341] covering Schnorr signatures expired in 2008. The patent prevented the implementation of this signature scheme in the original version of Bitcoin.

[9]Stylized as *eltoo* in the original paper.

[10]Not to be confused with Basics of Lightning Technology (BOLT) – the Lightning Network specification.

Chapter 6

Probing Lightning channel balances

The LN protocol provides no method for a node to query the distribution of funds in remote channels. In this Chapter, we show that this information may not be private in practice.[1] We demonstrate how a low-resource attacker can infer balances of most channels by sending a fake payment (a *probe*) and observing the resulting error. Our *probing* method shows better accuracy and cost compared to similar approaches described in the literature and scales to the whole network. Compared to earlier approaches [194, 107], we do not need to establish a channel with one of the endpoints of each target channel. The ability to probe remote channels dramatically reduces the time and capital required for the attack. Half of the channels can be probed in under 21 seconds each. The attacker does not spend the committed capital but only temporarily locks it up. We test our proof-of-concept implementation on the Bitcoin testnet and successfully probe a significant portion of its channels. We also outline potential countermeasures, including changes in error handling, sharing channel balances explicitly, and *just-in-time* (JIT) routing.

On a more general note, we raise a question of the *privacy-efficiency trade-off* in the LN. On the one hand, the lack of information about channel balances increases the payment failure rate. On the other hand, the LN does a poor job of protecting this information. Can we strike a better balance between balance privacy and routing efficiency? Answering this question remains an exciting avenue for future work.

6.1 Probing algorithm

6.1.1 Overview

As described in Chapter 5, an LN channel operates as follows. Two parties lock funds in a multi-signature UTXO and then change the distribution of funds in a sequence of off-chain payments. Nodes gossip about channels available for routing and their total capacities. To issue a multi-hop payment, the sender chooses a route based on its local knowledge of the network. Nodes do not announce the distribution of funds in their channels.

Consider two adjacent LN nodes. Let us denote the total capacity of their channel as c. Without loss of generality, let us denote one of the nodes as *source* (with balance b_s) and the other as *destination* (with balance b_d).[2] By definition, $c = b_s + b_d$. Our goal is to determine how c is split between b_s and b_d. For concreteness, for each channel, we estimate b_s, and refer to it as b.

[1]This Chapter is based on [376].

[2]The BOLT specification defines the source as the node with an alphanumerically smaller node ID.

In a nutshell, our algorithm consists of the following steps:

- set up a Lightning node;
- open a few *entry channels* to manually selected nodes (*entry nodes*);
- compile a list of target channels;
- for each channel in the list, perform a binary search for the value of b by sending fake payments through routes ending with the target channel.

6.1.2 Assumptions

To be suitable for probing, the channel must be *active* (available for routing) and *live* (responding to requests). We assume that nodes follow the BOLT specification (in particular, they return errors as prescribed).

Error interpretation The types of errors that we use have broader semantics than our method is aware of. A node returns a `temporary_channel_failure` (`UPDATE|7`) if it is "unable to handle this HTLC, but may be able to handle it, or others, later." We interpret this error as "insufficient balance," though a channel may be unavailable for other reasons. A node returns `incorrect_or_unknown_payment_details` (`PERM|15`) if "[t]he payment_hash is unknown to the final node, the payment_secret doesn't match the payment_hash, the amount for that payment_hash is incorrect or the CLTV expiry of the HTLC is too close to the current block height for safe handling." We interpret this error as only the first of the listed conditions. The payment hash is unknown to the receiving node, as we have generated it randomly. Other conditions should not hold: the amount and the HTLC expiry date should be consistent, as we rely on the standard functionality of c-lightning to construct payments. We assume it to be compliant with the specification and well-tested. Experiments on our own channels show that c-lightning indeed returns these errors under the conditions relevant to our experiment.

A note on probing directions Our probing algorithm is agnostic to the direction of probing. For instance, sending a probe via a route ending in "Alice – Bob" theoretically gives the same information as a probe via a route ending in "Bob – Alice." However, channel directions may have different properties in practice. Each channel party controls the routing policy in its channel direction. Alice may allow routing to Bob without Bob allowing routing to Alice. To extract the maximum amount of information, we probe channels in both directions. Doing so also helps us overcome the technical issue with large channels. Due to the limitation on the LN payment amount, we cannot fully probe large channels. However, if a large channel's capacity is skewed, we can successfully probe it from the "smaller" instead of the "larger" end. For clarity, we omit this implementation detail from the algorithm description.

A note on applicability Our experiments demonstrate the probing technique's feasibility, but the results from the testnet cannot be directly applied to the mainnet. In particular, the mainnet LN contains four times more channels than the testnet LN. Therefore, probing the whole LN on the mainnet would take more than two days instead of 14 hours. We note that the attacker can still target specific channels, such as those belonging to an individual service provider. Tens or hundreds of chosen channels can be probed in a feasible amount of time. Probing selected channels would give the attacker valuable insights into the victim's financial situation.

6.1.3 Selecting channels for probing

First, we compile a list of active and live channels. We define a channel as *active* if at least one of its two directions is announced as active (available for routing) in the gossip data. To determine liveness, we use the heuristics presented in Algorithm 1.

```
Data: Gossip data
Result: Channels selected for probing
for node in gossip data do
    connect to node;
    if connection established then
        add node to live nodes;
    end
end
for channel in gossip data do
    if source and destination in live nodes then
        add channel to channels to probe;
    end
end
for channel in gossip data do
    send a 1 000 sat probe;
    if error returned then
        add channel to channels to probe;
    end
end
```

Algorithm 1: Selecting channels for probing.

Heuristic 1: Connecting to nodes For a channel to be live, both its parties must be live. We extract a list of nodes from gossip data and establish a P2P connection to each of them.[3] We consider a channel live if both its parties are live. We close all the connections after this step, except for the connections to our entry nodes.

Heuristic 2: Pre-probing To further optimize probing, we introduce a pre-probing step. We send a probe of 1 000 satoshis to every channel marked as active in the gossip data.[4] If we get no response, we mark the channel as dead and do not probe it in the main probing step. For the first round of probing, we consider all channels classified as live by either the first or the second heuristic.

Heuristic 3: Liveness detected during probing Each probe results in an error propagated back to the sender. During the first probing round, we expand our list of live channels. If we issue a probe along the route of channels $c_1, c_2, \ldots, c_n$ and receive an error from channel c_i, we conclude that all preceding channels $c_j, j <= i$ are live. If any of c_j is not on our live channels list, we add it. During the second probing round, we use the updated live channels list.

[3]Establishing a P2P connection is nearly instant and, unlike opening a channel, does not require capital commitment.

[4]We use the same probing function as for the main probing.

Channel order

We say that channel c_1 is *closer* to us than channel c_2, if the shortest route from our node to one of the endpoints of c_1 is shorter than that for c_2. We informally refer to a channel as *important*, if a large share of our probes is forwarded through it. Our method is agnostic to the order in which we probe the channels. However, we choose to probe the "closer" and more "important" channels first. The rationale is that it is beneficial to first probe the channels often used as intermediary hops. Knowing their balances allows us to avoid sending payments that would fail due to insufficient balance at an intermediary hop.

We probe channels in the following order:

1. channels adjacent to our entry nodes (the "first layer");
2. channels between hubs – channels connecting nodes out of 1% of the most connected nodes (if not already probed);
3. channels adjacent to the "first layer" (if not already probed);
4. all other channels (if not already probed).

6.1.4 Probing

After compiling a list of live and active channels, we issue a series of probes to each of them.

Recall that in the ordinary course of operation, the payment receiver generates a random value r and sends its hash $h = H(r)$ to the sender. The sender then creates a series of HTLCs with the same hash value h. A probe is an unsolicited payment with a random value instead of h. Such payments fail in any case. However, we can obtain the information on channel balances based on which error occurred and where. In particular, we can infer whether the balance of the erring channel is higher than the probing amount a. Intermediary nodes cannot distinguish randomly generated and genuine payment hashes. They will, therefore, forward our probes as regular payments. If all channels along a route have sufficient balances (higher than a), the probe only fails at the last step. The final recipient finds out that it does not know the preimage for the payment hash and emits the corresponding error message. If any channel along the route has insufficient capacity, the payment fails at that channel before reaching the final recipient.

Let $c_1, c_2, \ldots, c_n$ be a sequence of channels in a route, and b_i be their respective balances. Let c_j be the erring channel. After each probe with an amount a, we obtain the following information:

- $b_i > a$ for $i < j$;
- $b_j < a$ if the error is "insufficient capacity", or $b_j > a$ if the error is "unknown preimage".[5]

The probing algorithm for a single route is presented in Algorithm 2.

For each channel, we keep a lower (b_{min}) and an upper (b_{max}) bound for its balance b. Initially, $b_{min} = 0$ and $b_{max} = c$. At each probing step, we aim at shrinking this interval with binary search, i.e., by issuing a probe with the amount of $\frac{1}{2}(b_{min} + b_{max})$. If the midpoint between b_{min} and b_{max} is larger than the maximum HTLC amount allowed by the specification, we decrease it to that maximum minus a safety margin. The algorithm for all channels selected for probing is presented in Algorithm 3.

[5]The latter is only possible if $j = n$.

```
Data: Route and amount to probe
Result: Updated balance estimates for channels in route
send payment along route;
for channels before erring channel do
    $b_{min} = a$;
end
for erring channel do
    if insufficient funds then
        $b_{max} = a$;
    end
    if unknown preimage then
        $b_{min} = a$;
    end
end
```

Algorithm 2: Probing a route.

```
Data: Gossip data
Result: Improved estimates for channels
SelectChannelsForProbing;
for channel in channels for probing do
    $b_{min} = 0$;
    $b_{max} = c$;
    for number of probings per channel do
        for number of attempts per probing do
            GetRouteToTargetChannel;
            ProbeRoute;
            for channel in route do
                if channel is live and not marked as live then
                    mark channel as live
                end
            end
            if target channel estimates updated then
                continue;
            end
        end
        if required precision reached then
            continue;
        end
    end
end
```

Algorithm 3: Probing all channels.

Choosing routes For each probe for a chosen *target channel*, we select routes towards it based on the following criteria:

- the target channel is the last channel in the route;
- all previous channels in the route have sufficient balances to forward the probe (to the best of our current knowledge).

The route generation algorithm is presented in Algorithm 4.

```
Data: target channel, amount $a$
Result: Route to target suitable for $a$
for channels adjacent to destination do
    if channel is not target then
        add channel to excluded channels;
    end
end
for all channels do
    if $a > c_{max}$ then
        add channel to excluded channels;
    end
end
while route is bad do
    get route to target without excluded channels;
    for channel in route do
        if $a > b_{max}$ then
            route is bad;
        end
    end
    route is good;
end
return route;
```

Algorithm 4: Getting a route to the target channel.

We rely on the built-in functionality of our LN node (c-lightning) to generate routes.[6] The c-lightning API allows us to customize routes, excluding specified nodes and channels. We use this functionality to pre-filter suggested routes based on the information we have obtained through probing so far. If we know that a balance of some channel in a suggested route is insufficient, we exclude this channel from consideration for the current probe. This filtering allows us to speed up the probing by not sending probes through routes with insufficient balances in intermediary channels.

Finally, we perform the second probing pass to probe the channels that have been only detected as live during the first pass (the third liveness heuristic).

Channel information coefficient To measure the effectiveness of our technique, we introduce the *channel information coefficient*. Let c is the original channel capacity, and b_{max} and b_{min} be our upper and lower bound estimates for b. Then the information coefficient is defined as:

[6] Internally, c-lightning uses the Dijkstra algorithm.

$$i = 1 - \frac{b_{max} - b_{min}}{c}$$

Thus, $i = 0$ means that we do not know any extra information about b besides public knowledge. The value $i = 1$ means that we know b precisely.

6.2 Experimental setup

We implement the algorithm described in Section 6.1 as a c-lightning plugin. The plugin functionality allows developers to integrate Python code with the c-lightning node [93]. We only collect data from the Bitcoin testnet. We try to perform at least 7 probings per channel, which means that in a successful probing, we shrink the $[b_{min}, b_{max}]$ interval to $\frac{1}{128}$ of its initial length, determining b with the precision better than 1%.

Entry nodes We launch our own LN node[7] and fund it with testnet coins. Then, we establish five entry channels to handpicked nodes with high connectivity and liquidity. Four of our channels have the maximum standard capacity of 0.167 BTC.[8] One entry channel has a capacity of 0.043 BTC. We choose nodes to connect to based on the following requirements (as reported by 1ML [1]):

- well-connected and well-capitalized;
- located relatively close to our node (i.e., in Europe) to decrease latency.

Choosing the timeout One of the decisions we have to make is to set a timeout, after which we declare a channel unresponsive and move to the next one. The LN protocol does not prescribe how quickly a node should react. The only time limitation is the HTLC timeout, usually on the order of hours or days. Therefore, to probe all channels in a reasonable time, we choose a timeout of 10 seconds. Our later results showed that this was a reasonable trade-off between probing speed and accuracy.

Route selection We generate routes with the standard c-lightning `getroute` routine and exclude routes with known insufficient balances in intermediary channels. Note that route generation is a local operation. One route generation takes under 10 ms in our experiments, two orders of magnitude less than the average probe response time (3 seconds).

In our main experiment, we perform 12 895 payments. We reject 96 768 routes because of low balance. That is, we filter out around 8 routes per payment. Therefore, our method of route generation does not significantly increase the time of our experiment. This approach may be improved with custom route generation.

Another trade-off we have to address is the maximum route length. The LN protocol limits the length of a route to 20 channels. Longer routes allow for collecting more information per probe but increase the probability of failure at intermediary channels. For our experiment, we limit the length of routes to 10 channels.

[7] c-lightning version `v0.8.0-40-g899f5de`.

[8] The LN limits the channel capacity at 0.167 BTC. Larger channels can be created with a special command-line argument and are sometimes called *wumbo channels*.

Hanging HTLCs Our method assumes that an error is returned quickly (within seconds, as Figure 6.1 shows). If some intermediary hop does not return an error, our entry channels are left with an unresolved in-flight HTLC that we call *hanging*. Hanging HTLCs occupy our channels' capacity, preventing us from issuing large probes from the affected channel. The protocol does not allow us to cancel HTLCs unilaterally, and closing a channel involves long timeouts until the funds are available and can be committed to a new channel. Therefore, we use multiple entry channels to be able to tolerate some hanging HTLCs. This issue is related to the attack presented in Chapter 8 and the one described in [260].

6.3 Results

The LN on the Bitcoin testnet contains 1 974 nodes and 5 884 channels, including 2 527 announced as active (as of 26 February 2020). The initial estimate shows that 207 nodes and 1 625 channels are live. We detect 3 more live channels during the first probing pass. The strongly connected component of the live subgraph contains 1 489 channels. The other 139 channels point towards nodes for which we could not get meaningful error messages.

We send 3 153 (24.45%) during the pre-probing and 9 742 (75.55%) during the main probing phase (12 895 probes in total). Out of 9 742 probes in the main phase, 8 256 (84.75%) return errors that we can use to improve the balance estimates ("channel temporary unavailable" and "incorrect or unknown payment details"). The time of the experiment is 14 hours and 6 minutes. Probing 1 628 live channels takes 65% of the time (roughly 9 hours). The rest is spent on slow-responding channels or channels that reply with an unexpected error.

6.3.1 Probing times

First, we consider the distribution of probing times for various route lengths (Figure 6.1).

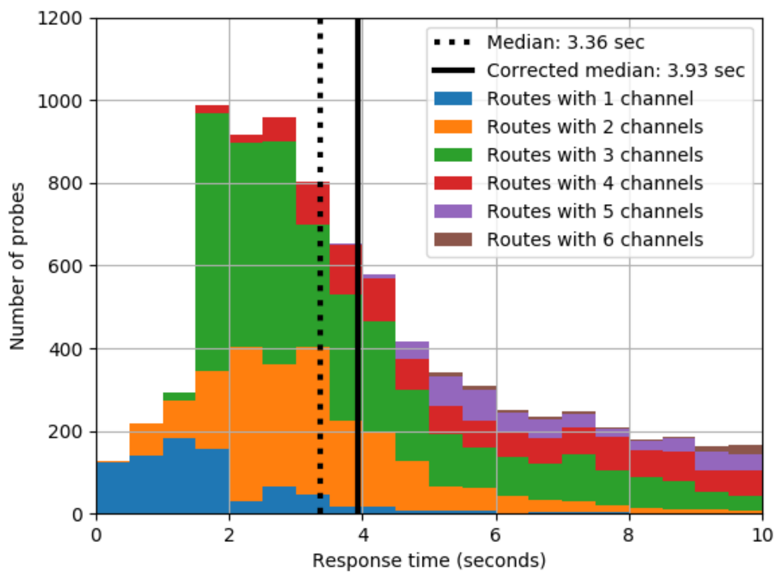


FIGURE 6.1: The distribution of probes (onions) by response time.

Nearly all probes sent along the routes of 3 hops and shorter return within 10 seconds. The median of the 8 256 probes is at 3.36 seconds. Recall that 15.25% of the probes time out or return an error that we cannot interpret within our algorithm. The corrected median without the timed out and erring probes is 3.93 seconds. We conclude that the cutoff at 10 seconds presents an acceptable trade-off. The diameter of the strongly connected component on the LN is 3.[9]

Next, we consider the time it takes to probe a single channel. With the parameters we have chosen, each probe cannot take longer than 70 seconds (7 probes of up to 10 seconds each). For each channel, we add up the times of probes sent to this channel. Figure 6.2 shows the cumulative distribution function of channels by the total time spent probing them. We observe that 50% of channels can be probed in less than 21.2 seconds.

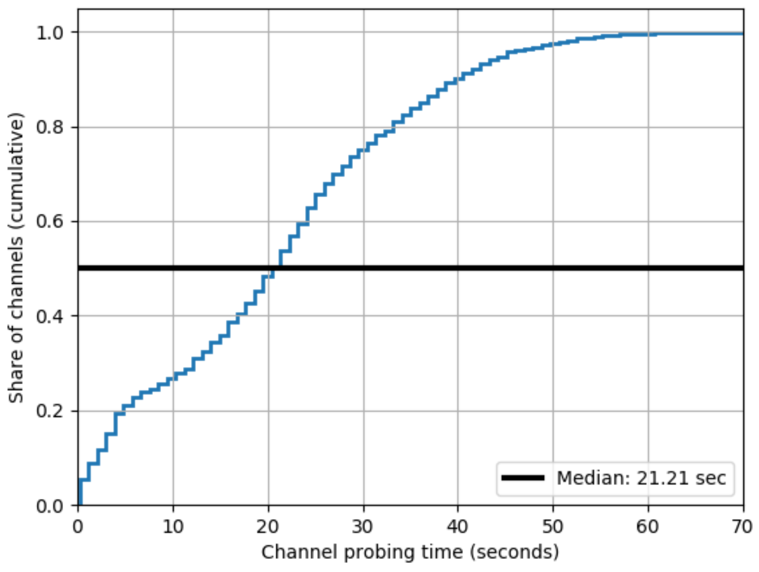


FIGURE 6.2: Distribution of channels by total probing time.

6.3.2 Probing coefficients

We use the channel information coefficient to measure how much information we obtain for each channel. The LN specification limits the value of a single payment to 0.043 BTC. We denote channels with the capacity more than two times that (i.e., $2 \times 0.043 = 0.086$ BTC) as *large channels*. We call other channels *small*. All small channels can, in principle, be fully probed. A large channel can only be fully probed if the local balance at one of its endpoints is smaller than the maximum payment amount.

Figure 6.3 represents the distribution of channels by their information coefficients. We obtained full balance information on over 1 000 of 1 628 channels. We explain the jump at 0.5 for the large channels as follows. The maximum standard channel capacity (0.167 BTC) is approximately four times the maximum HTLC amount (0.043). Consider a channel with a total capacity of 0.167 BTC and the local balance between 0.043 BTC and $0.167 - 0.043 = 0.124$ BTC. We probe this channel from both sides and receive the "unknown hash" error in both cases. From that, we conclude

[9]It is not always possible to use the shortest route, as it may not have sufficient balances in all channels.

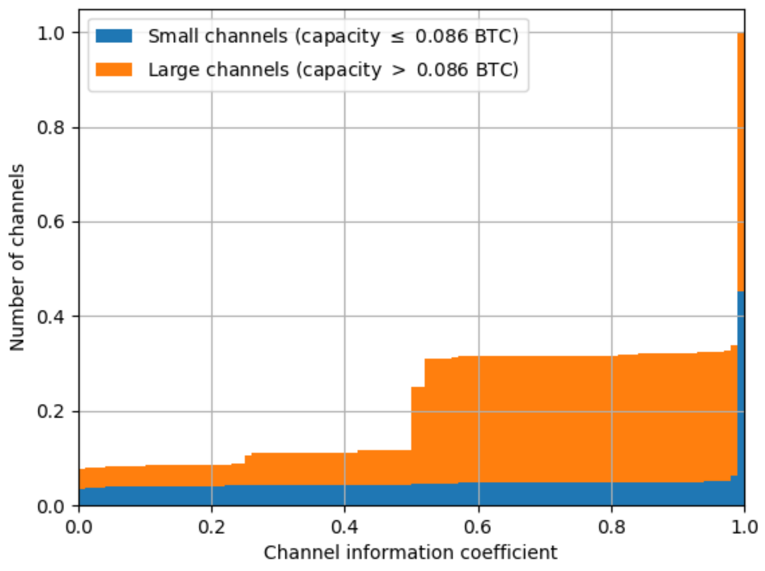


FIGURE 6.3: Distribution of channels by the obtained information coefficient.

that the local balance is between 0.043 BTC and 0.124 BTC. This estimate yields a channel information coefficient of 0.515. However, with our current technique, we cannot improve it.

6.3.3 Distribution of channels in routes

We also want to understand how often we send probes through each channel. Most of our routes go through the same few channels (Figure 6.4), which is partially explained by the fact that all routes include one of our entry channels.

6.3.4 How balanced are the channels?

A channel is informally called *balanced* if the parties have roughly equal balances. We introduce the *balance coefficient* to quantify this. The balance coefficient represents the distance from the actual channel balance to 0.5 of the total capacity, where b is the estimated local balance and c is the total channel capacity:

$$c_{bal} = 0.5 - \frac{|b - c|}{c}$$

A channel is unbalanced if its whole capacity is on one side ($c_{bal} = 0$). A channel is perfectly balanced if the two parties have equal balances ($c_{bal} = 1$).

Figure 6.5 depicts the distribution of balance coefficients among "small" channels that we are able to probe with high accuracy (information coefficient higher than 0.9). We conclude that many small channels are unbalanced (coefficient close to zero). 15% have the balance coefficient below 0.001, 45% below 0.01, and 62% below 0.1. However, note that the picture may change if we consider large channels, and that channel management practices on mainnet may differ.

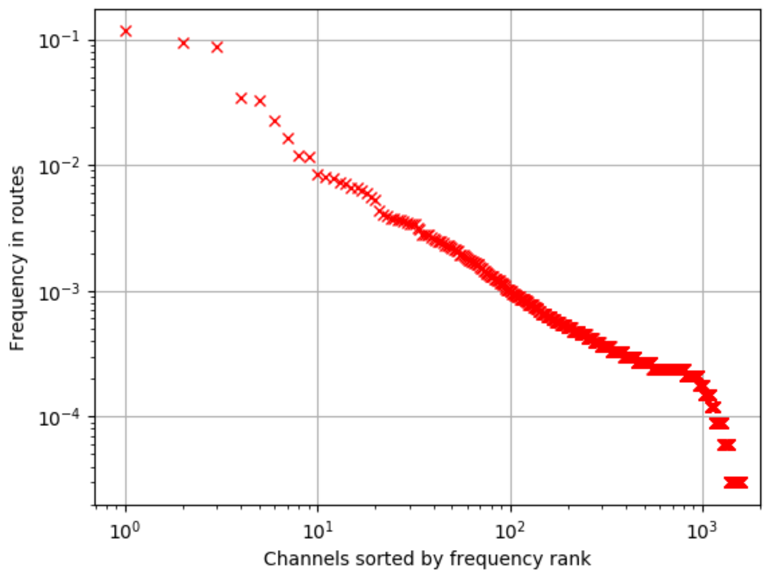


FIGURE 6.4: Distribution of relative frequencies of channels in routes.

6.4 Estimating the attack cost

The attack requires moderate resources. The attacker only needs to commit funds to the entry channels. The capacities of the entry channels determine the maximum probing amount. The computational and communication requirements are similar to the ones required to run a standard LN node. The adversary only needs to maintain a few TCP connections during the main phase of the experiment. (We also open and immediately close connections to all nodes to check their liveness; this process can be parallelized.) Note that running an LN node implies running a fully synchronized Bitcoin node, which requires hundreds of gigabytes of storage (265 GB at the time of our experiments in February 2020).

With our current approach, the maximum probing amount is the protocol's limit of 0.043 BTC. This is the minimal amount the attacker has to commit to theoretically be able to probe all "small" channels fully. Our experience shows that it is beneficial to open multiple channels to decrease the negative effect of hanging HTLCs. In our experiments, we use five entry channels.

Note that since all probing payments fail, the attacker pays no fees. If no HLTCs are left unresolved after the probing, the attacker can close the entry channels collaboratively and immediately withdraw the committed funds. If some HTLCs are left unresolved, or the attacker's channel partners are offline or unwilling to cooperate on channel closure, the attacker would have to wait for the agreed-upon timeout (usually on the order of days) before withdrawing the funds. In any case, no coins are irrevocably lost. However, the attacker still bears the opportunity cost: the coins committed to the attack could have been invested elsewhere.

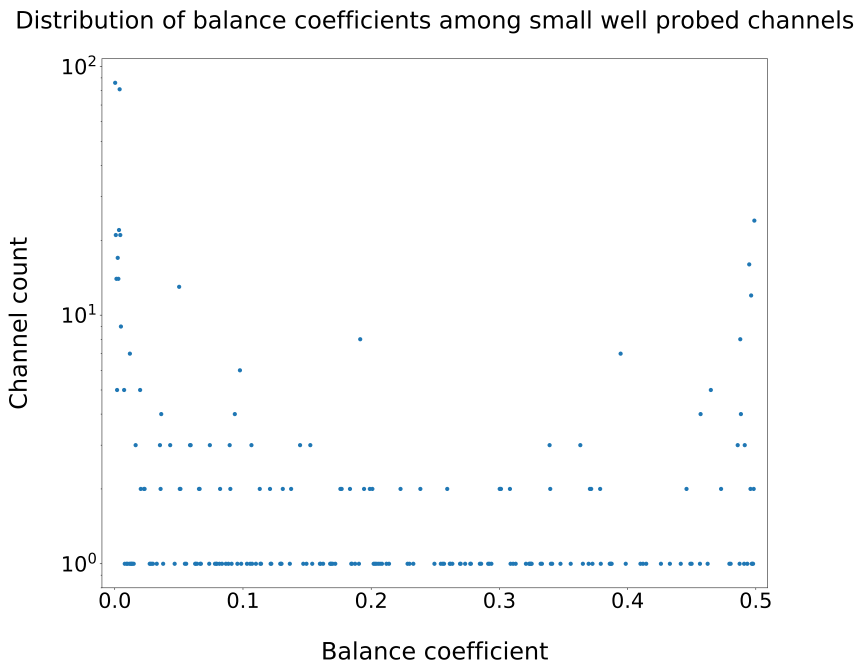


FIGURE 6.5: Distribution of balance coefficients.

6.5 Limitations

Now we discuss the limitations of our approach and potential ways to improve it.

Unannounced channels LN nodes do not have to announce their channels. For example, casual users using mobile devices are not supposed to do so. According to a 2020 study [321], 28% of LN channels are unannounced.[10] Unannounced channels are not prone to our probing methodology. The sender does not know the private channel endpoint's identifier and cannot construct a route to it.

It may be possible to extend our technique with on-chain heuristics to locate unannounced channels. In particular, each channel has a short identifier composed of the block number, the transaction index, and the UTXO index of the funding transaction's multisignature output. An attacker may scan the blockchain looking for such outputs and cross-reference them with the LN gossip data [296].

Lack of suitable routes We cannot probe a route if we do not find a suitable route to the target channel. In particular, we cannot probe a high-capacity channel if only a low-capacity channel connects it to the rest of the network. We can partially overcome this limitation by diverging from the series of probing amounts determined by binary search. Recall that for each yet unknown channel balance b we maintain the current estimation interval $[b_{min}, b_{max}]$. The binary search prescribed to choose the next probing amount as $\frac{b_{min}+b_{max}}{2}$. If this value is too high, we may instead use the maximum value for which we can find a suitable route. Decreasing the probing amount would allow us to obtain at least some information on the channel in question. In the initial version of our algorithm, we do not do it for simplicity.

[10]Unannounced channels are also called *private*.

Concurrent probing of large channels Recall that our method cannot fully probe large channels because of the limitation on the HTLC amount. Lightning implementations impose a limit on the maximal amount transferred in one HTLC. This limit is approximately 0.043 BTC.[11] Therefore, we can only fully probe channels where the local balance of one of the channel parties is below 0.043 BTC.

A possible way to overcome this limitation would be to probe large channels with multiple HTLCs concurrently. The goal of concurrent probing is to temporarily block the capacity of the attacked channel. Our current method does not support concurrent probing because we only control the sender, not the receiver. An honest receiver quickly returns an error and unblocks the capacity. Concurrent probing would involve another malicious node acting as the receiver of all our probes. The malicious receiver would deliberately delay the response, thus temporarily blocking funds along the route.

Concurrent probing could also decrease the probing time, bringing the results closer to an instant network snapshot. However, adding concurrency is a non-trivial task. Parallel probings may interfere with each other if the same channel is involved in two routes probed simultaneously.

Note also that some realistic attack scenarios do not involve probing the whole network. An adversary may choose one "important" Lightning node and probe all its channels relatively quickly. The obtained information may be a business secret of the node operator.

Parallel channels and non-strict forwarding Our method is based on the assumption that the probe is forwarded through the *channels* determined by the sender. However, the LN specification only guarantees that the payment follows the chosen sequence of *nodes*. A pair of nodes may share multiple *parallel* channels. A forwarding node is free to choose a channel from all parallel channels to the next node. This practice is known as *non-strict forwarding* and provides flexibility in addressing local balance restrictions. While the LN specification allows non-strict forwarding, c-lightning cannot open multiple channels to the same node. The other two popular implementations, LND and Eclair, support parallel channels. Therefore, we cannot ensure that an intermediary node uses a given channel. It may forward our probe through a parallel channel instead. We accept this issue as a limitation of our approach.

As seen from our LN snapshot dated 25 February 2020, the mainnet LN contained 1 438 parallel channels (17.64% of all channels), which indicates that the effects on the probing precision could be significant on mainnet.[12] However, most *node pairs* have at most one channel, which thus can be probed using our method.

6.6 Countermeasures

The simplest countermeasure that does not require protocol changes can be implemented as part of a node routing policy. Note that all our probing payments fail (either due to insufficient balance or unknown hash preimage). Intermediary nodes know whether a payment they participate in succeeds or fails. Therefore, an intermediary node observing a flood of failing payments from the same channel may suspect a probing, especially if the amounts follow the binary search pattern. An intermediary node can then close the channel or otherwise limit the flow of failing

[11] 4294967295 millisatoshis.

[12] Unannounced parallel channels may also influence our results.

payments from the node in question. Of course, the adversary can trick such detection techniques, for example, by connecting to Bob via Alice and making Bob think that Alice is performing the probing.

We divide the other potential countermeasures into two categories: prioritizing privacy and prioritizing efficiency.

Prioritizing privacy

We argue that reliably protecting channel balances in the LN is currently infeasible. This conclusion comes from the following observations:

- the sender knows whether the payment has failed or succeeded;
- if the payment fails, the sender knows the erring channel.

However, we can change the protocol to make the latter assumption not hold.

Merging error types When a payment fails, the sender receives an error message. Depending on the cause of the error, these messages differ in two ways. They have different error codes and originate from different nodes. In particular, if the target channel has insufficient balance, the error is returned by the *previous* node. If the target channel has enough balance, then the *final* recipient reports incorrect payment details. We propose a change to error handling in the LN that would prevent the sender from knowing where the payment has failed. In particular, each node in a route changes the error it sends back as if it has originated from its own channel. We also suggest merging the two error types ("incorrect or unknown payment details" and "temporary channel failure"). A similar countermeasure has already been implemented (see note about error types 16 and 17 in BOLT4 [58]). The drawback of this method is a decrease in payment reliability. The sender can no longer exclude the failing channel from the subsequent route search. However, the payment reliability problem may become less pressing with *multi-part payments* (MPP) that split a large payment into small parts that are sent along different routes and thus increase routing efficiency.

Added loops Another potential countermeasure would be for intermediary nodes to add extra hops to the route. Currently, the sender chooses the route. The order of nodes in the route is enforced with onion routing. If this scheme is modified, an intermediary node could instead forward the payment to the next node in the route through an added random sub-route. Added loops would blur the picture for the sender, as the sender would not know which path the payment has taken. One drawback of this approach is the requirement to substantially change the onion routing protocol, which, as argued in [240], is necessary for LN security. The fee structure would also have to be more complex.

JIT routing Just In Time Routing (JIT routing) algorithm has been originally proposed to improve payment reliability [295, 297]. JIT routing works as follows. If a forwarding node lacks the balance to forward a payment, it sends a circular payment to itself to add more funds to the necessary channel. This process is called *channel rebalancing*.[13] A JIT-supporting node does not send an error message back

[13]Another research paper [96] analyzes the influence of hubs on the LN and proposes a channel rebalancing algorithm.

if it lacks funds on the attacked (probed) channel in the probing scenario. Instead, it interrupts the routing process, re-balances its channels, and then continues the forwarding. The attacker would interpret the lack of error as a signal that the target channel has enough funds, but the same would hold if the channel is probed from the other side. JIT routing can be an effective countermeasure against channel probing attacks. However, timing attacks may become an issue.

Prioritizing efficiency: sharing balance data

If hiding LN channel balances is infeasible, we may want to use balance information to improve routing efficiency. Broadcasting all intermediate balances to all nodes would introduce a large networking overhead. We propose to develop a reasonable method for nodes to share information about their channel balances selectively. This information would improve path-finding and help nodes decide how to allocate funds to new channels. We propose adding an API call that would allow the sender to query a channel's balance it wants to route a payment through. In this scenario, a sender creates a preliminary route and asks the nodes along this route, whether they have sufficient balance. If some of them do not, the sender re-calculates the route until a suitable route is found. This algorithm would improve upon the current LN payment workflow, where a sender is receiving errors and re-sending a payment along multiple routes until it succeeds. Nodes could develop policies regarding balances, for instance, only reveal balances to trusted nodes, or only to nodes that pay a fee. A node's ability to reveal a channel balance for routing purposes may also be subject to negotiation between channel partners during channel establishment. A detailed analysis of this protocol is needed to prevent abuse.

6.7 Conclusion

Hiding balances from everyone except for the channel parties is a cornerstone of L2 privacy. Making intermediate channel balances available would enable revealing remote channel balances in real time. However, unknown local balances decrease routing efficiency. The sender cannot know in advance whether the chosen route can handle the required amount. Therefore, LN payments often fail due to insufficient balance at an intermediary hop. In that case, the payment is attempted again with a different route. The sender could prevent such failures if it knew the channel balances in advance.

Our experiments show that channel balances cannot be considered private data. A low-resource attacker can probe the balances of most live and active channels with high precision. We implement and evaluate our technique on the Bitcoin testnet, successfully probing a large portion of channels. We identify a *privacy-efficiency trade-off*: hidden balances improve privacy but hinder routing efficiency. The LN does not address this trade-off optimally: channel balances are neither well protected nor utilized. We envision two paths for LN development with either privacy or routing efficiency prioritized. Future research is needed to find the right balance between the two.

Chapter 7

Quantitative analysis of Lightning network privacy

Payment channel networks such as the Lightning Network introduce novel security and privacy challenges. Multiple attacks on the LN have been described [241]. In *value privacy* attacks, the adversary learns payment amounts. In *relationship anonymity* attacks, the mapping between senders and receivers becomes known. The *wormhole attack* [240] allows an adversary to steal fees from honest intermediaries and temporarily block their funds.

In this Chapter, we quantitatively analyze the LN's resistance to the three attacks.[1] We estimate attack success probabilities based on a simulated network for an array of parameters. Our findings suggest that the privacy of the LN depends on highly connected and highly capitalized nodes. An attacker who successfully compromises those nodes succeeds with a high probability. Our results are concerned with the LN as described in the specification and apply to all implementations.

7.1 Datasets

We use a snapshot of publicly announced LN channels as of 25 February 2020 [163]. This snapshot consists of 5 929 nodes and 35 233 channels. We model the LN as an undirected multi-graph (i.e., may contain multiple edges between each pair of nodes), as two LN nodes can share several channels. We only consider the largest connected component, which contains 5 862 nodes (98.87%) and 35 196 channels (99.89%). We refer to this dataset as *LN20*.[2]

Based on *LN20*, LN nodes have an average degree of 12.01 and a median degree of 3 (Figures 7.1 and 7.2). Most nodes have only a few channels, whereas a small number of nodes have many channels. In particular, 1 744 nodes have degree 1, and the most connected node has 1 198 channels. The capacity is also unequally distributed. These observations motivate the methodology in our experiments in Section 7.2.

Ethical considerations Our analysis is based solely on publicly available data. All our calculations use a local representation of the LN network graph obtained from [163]. We do not interfere with the LN activity, nor deanonymize any nodes.

[1] This Chapter is based on [375].

[2] Our code is available at [373].

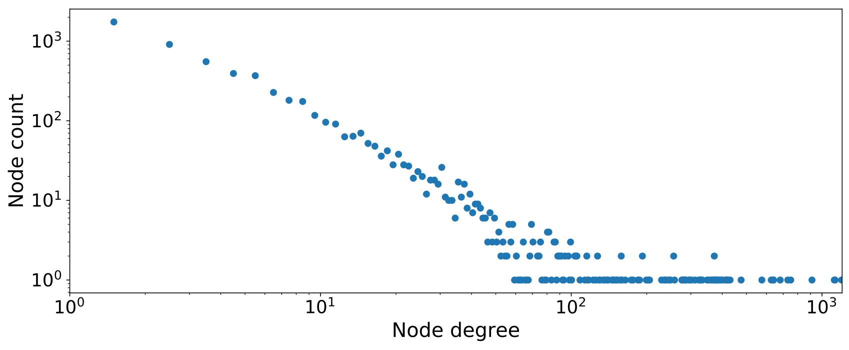


FIGURE 7.1: Node degree distribution.

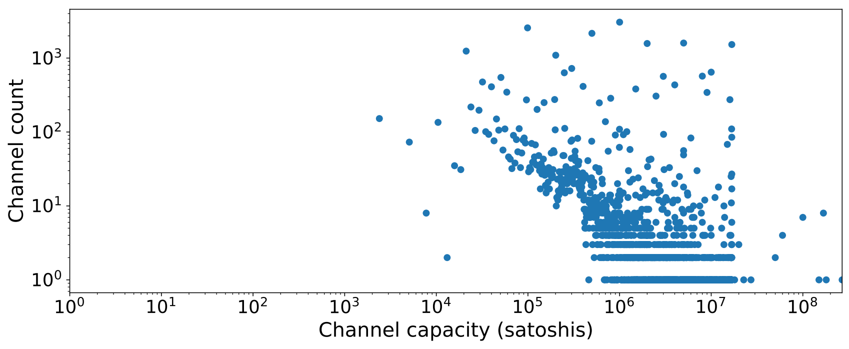


FIGURE 7.2: Channel capacity distribution.

7.2 Security and privacy attacks: background

The LN builds upon hash time-locked contracts aiming to achieve atomicity in multi-hop payments. However, [240] argues that due to the *wormhole attack*, atomicity does not hold in the LN. Another study [241] shows that attackers can breach the privacy of LN users. The feasibility of these attacks depends, among other factors, on the topology of the LN. In this experiment, we aim to quantify the impact of these security and privacy issues on the LN.

Value privacy

Intuitively, value privacy ensures that for a payment involving only honest users, corrupted users outside the payment path learn no information about the payment value. This notion thus heavily relies on the existence of paths without malicious nodes. Otherwise, a malicious intermediary node can trivially learn the (upper bound of the) amount of a payment that it forwards. For instance, in Figure 7.3 the adversary u_3 forwards 1.2 coins to u_4, estimating the payment amount at around 1 coin plus forwarding fees.

Relationship anonymity

Intuitively, relationship anonymity ensures that given two simultaneous payments between two pairs of nodes (u_1, u_2) and (u'_1, u'_2) routed through the same path of intermediary users $i_1, \dots, i_n$, the adversary controlling some of those intermediaries cannot tell who is paying to whom with probability better than $1/2$. However, the LN does not achieve relationship anonymity. An adversary controlling i_1 and i_n can

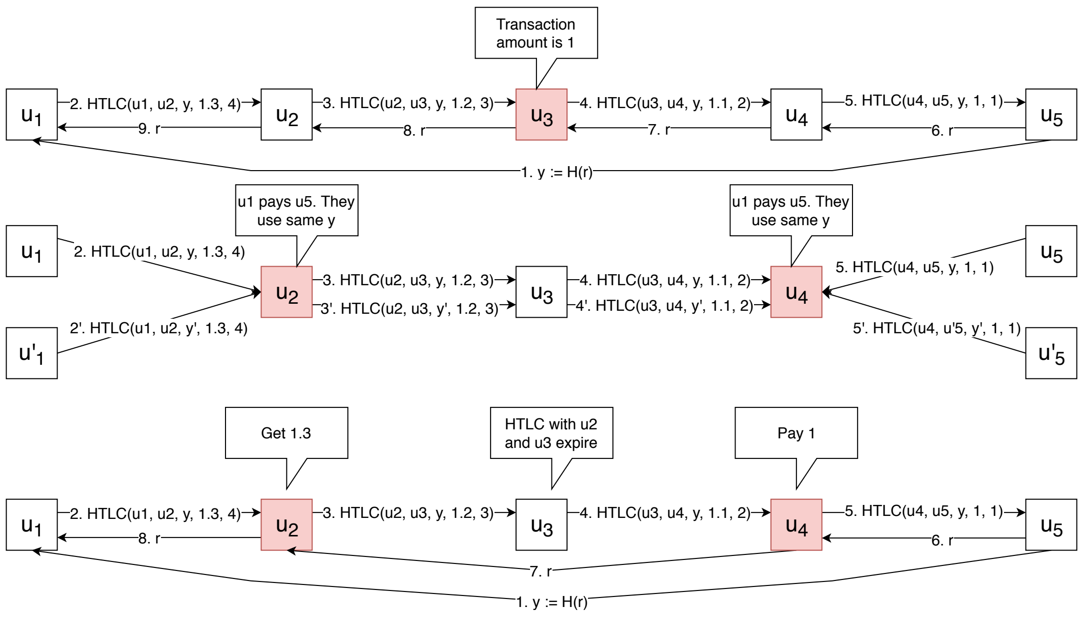

Transaction amount is 1
u1
2. HTLC(u1, u2, y, 1.3, 4)
u2
3. HTLC(u2, u3, y, 1.2, 3)
u3
4. HTLC(u3, u4, y, 1.1, 2)
u4
5. HTLC(u4, u5, y, 1, 1)
u5
9. r
8. r
7. r
6. r
1. y := H(r)
u1 pays u5. They use same y
u1 pays u5. They use same y
2'. HTLC(u1, u2, y', 1.3, 4)
3'. HTLC(u2, u3, y', 1.2, 3)
4'. HTLC(u3, u4, y', 1.1, 2)
5'. HTLC(u4, u'5, y', 1, 1)
u'1
u'5
Get 1.3
HTLC with u2 and u3 expire
Pay 1

FIGURE 7.3: An illustrative example of value privacy (top), relationship anonymity (middle), and the wormhole attack (bottom).

use the cryptographic challenge included in the HTLC to determine who pays to whom. For instance, in Figure 7.3 the adversary controlling u_2 and u_4 can determine that u_1 is transacting with u_5 as the same value y is used along the whole path. Similarly, u_2 and u_4 can determine that u_1' is transacting with u_5' as the same y' is used along the path.

The wormhole attack

In the wormhole attack, two colluding nodes in a payment path prevent honest intermediaries from participating in the payment, stealing the fees intended for honest intermediaries. One may argue that this is not an attack, as, from the sender's point of view, the payment is delivered. However, this attack diminishes the economic incentive for honest users to forward payments in the first place.

An example of the wormhole attack is depicted in Figure 7.3. Here, u_4 does not send the opening value r to u_3 (step 7 in Figure 5.1). Instead, u_4 sends the value r to u_2 outside the LN protocol, which allows u_2 to settle the HTLC with u_1. As a result, contracts with u_3 expire, simulating payment failure, and prevent u_3 from participating in the payment's successful completion.

7.3 Methodology

We first compute the paths between pairs of nodes. Given nodes u_1 and u_2, we compute the list of paths that connect them with one restriction: we consider only the paths with at most three intermediary nodes. We observe that these path lengths suffice to allow more than 85% of payments between a random pair of nodes (Figure 7.4) and allow us to exemplify all attacks that we want to study in this experiment.

Let $paths_{\langle u_1,u_2\rangle}$ be the set of paths between u_1 and u_2. We prune the set $paths_{\langle u_1,u_2\rangle}$ into a subset $paths_{\langle u_1,u_2\rangle,x}$, containing only the paths that allow to transfer at least

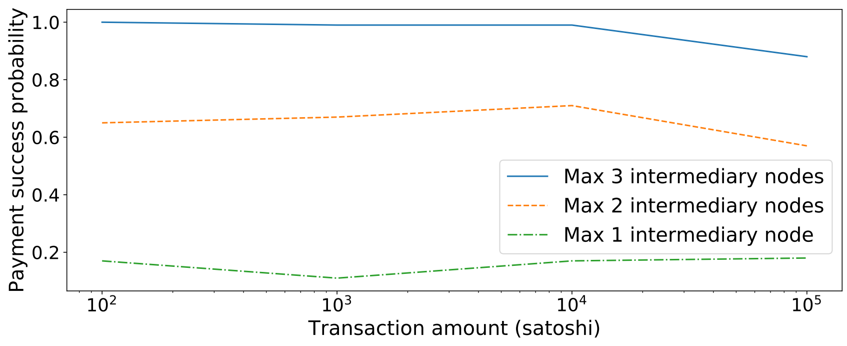


FIGURE 7.4: The share of experiment runs where paths with sufficient capacity exist between sender and receiver.

x satoshis. For instance, $paths_{\langle u_1,u_2\rangle,10}$ contains the paths between u_1 and u_2, which allows transferring at least 10 satoshis.

For a channel to be capable of transferring x satoshis from u_i to u_j, u_i must have a balance of at least x satoshis. However, the current balance of each counterparty in a channel is not publicly available. Thus, we consider a path suitable for a given payment if the total capacity of every channel in the path is above the payment amount, independently of how this capacity is distributed between the counterparties. This heuristic might consider a path suitable for a payment while it is not. We nevertheless follow this heuristic as LN nodes also use it in practice.

As the next step, we study the effectiveness of the selected attack. For a chosen payment amount x, we split the set $paths_{\langle u_1,u_2\rangle,x}$ into two subsets:

1. $paths\text{-}prone_{\langle u_1,u_2\rangle,x}$: the subset of paths prone to the attack;
2. $paths\text{-}safe_{\langle u_1,u_2\rangle,x}$: the subset of paths not susceptible to being attacked.

The definition of a path of the form $u_1 \to i_1 \to \ldots \to i_n \to u_2$ being prone to the specific attack depends on the attack:

- *Value privacy*: We say that a path is prone to the value privacy attack if any intermediary node is under adversarial control.
- *Relationship anonymity*: We say that a path is prone to the relationship anonymity attack if nodes i_1 and i_n are under adversarial control.
- *Wormhole attack*: We say that a path is prone to the wormhole attack if there exist two non-neighboring intermediary nodes i_j and i_k under adversarial control (i.e., $j < k$ and $k \neq j + 1$).

Note the difference between the definitions of a prone path in the wormhole attack and the relationship anonymity attack. The latter does not require an honest user to be located between the two malicious nodes. For instance, a path of the form $u_1 \to i_1 \to i_2 \to u_2$ where i_1 and i_2 are under adversarial control would be considered prone to the relationship anonymity attack but safe against the wormhole attack.

Another aspect that we consider is which nodes are malicious. We follow three strategies. First, we assume that nodes with the highest degrees (i.e., highly connected nodes) are colluding. Highly connected nodes have the highest stake in the network. Thus, an adversary might attempt to corrupt them (e.g., by bribery or stealing the private key) to maximize the effect of the attack. Second, we assume that nodes with the highest total capacity in their adjacent channels are colluding. Finally, we consider that random nodes are colluding. We illustrate here that any node (independently of its node degree) might be corrupted. For instance, the same user might create several LN nodes at strategic positions to carry out the attacks.

We then consider these three attack strategies. For each number of malicious nodes (y) and each strategy, we re-split the set $\textit{paths-prone}_{\langle u_1,u_2\rangle,x}$ into prone paths and safe paths. For instance, we denote by $\textit{paths-prone}_{\langle u_1,u_2\rangle,x,y\text{-}con}$ the subset of paths between u_1 and u_2 that allow to transfer x satoshis and that are prone to the attack if y nodes with the highest node degree are corrupted. Correspondingly, we denote by $\textit{paths-prone}_{\langle u_1,u_2\rangle,x,y\text{-}ran}$ the subset of paths between u_1 and u_2 that allow to transfer x satoshis and that are prone to the attack if y nodes chosen uniformly at random are corrupted.

Finally, for each attack strategy, we consider

$$\alpha_{\langle u_i,u_j\rangle} := \frac{|\textit{paths-prone}_{\langle u_i,u_j\rangle,x,y}|}{|\textit{paths-prone}_{\langle u_i,u_j\rangle,x,y}| + |\textit{paths-safe}_{\langle u_i,u_j\rangle,x,y}|}$$

the probability that a payment between u_i and u_j is vulnerable to the attack. Averaging across all the pairs of nodes tested, we extract the final probabilities reported in Figure 7.5.

7.4 Results and discussion

For every attack and a given number of compromised nodes, the share of prone paths is relatively stable for all payment amounts (Figure 7.5). This result indicates that the payment amount does not significantly affect the security of payments.

The three attacks differ in how quickly the share of prone paths changes as the number of compromised nodes increases. For value privacy, the effect of added highly-connected nodes being compromised is the most profound. The share of prone paths is 50% if only the 5 most connected nodes are compromised, and nearly 100% if the 100 most connected nodes are compromised. Thus, we conclude that an adversary needs to corrupt only 2% of the nodes to almost nullify any value privacy guarantee in the LN.

The average share of prone paths decreases for relationship anonymity. However, the adversary controlling the 100 most connected nodes can launch the relationship anonymity attack on about 70% of the paths. Interestingly, the adversary has fewer possibilities to launch the wormhole attack. For instance, with 100 most connected nodes corrupted, around 30% of the paths are prone to the attack. A restrictive path structure in the definition of the attack may explain this reduction in the attack's effectiveness.

The attacker benefits less from compromising high-capacity nodes, as opposed to high-degree nodes. This distinction is most profound for relationship anonymity: around 50% of paths are vulnerable if the 50 highest degree nodes are corrupted, but only around 25% paths are vulnerable if the 50 highest capacity nodes are corrupted. This difference may be explained by the fact that routing algorithms optimize for short paths. Note that forwarding channels' capacity is not as important as good connectivity, especially for payments of small and medium amounts.

Finally, we consider random nodes compromised. In contrast to the earlier results, less than 10% of paths are prone to value privacy, and nearly no paths are

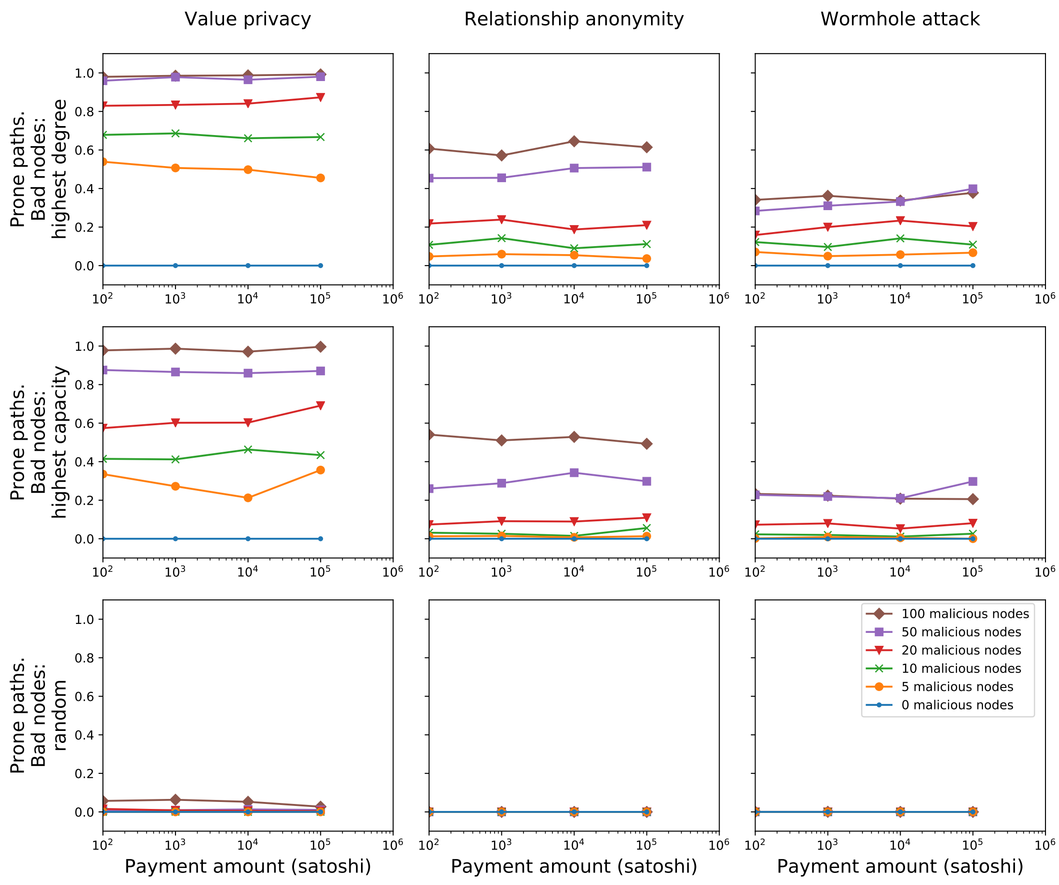


FIGURE 7.5: Share of prone paths for each parameter combination.

prone to relationship anonymity and wormhole attacks. We note that randomly selected nodes have few connections (note the degree distribution in Figure 7.1). Thus their compromise does not affect routing at large.

In summary, our results show that highly connected nodes and nodes with high capacity channels have a high impact on the security and privacy of the LN. Assuming that paths are selected uniformly at random from the set of available paths, an adversary that selectively corrupts 100 (i.e., only 2%) of LN nodes can effectively learn all the payment values, the sender and the receiver for most payments, and carry out the wormhole attack in about 30% of the paths. These estimations evidence that the security and privacy attacks shown in theory are indeed crucial in practice.

Carrying out such attacks might be feasible in the live network: an unknown entity under the pseudonym LNBIG is known to control 23 out of top 50 highest connected nodes [1] and 40% of the network capacity [53] (as of 2019).

7.5 Countermeasures

We assume that every two nodes carry out their payments along a path chosen uniformly at random from the set of all available paths. However, LN nodes might implement different routing strategies. For instance, while routing through well-connected nodes improves the chances to reach the receiver through a short and highly liquid path, the sender might instead choose low degree nodes. Routing around popular nodes may reduce the probability of choosing a compromised path. We envision a trade-off between connectivity on the one hand and security and privacy on the other, which constitutes a direction for future work. A node may also route payments through a trusted proxy node, thus guaranteeing that the first node in a path is not compromised. This strategy would mitigate the relationship anonymity and wormhole attacks (if the path contains no more than three intermediaries).

7.6 Conclusion

The LN has emerged as the most widely deployed solution for scalability issues affecting current blockchains such as Bitcoin. Despite its conceptual appeal and growing adoption, several works [241, 240] have identified security, anonymity, and scalability limitations. However, a quantitative analysis of their impact is missing, and this work aims to fill this gap.

We quantitatively study the LN's proneness to the wormhole attack and attacks against value privacy and relationship anonymity. We observe that a moderately resourceful adversary controlling only 2% of the total node count can carry out these attacks with high success probability. As the LN evolves, the developers should acknowledge these results in future protocol design decisions.

Chapter 8

Throughput limitation of the Lightning network

In this Section, we describe an inherent limitation on the number of concurrent payments Lightning channels can handle.[1] Due to the size limitations on Bitcoin transactions, a payment channel can hold only a certain number of concurrent unresolved HTLCs. An attacker can create unresolved payments between two nodes under their control. This attack blocks the capacity of channels along the route by depleting their "HTLC slots." This effect may be critical for micro-payment applications – a significant use case for the LN. This limitation has been pointed out [141] but never quantitatively analyzed.

We study the management of concurrent payments in the LN and quantify its negative effect on scalability. We observe that for micropayments, the forwarding capability of up to 50% of channels is under-utilized. This phenomenon not only hinders scalability but also opens the door for DoS attacks. We estimate that a network-wide DoS attack costs within 1.64M USD, while isolating the biggest community from the rest of the network costs only 250k USD. We also evaluate the evolution of this phenomenon over the two years of the LN's existence, based on the historical data. We finally discuss potential countermeasures.

8.1 Background

Bitcoin Core imposes a 100 KB transaction size limit [168, 46]. More precisely, transaction size is one of the requirements for a *standard* transaction. Bitcoin Core nodes, which make up 97% of the network [108], do not propagate non-standard transactions, which are therefore unlikely to get confirmed. An LN channel cannot contain more than 966 unsettled HTLCs [56]. This limit ensures that both counterparties can close the channel using one standard Bitcoin transaction. We call this limitation the *HTLC limit*.

Despite the perceived focus on micropayments, the LN does not fully support payments of very small value. Every HTLC increases the weight of a potential closing transaction and therefore increases the on-chain fees. Redeeming very small outputs on-chain can be more expensive than their value. Therefore, BOLT specifications prescribe that nodes negotiate the *dust limit* before opening a channel. Nodes do not create HTLCs for payments below this limit. Such non-existent outputs are called *trimmed HTLCs* [57]. Out of the three most popular LN implementations, c-lightning and Eclair use the default dust limit of 546 satoshis. LND estimates the dust limit dynamically based on the current on-chain fees. We thus assume 546 satoshis as the dust limit.

[1]This Chapter is based on [375].

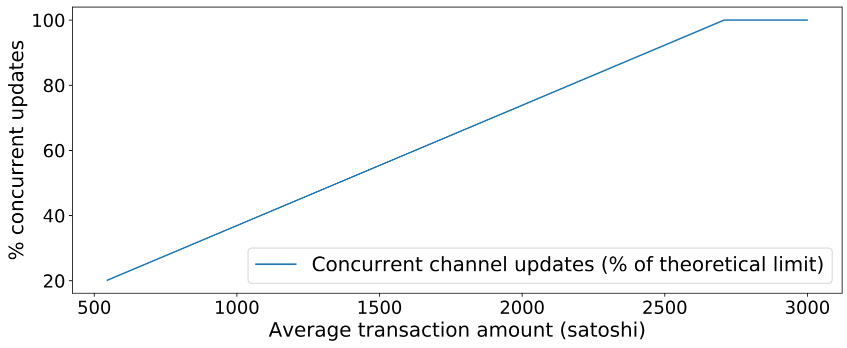


FIGURE 8.1: Ratio between the current limit on concurrent channel updates and the theoretically possible capacity-based limit.

8.2 The HTLC limit effect on LN scalability

This section estimates the effect of the HTLC limit on the number of concurrent channel updates. We use the same dataset *LN20* as in Chapter 7. We derive a series of historical snapshots representing the state of the LN on the first day of each month from April 2018 to February 2020. We refer to this dataset as *LNHist*.

Let C be the total network capacity (i.e., the sum of all channels' capacities). We only consider amounts above the dust limit D. Let a_{avg} be the average payment amount. We define the limit on concurrent updates based solely on capacity as $u_{cap} := C/a_{avg}$. In contrast, the limit on concurrent updates considering the HTLC limit is $u_{HTLC} = N * 966$, where N is the number of channels. We remark here that u_{HTLC} does not depend on payment amounts.

Given those two values, we define the *effective update rate* ur_{eff} as the ratio between the actual limit on concurrent payments considering the HTLC limit and the theoretical limit based solely on capacity:

$$ur_{eff} = \frac{min(u_{cap}, u_{HTLC})}{u_{cap}}$$

Note that the effective update rate ur_{eff} depends on the average payment amount, as shown in Figure 8.1. Starting from D, the effective number of concurrent updates diverges from what could be theoretically possible without the HTLC limit. We observe that 2 677 satoshis (0.27 USD) is the *borderline amount*: for higher average payment amounts, the limiting factor for the number of concurrent channel updates is channel capacity. For amounts between D and 2 677 satoshis, the limiting factor is the HTLC limit.

Affected channels The ur_{eff} is an aggregated measurement that does not shed light on how the HTLC limit affects individual channels. We now study how many channels are affected by the HTLC limit. The number of affected channels depends on the average payment amount a_{avg}. For high values of a_{avg}, it is more likely that the channel's capacity limits its effective update rate, whereas the HTLC limit determines the update rate cap for small values of a_{avg}. We quantify this as follows. Given a fixed average payment amount a_{avg}, we consider a channel *affected* by the HTLC limit if $u_{HTLC,a_{avg}} < u_{cap,a_{avg}}$, i.e, $u_{eff,a_{avg}} < 100\%$ (Figure 8.2).

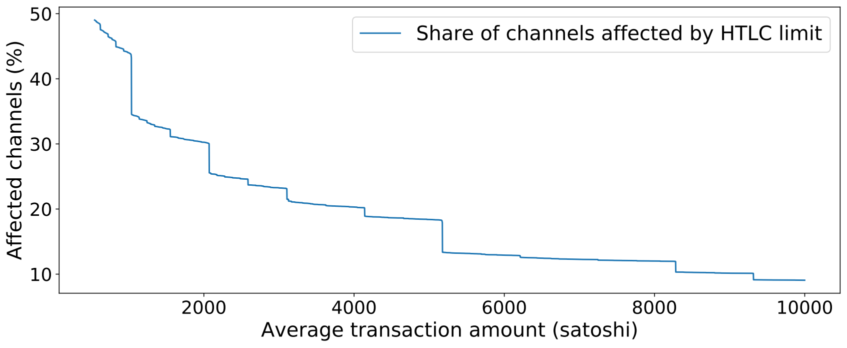


FIGURE 8.2: Share of affected channels for different payment amounts.

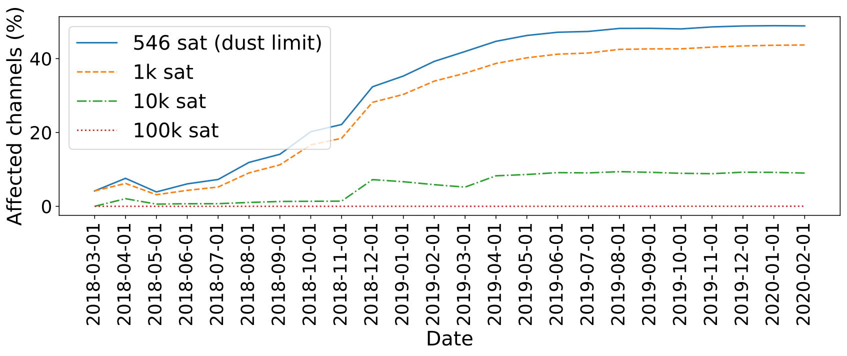


FIGURE 8.3: Historic share of HTLC-limited channels.

The effect of the HTLC limit over time We study the effect of the HTLC limit on the LN using our historical snapshots *LNHist*. For each monthly snapshot and four assumed average payment amounts, we calculate the share of channels affected by the HTLC limit (Figure 8.3). As expected, the HTLC limit becomes a more pressing issue with smaller payment amounts, if they are above the dust limit. We also observe that the share of affected channels has been increasing in the early months of the LN's existence and has remained stable since mid-2019.

We finally study how the *borderline amount* has changed over time. As Figure 8.4 shows, the HTLC limit finds its inflection point at payment amounts of approximately 2 500 satoshis, with the borderline amount stabilizing in mid-2019, after the initial growth.

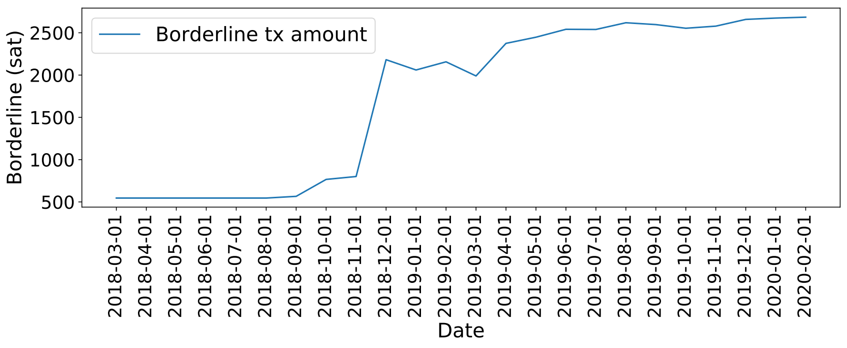


FIGURE 8.4: Historic borderline amounts.

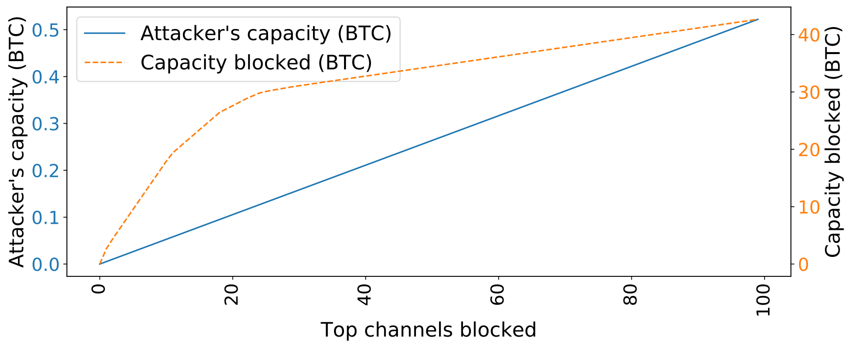


FIGURE 8.5: Effectiveness of targeting highest-capacity channels.

8.3 Depleting the Lightning Network

The HTLC limit enables a network-wide DoS attack. An adversary connects to both endpoints of the target channel and forwards multiple small payments to itself, but does not finalize them. After 966 HTLCs are added, the channel loses its ability to forward payments until some HTLCs expire. The attacker can thereby deplete a channel, making it unusable.

The cost of this attack depends on the minimum payment amount. We assume it equal to the dust limit of 546 satoshis (the default value in two out of three major LN implementations).

We calculate the total capital requirements for an attacker to block the LN completely. To block all 31 084 channels, the attacker would send, in the worst case, 966 payments of 546 satoshi to each channel. This brings the total capital requirements to approximately 163.9482 BTC (1.64M USD).

Each HTLC defines a timeout, after which the funds are returned to the sender if the receiver provides no preimage. From our dataset, we see that HTLC timeouts are long: 75.44 blocks on average. At a block creation rate of 10 minutes per block, this implies that an average HTLC can block the capacity for around 12 hours. Therefore, the attacker can render channels useless for around 12 hours using the same HTLC parameters as regular LN users.

While this rough upper bound estimate suggests a relatively high attack cost, the following optimizations make it more affordable.

Targeting highest-capacity channels The attack impact can be maximized by targeting the highest-capacity channels. For example, it requires 0.05 BTC to block 10 top channels with combined capacity of 17.91 BTC (Figure 8.5).

Real HTLC limit Our calculations are based on the maximum number of concurrent HTLCs (483) as defined by BOLT specifications. LN implementations may choose other values. In particular, Eclair and c-lightning enforce a lower default HTLC limit (30). Lower limits mean that in the real network, the attacker would need to create fewer HTLCs to block channels between c-lightning and Eclair nodes than between LND nodes (which by default support 483 concurrent HTLCs per channel). LND makes up 91% of the nodes in the network, and Eclair is another 1% [260]. That brings the real average HTLC limit to 442.23 and lowers the attack cost by 8.44%.

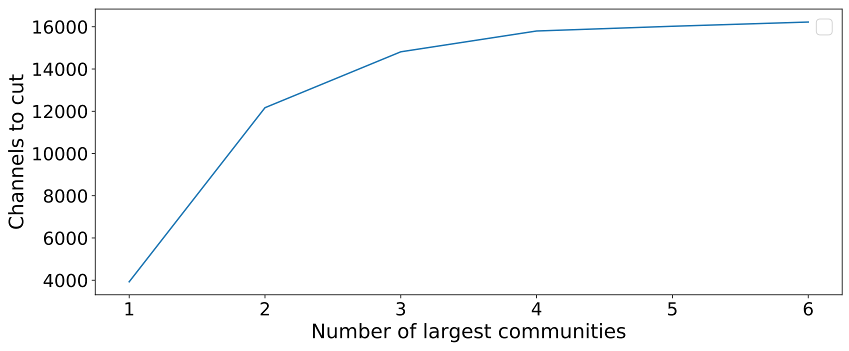


FIGURE 8.6: Number of channels to cut to isolate the largest communities.

Multi-hop payments Our estimation assumes single-hop payments. An attacker can leverage multi-hop payments to multiply the effect of the committed capital, connecting to both ends of a 20-hop [58] payment path and performing a payment to itself that never gets completed. This technique resembles capacity-based griefing attacks [291, 326] but entails much lower capital requirements.

Optimizing the attack based on communities The attacker may wish to prevent different parts of the network from transacting to each other. To evaluate this possibility, we first divide the network into *communities* using the Clauset-Newman-Moore greedy modularity maximization algorithm [92]. Then we consider a scenario where the attacker tries to block the channels that connect communities rather than channels within communities. For a chosen number N of the largest communities, we calculate how many channels the attacker has to block to split the network into at least $N + 1$ parts: the N largest communities and the rest of the network (Figure 8.6). We infer, e.g., that the attacker needs to block 4 670 channels (13% of all channels) to isolate the largest community from the rest of the network, locking up 25 BTC (250k USD) – or around 2.8% of the total LN capacity.

8.4 Discussion

Our simplistic model does not fully reflect all the details of payment handling. In particular, we do not account for multi-hop payments and payments that try multiple paths before succeeding. We do not reflect the unequal forwarding ability of a unit of capacity at a well-connected node, as opposed to a poorly connected one. We also do not account for unannounced channels, which may account for 28% of all channels [321]. Nevertheless, our approach allows us to calculate the effect of the HTLC limit, as we derive both estimations (capacity-based limit and HTLC limit) under the same assumptions.

The negative effect of the HTLC limit manifests itself at low payment amounts. This threatens potential LN applications involving micro-payments, such as paying for online content [300]. Our calculations show that for payments of 1 000 satoshis (0.1 USD), the network-wide rate of concurrent channel updates is 60% lower than it could have been based solely on capacity limitations.

The low value for the default minimum payment amount and the reduced number of in-flight payments open a DoS attack vector with a moderate cost for the adversary. Note that the capital in the attacker's channels will be recouped after the

HTLCs time out. Moreover, the unequal distribution of connectivity in the current LN paves the way for optimized attacks where the attacker focuses on high-capacity or inter-community channels to disrupt the transfer of value across the network.

8.5 Countermeasures

One of the limiting factors for payment throughput is the total available capacity. This limitation is overcome by opening new channels, a countermeasure naturally implemented with the growing LN adoption. The HTLC limit issue is more challenging as it comes from the limitations of the Bitcoin and Lightning protocols themselves. Therefore, more fundamental changes are needed to reduce the information required to carry out the functionality encoded in HTLCs. One countermeasure involves replacing HTLCs with atomic multi-hop locks (AMHL) [240]. While an HTLC requires a digital signature, hash value, and a timelock, AMHL only requires a digital signature and a timelock while providing the same functionality. This countermeasure would reduce the number of bytes required per in-flight payment and increase the number of payments handled concurrently. While not removing the limitation on the number of concurrent payments, this countermeasure raises this limit, reducing its harmful effect.

Another countermeasure is implementing payments across multiple channels with small *packetized payments* that are not secured by HTLCs [324]. Instead, this mechanism relies on economic incentives: if a counterparty misbehaves, an honest party stops the interaction and only loses a negligible value.

Firewall-like solutions have been proposed to protect Lightning nodes against HTLC-based DoS attempts [205].

8.6 Conclusion

We have quantitatively analyzed the negative effect on scalability produced by the limit on concurrent payments in the LN. The LN's limited concurrency implies that an adversary can block the complete network investing around 1.64M USD (18.5% of the network capacity). In comparison to [291], our HTLC depletion attack achieves the same result (a victim node cannot forward payments) but exploits the HTLC limit at each channel rather than its capacity. Compared to [260], we properly account for the way LN handles payments below the dust limit. The attack cost can be substantially reduced by targeting highly valuable channels (e.g., high-capacity channels or those connecting the network's largest communities).

Part III

Security of Ethereum smart contracts

Chapter 9

Introduction to smart contracts

In this Chapter, we provide an introduction to smart contracts and Ethereum.[1] We describe the history of the concept of smart contracts and its implementation in blockchain networks. We then provide the background on Ethereum – a blockchain-based smart contract platform – and outline its security and privacy challenges.

9.1 History of smart contracts

A contract is a fundamental building block of the market economy. The foundational mechanics of a contract has not changed in centuries. A contract is typically a written text interpreted by humans.

Financial contracts face major challenges in the digital era. First, contacts are becoming more complex. It is hard for humans to keep up with the speed and complexity of international finance. Second, contracts must be enforced. Usually, the government assumes the responsibility for contract enforcement. However, this introduces challenges in the global Internet.

9.1.1 Early ideas

The term *smart contract* originates from a 1997 essay by Nick Szabo [368]. The author argues that "contractual clauses [...] can be embedded in the hardware and software [...] to make breach of contract expensive." A vending machine exemplifies a primitive smart contract. The machine receives cash and dispenses goods and change according to the listed price. The physical properties of the machine make attacking it sufficiently expensive to be unprofitable. A vending machine contract is automated but still requires trust. Moreover, the administrator has privileged access to the money and goods in the machine.

9.1.2 Smart contracts in Bitcoin

The idea of digital smart contracts remained dormant until the introduction of Bitcoin. Bitcoin, for the first time, enabled *trustless* digital contracts. A user can only spend coins by providing the required arguments to their respective output scripts. Not only do Bitcoin scripts execute as programmed, but every user can independently verify this. Therefore, after a contract is created, users no longer have to trust any administrator.

This key property of Bitcoin has lead to the first attempts at encoding complex contracts in Bitcoin outputs. Early Bitcoin-based protocols towards this goal include Colored Coins [330], Counterparty [100, 23], and Mastercoin [405].

[1]This Chapter is partially based on [374].

Bitcoin's programming capabilities are too restrictive to encode many types of contracts. First, Bitcoin script is not Turing-complete. It means, for instance, that it does not support loops. Second, Bitcoin contracts cannot access other contracts. Each transaction is executed in isolation and can either be deemed valid or not. It is impossible to branch depending on the results of executing another transaction. On the one hand, such constraints limit the potential attack surface and simplify security analysis. On the other hand, they make implementing many types of contracts difficult, if not impossible [258].

9.1.3 Smart contracts in Ethereum

Ethereum is a blockchain platform that aims to address Bitcoin's limitations regarding programmability. It was announced in 2014 [70, 407] and launched in 2015.

Ethereum differs from Bitcoin in three major ways. First, Ethereum provides a richer programming environment. An Ethereum smart contract is written in a Turing-complete programming language and stored at a unique address. Contracts can be *called* by users or by other contracts. Each transaction can have complex effects in addition to moving cryptocurrency units. This interoperability allows for composing contracts. Second, Ethereum has updatable and addressable *storage*. Ethereum maintains consensus on the *state* of all accounts. Contracts can permanently store arbitrary data on-chain. Third, Ethereum implements an account-based model, as opposed to Bitcoin's UTXO-based model. Explicitly modeling accounts simplifies the contract logic. For instance, an Ethereum node can perform a single lookup to find out the current balance of an address, instead of scanning its whole history.

Bitcoin and Ethereum represent different points in the blockchain design space. The differences between these two dominant open blockchain networks are often a subject of heated debate. Some proponents of Bitcoin are eager to point out the questionable architectural decisions in the core Ethereum protocol and the vulnerabilities discovered in Ethereum smart contracts. Members of the Ethereum community emphasize a faster evolution of Ethereum and a wider range of applications it powers.

Simple systems are usually more secure than complex ones. The more interactions parts of the system engage in, the more things can break or be exploited. Ethereum's rich programming environment leaves more possibilities for programmers to make mistakes. Indeed, multiple high-profile smart contracts have been hacked, losing tens of millions of US dollars. These unfortunate events have shown that secure programming practices are crucial for smart contracts.

9.2 Ethereum architecture

We now provide a short introduction to the architecture of Ethereum.

9.2.1 Accounts

Ethereum can be thought of as a state machine. Ethereum nodes communicate through a peer-to-peer network and maintain a shared view of the global state, which consists of *accounts* with their respective *states*. Accounts belong to one of two types: externally owned accounts and contract accounts. An *externally owned account* is controlled by a private key. A *contract account* is controlled by a *smart contract*. Each account has a *balance* and a *nonce*. The balance is the amount of the native cryptocurrency *ether* controlled by this account. Internally, balances are expressed in *wei* – the smallest denomination of ether.[2] The nonce is a counter that keeps track of the number of transactions issued from the account, preventing replay attacks.

Apart from balance, contract accounts have *code* and *storage*. Contract code is a piece of bytecode for the Ethereum virtual machine (EVM). Contract storage is a mapping of arbitrary variable names to their values. The contract code controls the changes to the balance and storage of the account.[3] Internally, account data is stored in Merkle Patricia trees – radix trees with 256-bit keys [398, 67].

9.2.2 Transactions

Users interact with Ethereum by issuing *transactions*. A transaction represents a proposed state transition.

A transaction includes the following fields:

- *nonce* – the number of transactions sent by the sender;
- *gasPrice* – the number of wei per gas unit that the sender is paying;
- *gasLimit* – the maximum amount of gas to be spent during execution;
- *to* – the destination address (`0x0` for contract creation transactions);
- *value* – the number of wei transferred along with the transaction;
- v, r, s – signature data.

An Ethereum transaction belongs to one of two categories. A *message call transaction* executes a function of an existing contract or transfers ether. It contains the arguments for the function call in an optional *data* field. A smart contract function called by a transaction can, in turn, call other functions in this or other contracts. A *contract creation transaction* deploys a new contract. It contains an *init* field – a byte array containing the EVM bytecode of the new contract and the initialization code executed once on contract creation.

Transactions incur fees to prevent resource abuse. Every computational step in EVM is priced in units of *gas*. A transaction sender specifies the *gas limit* and the *gas price*. The *gas limit* is the maximum amount of gas that the transaction is allowed to consume. The *gas price* is the amount of ether the sender wants to pay per unit of gas consumed. Therefore, the maximum transaction fee (in ether) equals the gas limit multiplied by the gas price. If the execution is successful, the remaining ether is refunded.

EVM executes transactions *atomically*. A failed transaction does not affect the state[4] but consumes all provided gas. Ethereum specification (the Yellow paper [407]) defines gas costs of EVM opcodes. Developers occasionally change them in protocol upgrades. The market determines the price of a gas unit in ether. The *block gas limit* bounds the amount of gas consumed in one block. Miners can vote to gradually change this limit [208].

[2] 1 ether = 10^{18} wei.

[3] The balance of any account can also be increased by mining or by transferring the balance of another contract destroyed with `selfdestruct`.

[4] It still increases the nonce value for the sender's account and is included in a block.

9.2.3 Mining and coin issuance

Ethereum uses proof-of-work as a Sybil resistance mechanism. Miners pick unconfirmed transactions from the P2P network, serialize them, and apply them to the current state. They then solve a PoW puzzle based on the new state. Ethereum uses a memory-hard hash function Ethash for PoW [146]. Miners often use GPUs, though ASICs have also been introduced [283].

Miners are rewarded with transaction fees and block subsidy. Initially, Ethereum issued 5 ether per block. The block subsidy decreased to 3 ether in 2017 and 2 ether in 2019. Contrary to Bitcoin, the Ethereum issuance rate is not hard-coded into the protocol [151]. The core developers discuss proposed changes in the issuance schedule within a governance process.

Ethereum aims at a short average interval between blocks: 15 seconds, compared to 10 minutes in Bitcoin. Without specific countermeasures, short block intervals lead to a high *orphan rate*. An *orphan* is a valid block that is not included in the main chain. Network delays lead to miners unwillingly generating multiple blocks on the same height. Only one of these blocks is eventually accepted, whereas the hash power spent on other blocks is essentially wasted. In a double-spend attack, the attacker would only have to re-generate the PoW for the main chain, but not for orphan blocks. A short time between blocks exacerbates the problem.

To address the issue, Ethereum uses the modified [229] GHOST protocol [358, 123]. In Ethereum, orphan blocks are refereed to as *uncle* blocks, or *uncles*. Miners include hashes of uncles in block headers to receive a higher reward. The protocol rewards uncles no more than 6 blocks old. No more than 2 uncles per block are allowed. Miners of uncles whose headers get included in the main chain are also rewarded.

9.2.4 Contracts

Solidity [356] is the most popular high-level language for Ethereum smart contracts.[5] It is a statically typed language with a Javascript-like syntax. Listing 9.1 provides an example of a program in Solidity.

Developers compile Solidity code and deploy the resulting bytecode. Users then interact with it by issuing transactions.

Ethereum nodes execute contract bytecode on the *Ethereum virtual machine* (EVM). EVM bytecode is a low-level Turing complete stack-based language operating on 256-bit words. The EVM's design goals differ from those for general-purpose virtual machines such as the Java virtual machine (JVM). For instance, EVM executes deterministically and natively supports relevant cryptographic operations [71].

```
pragma solidity 0.4.17;
contract StringStorageContract {
string private str = "Hello, world!";
function getString() public constant returns (string) {
return str;
}
function setString(string _str) public {
str = _str;
}
}
```

LISTING 9.1: A simple contract in Solidity

[5]Vyper is an alternative language in an earlier stage of development [399]. Other alternative high-level languages, such as Serpent [347] and LLL [140], have been largely abandoned.

9.2.5 Applications

Crowdfunding is arguably the first widespread application of smart contracts [251]. This use case is exemplified by the so-called *initial coin offerings* (ICO). A *token* is a unit of new cryptocurrency built on top of Ethereum. An ICO is a crowdfunding mechanism whereby a team sells tokens to fund project development. A token is usually implemented as a smart contract that maintains a list of balances. Users can transfer tokens by interacting with the contract. ERC20 [397] is the de-facto standard way to implement a token on Ethereum. In 2017, ICOs attracted 1.8 billion USD [94], surpassing early stage venture capital funding [365]. Many ICOs turned out to be dubious or outright scams. However, Ethereum was proven useful as a global crowdfunding platform.

In 2019, another category of Ethereum-based applications known as *decentralized finance* (*DeFi*), gained momentum. DeFi projects build financial services, such as loans, in a more trustless way, eliminating the traditional intermediaries. The basic building block for DeFi is a *stablecoin* – a token with the value linked to a fiat currency, usually the US dollar. MakerDAO is a prominent *algorithmic* stablecoin project. It maintains price stability by providing economic incentives to market participants to restore the dollar parity if the exchange rate starts to diverge from it.[6] Other popular DeFi projects include Uniswap (a decentralized exchange) and Compound (a lending platform). Applications of Ethereum also include decentralized file storage [362] and computation [183], name systems [143], and prediction markets [17, 182].

9.3 Challenges for smart contracts

In 2016, an unknown hacker exploited a high-profile Ethereum contract called *The DAO* [350] and appropriated around 40 million USD worth of ether. Ethereum developers proposed a non-backwards-compatible protocol upgrade to return the funds. This move sparked controversy because the EVM had behaved correctly. It was the logical fault in the contract code that caused the calamity. Most community members accepted the proposal. A dissident minority continued to support the original chain (Ethereum Classic [147]). In 2017, an estimated 150 million USD were lost in two attacks against the Parity MultiSig contract [288]. These and other attacks on smart contracts [117, 16] show the importance of tools to ensure the correctness and security of smart contracts. Let us list some of the most pressing security challenges in this area.

Code security First of all, smart contracts must precisely reflect the developers' intent, which may not always be the case in practice. Smart contract programming languages, such as Solidity, are unfamiliar to developers. The execution model of Ethereum differs from centrally managed environments. Developers might not be used to their code being executed by a global network of anonymous, mutually distrusting, profit-driven actors. Some argue that the Solidity language itself inclines programmers towards unsafe development practices [413]. Moreover, the Ethereum network executes bytecode, not Solidity code. The security thus depends on the compiler and the EVM runtime environment.

[6]As of 2020, the most widely used stablecoin is Tether (also known as USDT). In contrast to MakerDAO, it is centrally managed. We refer the reader to [91, 220] for an overview of stablecoins.

Approaches to smart contract security include systematizing good and bad programming practices [148, 88], designing general-purpose [197, 72, 292] and domain-specific [133] smart contract programming languages, formalizing smart contracts execution model [346]. Multiple tools aim at improving the security and correctness of Ethereum smart contracts [33, 239, 198, 196, 387, 207, 352, 244, 271, 131]. See [344] for an overview of approaches to smart contract programming.

Bug-fixing and updates Smart contracts cannot be patched, which presents a trade-off [303]. On the one hand, smart contracts should guarantee fairness. Ideally, nobody should be able to change the contract code after deployment. This immutability ensures that privileged parties cannot unexpectedly change the rules of the game in their favor. On the other hand, smart contracts are experimental software. Deployed contracts are likely to contain bugs. An adversary can then exploit contract vulnerabilities without restraint.[7] Real-world projects often address this issue by encoding an administrator key in their contracts. The owner of this key can perform a limited number of actions. For example, an administrator might be able to pause withdrawals without being able to steal users' funds.

Anonymous attackers Exploiting smart contracts is attractive compared to other types of cybercrime. First, smart contracts store significant amounts of value. Second, digital assets are more liquid than, e.g., data from compromised databases.[8] Third, the risk of punishment is relatively low.

External data sources Some smart contracts rely on external data to operate. Many potential use cases envision smart contracts that depend on the data from the "real world," e.g., financial quotes. External information can only get into a smart contract from a transaction. Centralized data providers (also known as *oracles*) are a potential point of failure. This issue is known as the *oracle problem*. Multiple designs of trust-minimized oracles have been proposed. Mechanisms to ensure data authenticity include TLS-based cryptographic guarantees [307], trusted hardware [417], and economic incentives [80].

Front-running and miner influence Miners determine which transactions and in which order are included in a block. For certain applications, such as trading, transaction ordering is crucial. Miners and other users are incentivized to engage in *front-running*: prioritizing their transactions at the expense of others. It has been shown that automated bots do perform front-running in Ethereum-based decentralized exchanges [106, 325].

Privacy Smart contacts introduce new privacy challenges. On the one hand, an account-based model inclines users to perceive their Ethereum address as their semi-permanent identity, which harms privacy. On the other hand, richer programming capabilities enable sophisticated privacy solutions built as Ethereum smart contracts. For instance, the possibility to verify zero-knowledge proofs on-chain enabled the development of advanced protocols such as Aztec [18] and Tornado Cash [382].

[7]Smart contract immutability introduces a new dimension into the debate on responsible disclosure [340]. It is ethical to disclose a smart contact vulnerability publicly, if the developers have no technical means to fix it?

[8]However, the know-your-customer procedure that most exchanges require, and blockchain analytics, has made cashing out more difficult.

Chapter 10

Findel: Secure derivative contracts for Ethereum

The financial industry lacks a universal domain-specific language. The inherent ambiguity of natural language leads to misinterpretation of contracts. Multiple proposals have been drafted to create a rigorous domain-specific language (DSL) to mitigate disputes and stimulate automated processing of contracts. One approach is to leverage ideas from functional programming [293, 366, 169, 172, 343, 342]. These languages use a succinct set of basic building blocks to express financial agreements. Primitive contracts are combined using well-defined operators. A key feature of this approach is composability. New indefinitely complex derivative contracts can be defined based on existing ones. Due to their nested structure, contracts in these languages are well-suited for automated processing and valuation. However, an external enforcement mechanism is still required.

Blockchain networks present an environment for automated contract enforcement. Blockchains have fueled interest in the concept of smart contracts [78], theoretically described in the 1990s [368]. However, general-purpose smart contract languages, such as Solidity, are error-prone [350, 16].

In this Chapter, we introduce a declarative language for financial derivatives on top of Solidity.[1] Based on ideas from [293], we describe **Findel** (Financial Derivatives Language) – an Ethereum-based financial DSL. Findel unambiguously describes contract clauses. A user only defines what a contract *is* ("I owe you $10 tomorrow"), not *how* it is executed ("if the timestamp is greater than $t_0, \dots$"). The entire execution logic is implemented inside a smart contract, which is executed on-chain. Thus, we take the best of both worlds: unambiguity and composability of a concise declarative DSL, and trustless execution of a blockchain network. We implement an Ethereum smart contract that acts as a marketplace for Findel contracts and measure the cost of its operation.[2] We refer the reader to [202, 338] for a review of financial DSLs and to [344, 90] for a review of approaches to smart contract programming languages.

10.1 Findel contracts

Definition 1 *A **Findel contract**[3] C is a tuple (D, I, O), where D is the **description**, I is the **issuer**, and O is the **owner** (collectively called parties).*

[1]This Chapter is based on [38].

[2]See the related source code at `https://github.com/cryptolu/findel`.

[3]We may refer to Findel contracts simply as contracts, when the distinction between them and Ethereum smart contracts is clear from the context.

Definition 2 *A **description** of a Findel contract is a tree with **basic primitives** as leaves and **composite primitives** as internal nodes. The following BNF grammar defines primitives:*

$\langle basic \rangle$ ::= `Zero` | `One` ($\langle currency \rangle$)

$\langle scale \rangle$ ::= `Scale` ($\langle number \rangle$, $\langle primitive \rangle$)

$\langle scaleObs \rangle$ ::= `ScaleObs` ($\langle address \rangle$, $\langle primitive \rangle$)

$\langle give \rangle$ ::= `Give` ($\langle primitive \rangle$)

$\langle and \rangle$ ::= `And` ($\langle primitive \rangle$, $\langle primitive \rangle$)

$\langle or \rangle$ ::= `Or` ($\langle primitive \rangle$, $\langle primitive \rangle$)

$\langle if \rangle$ ::= `If` ($\langle address \rangle$, $\langle primitive \rangle$, $\langle primitive \rangle$)

$\langle timebound \rangle$ ::= `Timebound` ($\langle timestamp \rangle$, $\langle timestamp \rangle$, $\langle primitive \rangle$)

$\langle composite \rangle$ ::= $\langle scale \rangle$ | $\langle scaleObs \rangle$ | $\langle give \rangle$ | $\langle and \rangle$ | $\langle or \rangle$ | $\langle if \rangle$ | $\langle timebound \rangle$

$\langle primitive \rangle$ ::= $\langle basic \rangle$ | $\langle composite \rangle$

We distinguish between composite and basic primitives. Composite primitives contain other primitives as sub-nodes, basic primitives do not. *Currency*, *number*, *address*, and *timestamp* are implementation dependent data types. D and I cannot be modified after contract creation.

A financial company typically has templates for standard contracts. Parties who wish to sign an agreement write their names on a copy of a template and sign it, making it unique and legally binding. In our model, Findel contracts represent signed copies while their descriptions represent blank templates.

Traditional contracts usually contain clauses that regulate sub-ideal situations, i.e., a breach of contract. Findel does not distinguish between "ideal" and "sub-ideal" situations. All right and obligations are expressed uniformly. Section 10.3.1 discusses issues related to contract enforcement. Table 10.1 informally defines the primitives' execution semantics.

Table 10.2 illustrates the composability of Findel. INF is a symbol representing infinite time, i.e., $t_0 < INF$ for every t_0. δ is an implementation-dependent constant intended for handling imperfect precision of time in distributed networks.

10.1.1 Execution model

Findel contracts have the following lifecycle:

1. The first party **issues** the contract by specifying D, becoming its issuer. This is a mere declaration of the issuer's desire to conclude an agreement and entails no obligations.

2. The second party **joins** the contract, becoming its owner. As a result, both parties accept certain rights and obligations.

3. The contract is **executed** immediately as follows:

   (a) Let the root node of the contract's description be the current node.

TABLE 10.1: Findel contract primitives.

Primitive	**Informal semantics**
Basic	
Zero	Do nothing.
One($currency$)	Transfer 1 unit of $currency$ from the issuer to the owner.
Composite	
Scale(k, c)	Multiply all payments of c by a constant factor k.
ScaleObs($addr, c$)	Multiply all payments of c by a factor obtained from $addr$.
Give(c)	Swap parties of c.
And(c_1, c_2)	Execute c_1 and then execute c_2.
Or(c_1, c_2)	Give the owner the right to execute c_1 or c_2 (not both).
If($addr, c_1, c_2$)	If b is true, execute c_1, else execute c_2, where b is a boolean value obtained from $addr$.
Timebound(t_0, t_1, c)	Execute c, if the current timestamp is within $[t_0, t_1]$.

TABLE 10.2: Examples of custom Findel contracts.

Contract	**Definition**
At(t, c)	Timebound($t - \delta, t + \delta, c$)
Before(t, c)	Timebound(now, t, c)
After(t, c)	Timebound(t, INF, c)
Sell($n, CURR, c$)	And(Give(Scale(n, One($CURR$))), c)

(b) If the current node is either `Or` or `Timebound` with $t_0 > now$, postpone the execution: issue a new Findel contract with the same parties and the current node as root. The owner can later demand its execution.

(c) Otherwise, execute all sub-nodes recursively.[4]

(d) Delete the contract.

The execution outcome is fully determined by description D, execution time t, and external data $\mathcal{S}$ retrieved at time t.

10.1.2 Example

Suppose Alice sells to Bob a zero-coupon (i.e., making no periodic interest payments) bond that pays \$11 in one year for \$10:

$$c_{zcb} = \text{And}(\text{Give}(\text{Scale}(10, \text{One}(USD))), \text{At}(\text{now+1 years}, \text{Scale}(11, \text{One}(USD))))$$

[4]In case of `Or`, execute exactly one of the sub-nodes, according to the owner-submitted value indicating the choice; delete the other one. It is the only primitive that requires a user-supplied argument for execution.

We now show how c_{zcb} is executed step by step.

1. And executes; Bob temporarily owns two new contracts:

Alice's contracts	
Alice's balance	100
Bob's contracts	$\text{Give}(\text{Scale}(10, \text{One}(USD)))$ $\text{At}(\text{now} + 1 \text{ years}, \text{Scale}(11, \text{One}(USD)))$
Bob's balance	10

2. Give executes; Alice owns a new contract:

Alice's contracts	$\text{Scale}(10, \text{One}(USD))$
Alice's balance	100
Bob's contracts	$\text{At}(\text{now} + 1 \text{ years}, \text{Scale}(11, \text{One}(USD)))$
Bob's balance	10

3. Scaled One transfers $10 go from Bob to Alice:

Alice's contracts	
Alice's balance	110
Bob's contracts	$\text{At}(\text{now} + 1 \text{ years}, \text{Scale}(11, \text{One}(USD)))$
Bob's balance	0

4. In one year Bob claims $11 from Alice:

Alice's contracts	
Alice's balance	99
Bob's contracts	
Bob's balance	11

10.2 Implementation

We develop an Ethereum smart contract, called the *marketplace*, that keeps track of users' balances and lets them create, trade, and execute Findel contracts. The Findel DSL is network-agnostic and can be implemented on top of any blockchain with sufficient programming capabilities.

10.2.1 Users and balances

We implement the objects introduced in 10.1 with `struct` data types `Description` and `Fincontract`. We also introduce the `User` type that contains the user's Ethereum address and balances in all supported currencies. Users, descriptions, and contracts are stored in their respective mappings (a generic key-value storage type in Solidity) in the marketplace's storage.

The ultimate effect of every financial agreement is changing the parties' balances. Contract clauses specify when and under what conditions it should occur. Each user is assigned an array of balances for each supported currency. Although easily implementable, this approach introduces a single point of failure: the marketplace holds users' deposits.

The only primitive that transfers value is One. The `enforcePayment` function implements its execution. It subtracts a given amount in a given currency from the issuer's balance and adds it to the owner's balance. Our current implementation does not enforce any constraints on users' balances and does not prevent them from building up debt.

10.2.2 Ownership transfer

Besides `issuer` and `owner` (see Definition 1), a `Fincontract` contains an auxiliary `proposedOwner` field. On contract creation, `issuer`, `owner`, and `proposedOwner` are initialized to `msg.sender`. To transfer ownership, the owner sets `proposedOwner` either to the address of the proposed new owner or to `0x0`. Only the proposed owner can (but does not have to) `join` the contract; `0x0` means anyone can do so.[5]

10.2.3 Data sources and gateways

Ethereum contracts are intentionally isolated from the broader Internet and cannot pull data from external data feeds [185]. The issue can be solved with asynchronous requests, which work as follows. A smart contract records an Ethereum event with the request parameters properly encoded. A daemon process at an Ethereum node listens for such events, parses the requests, and sends them to an external data source. The responses are then sent to the requesting smart contract on behalf of an Ethereum account affiliated with the daemon. The submitted data may be accompanied by a proof of authenticity (i.e., a digital signature on an approved public key).[6]

Financial derivatives often use external data. To prevent a malicious or careless user from creating a Findel contract using untrusted sources, we need to guarantee data authenticity.

Definition 3 *A **gateway** is a smart contract that conforms to the following API:*

- ***int getValue()*** *Get the latest observed value.*[7]
- ***uint getTimestamp()*** *Get the timestamp at which the latest value was observed.*
- ***bytes getProof()*** *Get the authenticity proof for the latest value.*
- ***update()*** *Update the value.*

A gateway connects to an external data source and stores the latest value observed along with observation time and, optionally, a cryptographic proof of authenticity. We do not specify the form of proof a gateway provides. Possible options include Provable [307], TLSNotary [378], and Realio [316].

The marketplace queries a gateway at execution time, if necessary. If the value is fresh and the proof is valid, the execution proceeds, otherwise it is aborted, and all changes are reverted. Since a Findel contract may use multiple gateways, the owner is advised to `update` them all shortly before the execution.

A possible improvement would be for a gateway to store the latest observed value and a sequence of historical data. Multiple data points would allow for more straightforward modeling of certain derivatives, such as barrier options (execute

[5]Beware of front-runners: Bob can monitor the network and try to join a contract as soon as he sees Alice's attempt to do so. Depending on the network latency and miner's behavior, either transaction can be confirmed.

[6]BTCRelay is a prominent example: users submit Bitcoin block headers to a smart contract, which implies their authenticity from the validity of easily verifiable proof-of-work. After a header is stored on the Ethereum blockchain, users check with a Merkle proof that the Bitcoin block contains a given transaction.

[7]For simplicity, we only consider 256-bit integers as observable values. Boolean values can be trivially simulated via integers.

either c_1 or c_2, depending on whether an observable value touches a pre-defined threshold between acquisition and maturity).

We assume that the original data sources (e.g., feeds of reputable financial media) are trustworthy. An extra safety catch would be to query multiple sources, exclude outliers, and return an aggregated value. A secure connection (e.g., TLS) and the existing public key infrastructure (PKI) guarantee the authenticity of data sources.[8] Gateways without publicly available source code should not be trusted.

10.2.4 Execution

The `executeRecursively` function implements the execution logic (see Section 10.1.1) and returns `true` if executed completely (without creating new contracts) and `false` otherwise. The execution of an expired contract ($t_1 < now$) returns `true` unconditionally and deletes the contract.[9] Every step in the lifecycle of a Findel contract issues a system-wide notification (`Event`), allowing users to keep track of contracts they are interested in.

Our implementation deviates from the model (Section 10.1) in that the execution of contracts is not guaranteed. Ethereum contracts cannot act independently: the owner must issue a transaction to trigger contract execution. The owner may be unable to do so due to either opportunistic behavior or technical problems, such as loss of connectivity or lack of ether. Thus, we presume that Findel contracts are not guaranteed to execute.[10]

We model unbounded Findel contracts (i.e., with INF as the upper time bound) using a global $expiration$ constant inside the marketplace contract. Every Findel contract in the Ethereum implementation can only be executed within $expiration$ time units after creation (e.g., ten years).

10.3 Possible improvements

We now discuss the shortcomings of our model and ways to improve it.

10.3.1 Enforcement

As mentioned in Section 10.2.4, Findel contracts are not guaranteed to execute. At first sight, it is a major problem, as a contract must impose obligations on parties. In traditional finance, law enforcement is ultimately responsible for punishing violators. The closest we can arguably get to enforcement is a conditional penalty implemented inside a Findel contract itself.

Assume Alice issues, and Bob joins the following contract:

$$C = Before(t_0, \text{Or}(\text{Give}(\text{One}(USD)), \text{Give}(\text{One}(EUR))))$$

[8]See [170] and [230] for a discussion on blockchain-based PKI architectures.

[9]By definition, an expired contract is equivalent to Zero. An expired contract should also be deleted even if its owner is offline forever. Our current implementation does not handle the latter case, though it may be considered an attack vector due to increasing storage usage. A possible approach is for a marketplace to offer rewards for keeping track of expired contracts and triggering their deletion.

[10]Compare with [293]: "If you acquire *(c1 or c2)* you must immediately acquire either *c1* or *c2* (but not both)". We cannot force a user to make this decision.

C obliges Bob to give Alice either \$1 or 1€ before time t_0. If Bob fails to make a choice on time, Alice does not get the money she was planning to receive.[11] To prevent it, Alice attaches a "penalty" clause:

$$P = After(t_0, \text{If}(c_{executed}, \text{Zero}, \text{Scale}(2, \text{One}(USD))))$$

$c_{executed}$ is the address of a gateway indicating whether a Findel contract has been executed. When Bob joins $C_{penalty} = \text{And}(C, \text{Give}(P))$, Alice obtains the right to claim \$2 from Bob if he fails to fulfill his obligations.

Note that $C_{penalty}$ references $C_{executed}$, which in turn must be aware of $C_{penalty}$. Thus, the gateway should be either adjustable (with Alice tuning the gateway with a special transaction) or generic (reports the state of a Findel contract taking its identifier as an argument).

10.3.2 Defaulting on debt

A concise financial DSL does not prevent borrowers from defaulting on their debt. It is up to a marketplace to solve this problem.

Requiring a 100% guarantee deposit seems safe, but is questionable from an economic standpoint. People and organizations borrow money to invest it. The no-arbitrage principle states that no strategy guarantees a profit. The investor reward, e.g., interest, is the premium for taking the inevitable risk of business failure.

A marketplace can also mimic the fractional reserve banking model by requiring users to be always able to pay at least n% of their debt and punishing violators (e.g., by withholding their guarantee deposit). It does not solve the problem of defaults, however. In legacy finance, users have a fixed government-issued identity, allowing banks to maintain a shared credit history database. In a decentralized setting, users can create a practically indefinite number of identities. A production-ready marketplace should, therefore, take measures to combat Sybil attacks.

2020 update In the three years since the original paper's publication [38], the market has shown that fully collateralized (and *over*collateralized) loans do have useful applications. Ethereum-based stablecoin projects allow users to take loans in cryptocurrencies promised to hold parity with US dollar [259]. A prominent example is a stablecoin called *DAI*. To take out a loan of X DAI, users deposit at least $1.5X$ worth of cryptocurrency, usually ether, into a smart contract. If the *collaterization ratio* falls below the threshold, the position is *liquidated*: the collateral is forcibly sold. This system is useful for hedging against price volatility of cryptocurrencies and leveraged betting on their future price increase.

10.3.3 Modeling balances with Tokens

A more refined approach to modeling users' balances implies using ERC20 tokens. We assume that tokens can be freely exchanged to any currency the marketplace operates with. Given the address $\mathcal{T}$ of the Ethereum token contract, any Ethereum contract can query the balance of any user U, and transfer its tokens (if it has any) to an arbitrary address. Suppose Alice and Bob are token holders. Alice calls a standard API function `approve` to allow Bob to withdraw a certain amount of tokens

[11] In this case, an equivalent contract $\text{Give}(\text{Or}(\text{One}(USD), \text{One}(EUR)))$ solves the issue. In more complex cases, such measure may be insufficient.

from her account. Bob later calls `transferFrom` to transfer the tokens. The transfer succeeds if Alice has enough funds.

We suggest the following procedure. The issuer of a Findel contract approves the marketplace with the number of tokens they are potentially liable with. The marketplace implements `enforcePayment` by calling `transferFrom`, thus trying to withdraw tokens from the issuer and send them to the owner. For the execution to complete, the owner must either have enough tokens in the account or execute another Findel contract to fill it up. Thus, we delegate the banking functionality to the token smart contract and free the marketplace from holding and transferring money [215].

10.3.4 Multi-party contracts

We might want to extend the Findel contracts model to support more than two parties. An example of a three-party contract is buying a car with insurance. A user can only buy a car while simultaneously signing an insurance contract. We can express the two contracts (buyer – car dealer, buyer – insurance company) in Findel DSL, but executing them atomically is non-trivial. A possible way would be to use a gateway that keeps track of the state of Findel contracts. If *insuranceSigned* indicates whether a user joined the insurance contract, then buying with insurance looks like this (assuming CAR is a token representing the ownership over a car):

$$\text{If}(\textit{insuranceSigned}, \text{And}(\text{Give}(\text{Scale}(P, \text{One}(USD)), \text{One}(CAR))), \text{Zero})$$

10.3.5 Local client

To communicate with a Findel marketplace, users need client-side software. Besides communicating with the Ethereum network, it might also implement other functions:

- create and store Findel contracts locally;
- calculate the current value and other properties of Findel contracts based on assumptions about external data (e.g., the € / $ exchange rate is between 1.0 and 1.2) or valuation techniques such as the lattice binomial model [101];
- keep track of relevant Findel contracts and perform actions depending on their state (e.g., if c_1 gets executed, join c_2);
- store a predefined list of addresses of trusted gateways, similar to a list of trusted certificate authorities in web browsers.

10.4 Platform limitations

A Turing-complete programming language does not mean that all algorithms can be implemented inside an Ethereum contract. Apart from gas costs, the Ethereum network architecture imposes other limitations.

Lack of precise clock Almost all financial contracts contain temporal clauses. Clock synchronization is a challenging problem in decentralized systems, even more so if

TABLE 10.3: Cost of setup and helper functions (gas units).

Operation	Transaction cost	Execution cost
Create a marketplace smart contract	2 221 599	1 681 095
Register a user	79 462	58 190
Check user's balance	47 667	26 395
Get contract info	24 407	959
Get description info	24 706	1 258
Update a gateway	36 922	15 650

participants can profit from manipulating timestamps. Blocks in Ethereum are produced every 15 seconds on average. Block numbers provide causal ordering. Solidity contains keywords for time units, but miners ultimately control block timestamps.

Imperative paradigm The functional programming paradigm is well suited for developing embedded DSLs [180]. The original papers by Peyton Jones et al. and all existing implementations of their DSL use functional languages (Haskell [293, 209, 363], OCaml [231], Scala [401, 83]). In contrast, Solidity is imperative. Functional languages for Ethereum [171] are in an early stage of development.[12]

A limited type system The type system in Solidity is limited. Ethereum supports neither decimal nor floating-point types,[13] which often model amounts of money and currency exchange rates, respectively. The only numeric data types in Solidity are integers of various bit lengths. Moreover, Solidity lacks generic types, which could be useful for Gateways (i.e., `Gateway<int>`).

10.5 Gas costs

Ethereum users pay for every computational step in units of *gas*. Despite the use of expensive permanent storage operations, the cost of running our implementation is not prohibitively high for a proof-of-concept.

We measure gas costs of managing common Findel contracts as assessed by the Browser-solidity compiler [320][14] for a marketplace supporting two currencies (called USD and EUR in this example). The difference between transaction and execution cost is that the former includes the overhead of creating a transaction (i.e., a call from a client) and the latter does not (i.e., a call from another contract) [322].

10.5.1 Setup and helper functions

Registering a user implies initializing the user's balances to zero for all supported currencies. For testing purposes, we implement a gateway that uses the current timestamp as a data source and calculates a single `keccak256` hash as a dummy authenticity proof.

[12]In 2020, Solidity is the dominant high-level language in Ethereum. Functional contract languages have seen virtually no progress.
[13]A likely rationale: rounding issues break consensus.
[14]Solidity version: 0.4.4+commit.4633f3de.Emscripten.clang.

TABLE 10.4: Cost of handling derivatives in Findel (gas units).

Operation	Create and issue		Join and execute	
	Tx cost	**Exec cost**	**Tx cost**	**Exec cost**
One	184 239	177 967	58 493	93 602
Currency exchange (fixed rate)	663 149	656 877	101 878	138 430
Currency exchange (market rate)	300 842	294 570	59 822	96 196
Zero-coupon bond	373 783	367 511	143 891	201 750
Bond with two coupons	939 566	933 294	346 871	477 100
European option	519 628	513 356	278 191	411 103
Binary option	402 359	396 087	59 826	96 204

10.5.2 Managing common derivatives

In our measurements, we omit cases where parties split the execution cost. We assume that the issuer only pays for contract creation and issuance, whereas the owner pays for the execution. For simple Findel contracts, two Ethereum transactions (one from each party) represent the Findel contract's whole life cycle. When a contract executes in multiple steps, we sum up all costs that the owner bears to execute it. We also do not account for gateway update costs.

As of January 2017, the gas cost 10^{-9} ether per unit [152]; the price of ether fluctuated around 10 USD [408]. That brings the cost of a typical Findel contract operation (10^5 – 10^6 gas units) to 1.8 – 18 US cents.

2020 update As of September 2020, the cost of execution in Ethereum increased significantly. The average gas price is around 100 Gwei (10^{-7} ether) per unit of gas [149]. The price of ether is around 360 USD. Assuming the same gas costs, an operation with a Findel contract in 2020 would cost between 3.6 and 36 USD.

10.6 Conclusion

Smart contracts in public blockchain networks seem to be a perfect environment for implementing financial agreements. The unique value proposition of blockchains is trustless execution, which reduces counterparty risks. We have introduced Findel – a declarative financial DSL built upon ideas from prior research in financial engineering. Formalizing contract clauses using Findel makes them unambiguous and machine-readable. We prove Ethereum to be a suitable platform for trading and executing Findel contracts.

Nevertheless, the whole smart contract field is still in its infancy. Programmers who wish to implement a usable smart contract for handling financial agreements need to be aware of the future challenges: from fundamental limitations of the blockchain network architecture to imperfect development environment.

10.7 Examples

- A **fixed-rate currency exchange**: the owner sells 10 € for \$11.

$$\text{And}(\text{Give}(\text{Scale}(10, \text{One}(EUR))), \text{Scale}(11, \text{One}(USD))$$

- A **market-rate currency exchange**: the owner sells 10 € at market rate as reported by the gateway at *addr*.

$$\text{Scale}(10, \text{And}(\text{Give}(\text{One}(EUR)), ScaleObs(addr, \text{One}(USD))))$$

- A **zero-coupon bond**: the owner receives \$100 at t_0.

$$\text{Timebound}(t_0 - \delta, t_0 + \delta, \text{Scale}(100, \text{One}(USD)))$$

- A **bond with coupons**: the owner receives \$1 000 (face value) in three years (maturity date) and two coupon payments of \$50 at regular intervals before the maturity date.

$$\text{And}(\text{At}(\text{now + 3 years}, c_{face}), \text{And}(\text{At}(\text{now + 1 years}, c_{cpn}), \text{At}(\text{now + 1 years}, c_{cpn})))$$

where

$$c_{face} = \text{Scale}(1000, \text{One}(USD)), \quad c_{cpn} = \text{Scale}(50, \text{One}(USD))$$

- A **future** (a **forward**[15]): parties agree to execute the underlying contract c at t_0.

$$\text{Timebound}(t_0 - \delta, t_0 + \delta, c)$$

- An **option**: the owner can choose at (European option) or before (American option) time t_0 whether to execute the underlying contract c.

$$\text{Timebound}(t_0 - \delta, t_0 + \delta, \text{Or}(c, \text{Zero}))$$

$$\text{Timebound}(now, t_0 + \delta, \text{Or}(c, \text{Zero}))$$

- A **binary option**: the owner receives \$10 if a predefined event took place at t_0 and nothing otherwise.

$$\text{If}(addr, \text{Scale}(10, \text{One}(USD)), \text{Zero})$$

[15] In traditional finance, a future is a standardized contract while a forward is not. This distinction is not relevant for our model.

Chapter 11

SmartCheck: Static analysis of Ethereum smart contracts

In this Chapter, we study the security of smart contracts in Solidity.[1] We provide a comprehensive classification of code issues in Solidity and propose SmartCheck – an extensible static analysis tool.[2] SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. We evaluate our tool on a big dataset of real-world contracts, and compare its output with the results of a manual audit on three contracts. SmartCheck shows significant improvements in false discovery rate (FDR) and false negative rate (FNR) over existing alternatives. A static analyzer should be an essential part of the contract developers' toolbox. Automated detection of simple bugs lets developers allocate more effort to complex issues.

11.1 Classification of issues in Solidity code

We classify Solidity code issues as follows (based on [192]):

- **Security** issues lead to exploits by a malicious user account or contract;
- **Functional** issues cause the violation of the intended functionality;[3]
- **Operational** issues lead to run-time problems, e.g., bad performance;
- **Developmental** issues make code difficult to understand and improve.

We differentiate between functional and security issues: the former pose problems even without an adversary (though an external malicious actor can aggravate the situation), while the latter do not. Our primary sources are [97, 356, 16, 117, 88, 285]. Table 11.1 presents a summary of all issues.

11.1.1 Security issues

11.1.1.1 Balance equality

Avoid checking for strict balance equality: an adversary can forcibly send ether to any account by mining or via `selfdestruct`.

[1]This Chapter is based on [377]. The author of this thesis contributed to researching and formalizing the types of vulnerabilities and writing the paper. The author thanks Evgeny Marchenko for the help in adding "2020 updates" to reflect the changes in the applicability of the patterns since the original publication.

[2]The source code is available at [353].

[3]Though without a specification we only assume what the intended functionality is.

TABLE 11.1: Code issues detected by SmartCheck.
Gray background – false positives possible.

Name	Severity	Description
Balance equality (11.1.1.1)	medium	An adversary manipulates contract logic by forcibly sending it ether. Use non-strict inequality on balances.
Unchecked external call (11.1.1.2)	high	The return value is not checked. Always check the return values of functions.
DoS by external contract (11.1.1.3)	high	Expect external calls to deliberately `throw`.
`send` instead of `transfer` (11.1.1.4)	medium	The return value of `send` should be checked. Use `transfer`, which is equivalent to `if (!send()) throw;`.
Re-entrancy (11.1.1.5)	high	External contracts should be called after all local state updates.
Malicious libraries (11.1.1.6)	low	Using external libraries may be dangerous. Avoid external code dependencies, audit the whole code of the project.
Using `tx.origin` (11.1.1.7)	medium	A malicious contract can act on a user's behalf. Use `msg.sender` for authentication.
Transfer forwards all gas (11.1.1.8)	high	`a.call.value()()` forwards all gas, allowing the callee to call back. Use `a.transfer()`: it only provides the callee with 2300 gas (insufficient for a callback).
Integer division (11.1.2.1)	low	The quotient is rounded down. Account for it, especially for ether and token amounts.
Locked money (11.1.2.2)	medium	The contract receives ether, but there is no way to withdraw it. Implement a function to withdraw or reject payments.
Unchecked math (11.1.2.3)	low	Without extra checks, integer over- and underflow is possible. Use SafeMath.
Timestamp dependence (11.1.2.4)	medium	Miners can alter timestamps. Make critical code independent of the environment.
Unsafe type inference (11.1.2.5)	medium	Type inference chooses the smallest integer type possible. Explicitly specify types.
Byte array (11.1.3.1)	low	`byte[]` requires more gas than `bytes`.
Costly loop (11.1.3.2)	medium	Expensive computation inside loops may exceed the block gas limit. Avoid loops with a high or unknown number of steps.
Token API violation (11.1.4.1)	low	The contract `throws` where the ERC20 standard expects a `bool`. Return `false` instead.
Compiler version not fixed (11.1.4.2)	low	Contract compiles with future compiler versions. Specify the exact compiler version.
`private` modifier (11.1.4.3)	low	The `private` modifier does not hide the variable's value, only prevents external contracts from editing it.
Redundant fallback function (11.1.4.4)	low	The payment rejection fallback is redundant. Remove the function to save space: payments are rejected automatically.
Style guide violation (11.1.4.5)	low	Unfamiliar capitalization style causes confusion. Start function names with lowercase, events with uppercase.
Implicit visibility level (11.1.4.6)	low	Functions are `public` by default. Avoid ambiguity: explicitly declare visibility level.

```
if (this.balance == 42 ether) { /* ... */} // bad
if (this.balance >= 42 ether) { /* ... */} // good
```

The pattern detects comparison expressions with == that contain `this.balance` as either left- or right-hand side.

11.1.1.2 Unchecked external call

Expect calls to external contract to fail. When sending ether, check for the return value and handle errors. The recommended way of doing ether transfers is `transfer` (see Section 11.1.1.4).

```
addr.send(42 ether); // bad
if (!addr.send(42 ether)) revert; // better
addr.transfer(42 ether); // good
```

The pattern detects an external function call (`call`, `delegatecall`, or `send`) that is not inside an `if`-statement.

11.1.1.3 DoS by external contract

A conditional statement (`if`, `for`, `while`) should not depend on an external call: the callee may permanently fail (`throw` or `revert`), preventing the caller from completing the execution.

In the following example, the caller expects the oracle to return an integer value (`badOracle.answer()`). However, the actual oracle implementation may throw an exception in some or all cases.

```
function dos(address oracleAddr) public {
  badOracle = Oracle(oracleAddr);
  if (badOracle.answer() < 42) { revert; }
  // ...
}
```

This rule contains multiple patterns:

- an `if`-statement with an external function call in the condition and a `throw` or a `revert` in the body;
- a `for`- or an `if`-statement with an external function call in the condition.

11.1.1.4 `send` instead of `transfer`

The recommended way to perform ether payments is `addr.transfer(x)`, which automatically throws an exception if the transfer is unsuccessful, preventing the problem described in Section 11.1.1.2. The pattern detects the `send` keyword.

2020 update As of 2020, the best practice is that `call` is preferred to `send` and `transfer` (see comment in 11.1.1.8).

11.1.1.5 Re-entrancy

Consider the following code:

```
pragma solidity 0.4.19;
contract Fund {
  mapping(address => uint) balances;
  function withdraw() public {
    if (msg.sender.call.value(balances[msg.sender])())
    balances[msg.sender] = 0;
  }
}
```

The contract at `msg.sender` can get multiple refunds and retrieve all `Fund`'s ether by recursively calling `withdraw` before its share is set to `0`. Besides, it can change the state of some third contract that `Fund` depends on. Use the "checks – effects – interactions" pattern: first check the invariants, then update the internal state, then communicate with external entities (see also Section 11.1.1.4):

```
function withdraw() public {
  uint balance = balances[msg.sender];
  balances[msg.sender] = 0;
  msg.sender.transfer(balance);
  // state reverted, balance restored if transfer fails
}
```

The pattern detects an external function call followed by an internal function call.

11.1.1.6 Malicious libraries

Third-party libraries can be malicious. Avoid external dependencies or ensure that third-party code implements only the intended functionality. The pattern detects the `library` keyword (thus producing some false positives).

11.1.1.7 Using `tx.origin`

Contracts can call each other's public functions. `tx.origin` is the first account in the call chain (always an externally owned one, i.e., not a contract); `msg.sender` is the immediate caller. For instance, in a call chain `A` $\rightarrow$ `B` $\rightarrow$ `C`, from the `C`'s viewpoint, `tx.origin` is `A`, and `msg.sender` is `B`.

Use `msg.sender` instead of `tx.origin` for authentication. Consider a wallet:

```
pragma solidity 0.4.19;
contract TxWallet {
    address private owner;
    function TxWallet() { owner = msg.sender; }
    function transferTo(address dest, uint amount) public {
        require(tx.origin == owner);  // authentication
        dest.transfer(amount);
    }
}
```

User sends ether to the address of the `TxAttackerWallet`, which forwards the call to a vulnerable implementation of `TxWallet` and obtains all funds, acting as the user (`tx.origin`):

```
pragma solidity 0.4.19;
interface TxWallet {
    function transferTo(address dest, uint amount);
}
contract TxAttackerWallet {
    address private owner;
    function TxAttackerWallet() { owner = msg.sender; }
    function() payable {
        TxWallet(target).transferTo(owner, msg.sender.balance);
    }
}
```

The pattern detects the environmental variable `tx.origin`.

11.1.1.8 Transfer forwards all gas

Solidity provides many ways to transfer ether (see Section 11.1.1.4). The function `addr.call.value(x)()` transfers `x` ether and forwards all gas to `addr`, potentially leading to vulnerabilities like re-entrancy (see Section 11.1.1.5). The recommended way to transfer ether is `addr.transfer(x)`, which only provides the callee with a "stipend" of 2 300 gas units. The pattern detects functions whose name is `call.value` and whose argument list is empty.

2020 update In 2019, Ethereum underwent the Istanbul upgrade. Among other modifications, gas prices for some operations were increased [134]. As a result, the call subsidy of 2 300 gas units is often insufficient to handle the call. Therefore, it is recommended to forward more than 2 300 gas (probably all available gas) when calling external contracts. In the light of this pattern, `transfer` is no longer preferred over `send`. Both these commands are considered undesirable. The preferred way to interact with external contracts is `call`.

11.1.2 Functional issues

11.1.2.1 Integer division

Solidity supports neither floating-point nor decimal types. For integer division, the quotient is rounded down. Account for it, especially when calculating ether or token amounts. The pattern detects division (/) where the numerator and the denominator are number literals.

11.1.2.2 Locked money

Contracts programmed to receive ether should implement a way to withdraw it, i.e., call `transfer`, `send`, or `call.value` at least once. The pattern detects contracts that contain a `payable` function but contain none of the withdraw functions.

2020 update In Solidity 0.5.0, the address type has been split into `address` and `address payable` [357]. Ether can only be sent to `address payable`. A contract developer is thus forced to consider where money can be sent from their contract. However, the issue captured in this pattern persists on the receiving side. It is still possible to write a contract that receives money but cannot send it elsewhere.

11.1.2.3 Unchecked math

Solidity is prone to integer over- and underflow.[4] Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account. Use the SafeMath library[5] that checks for overflows [286]. The pattern detects arithmetic operations `+`, `-`, `*` that are not inside a conditional statement. This rule has been muted for testing (Section 11.3) due to a high false positive rate.

[4]Referred to as overflow for brevity.
[5]See Section 11.1.1.6 for advice on library usage.

11.1.2.4 Timestamp dependence

Miners can manipulate environmental variables and are likely to do so if they can profit from it. Consider a lottery that distributes prizes depending on whether `now` (alias for `block.timestamp`) is odd or even:

```
  if (now % 2 == 0) winner = pl1; else winner = pl2;
```

A miner can tweak the timestamp and gain an unfair advantage. Use secure sources of randomness, such as RANDAO [314]. The pattern detects the environmental variable `now`.

11.1.2.5 Unsafe type inference

Solidity supports type inference: the type of `i` in `var i = 42;` is the smallest integer type sufficient to store the right-hand side value (`uint8`). Consider a `for`-loop:

```
for (var i = 0; i < array.length; i++) { /*...*/ }
```

The type of `i` is inferred to `uint8`. If `array.length` is bigger than `256`, an overflow occurs. Explicitly define the type when declaring integer variables:

```
for (uint256 i = 0; i < array.length; i++) { /*...*/ }
```

The pattern detects assignments where the left-hand side is a `var` and the right-hand side is an integer (matches `^[0-9]+$`).

2020 update Implicit type inference from `var` was deprecated in Solidity 0.5.0 [357]. The pattern no longer applies.

11.1.3 Operational issues

11.1.3.1 Byte array

Use `bytes` instead of `byte[]` for lower gas consumption. The pattern detects the construction `byte[]`.

11.1.3.2 Costly loop

Ethereum is a resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized cloud providers. Moreover, Ethereum miners impose a limit on the total number of gas consumed in a block. In the following example, if `array.length` is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed:

```
for (uint256 i = 0; i < array.length; i++) { costlyF(); }
```

This becomes a security issue, if an external actor influences `array.length`. E.g., if `array` enumerates all registered addresses, and registration is open, an adversary can register many addresses, causing denial of service. The rule includes two patterns:

- a `for`-statement with a function call or an identifier inside the condition;
- a `while`-statement with a function call inside the condition.

11.1.4 Developmental issues

11.1.4.1 Token API violation

ERC20 is the de-facto standard API for implementing *tokens* – transferable units of value managed by a contract. Exchanges and other third-party services may struggle to integrate a token that does not conform to it. Certain ERC20 functions (`approve`, `transfer`, `transferFrom`) return a `bool` indicating whether the operation succeeded. Throwing exceptions (`revert`, `throw`, `require`, `assert`) is not recommended inside these functions. Note that library functions may also throw exceptions (see Section 11.1.1.6).

```
function transferFrom(address _spender, uint _value)
returns (bool success) {
  require (_value < 20 wei);
  // ...
}
```

The pattern detects a contract inherited from a contract with a name including the word "token," which may throw exceptions from one of the named functions.

11.1.4.2 Compiler version not fixed

Solidity source files specify the versions of the compiler they can be compiled with:

```
pragma solidity ^0.4.19;  // bad: 0.4.19 and above
pragma solidity 0.4.19;   // good: 0.4.19 only
```

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in unforeseen ways. The pattern detects the version operator ˆ in the `pragma` directive.

11.1.4.3 `private` modifier

Contrary to a popular misconception, the `private` modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum. The pattern detects state variable declarations with a `private` modifier.

11.1.4.4 Redundant fallback function

Contracts should reject unexpected payments (see Sections 11.1.1.1 and 11.1.2.2). Before Solidity 0.4.0, it was done manually:

```
function () payable { throw; }
```

Starting from Solidity 0.4.0, contracts without a fallback function automatically revert payments, making the explicit `throw` redundant. The pattern detects the described construction (only if the `pragma` directive indicates the compiler version above or equal to 0.4.0).

11.1.4.5 Style guide violation

In Solidity, function names usually start with a lowercase letter[6] and event names start with an uppercase letter:

[6] Except for constructors: they must share the name with the contract and usually start with an uppercase letter.

```
function Foo(); // bad
event logFoo(); // bad
function foo(); // good
event LogFoo(); // good
```

Violating the style guide decreases readability and leads to confusion. The pattern detects the described constructions.

11.1.4.6 Implicit visibility level

The default function visibility level in Solidity is `public`. Explicitly define function visibility to prevent confusion.

```
function foo() { /*...*/ } // bad
function foo() public { /*...*/ } // good
function bar() private { /*...*/ } // good
```

The pattern detects function and variable definitions with no visibility modifier.

2020 update Since Solidity 0.5.0, a visibility modifier is mandatory for all functions [357]. The pattern no longer applies.

11.2 SmartCheck architecture

The two major approaches to code analysis are *dynamic* analysis and *static* analysis [233]. Dynamic analysis runs the program, while static analysis considers the program code without running it. Static analysis usually includes three stages:

1. building an intermediate representation (IR), such as abstract syntax tree (AST) or three-address code, for a deeper analysis, compared to analyzing text;
2. enriching the IR with more information [406] using algorithms such as control- and dataflow analysis (synonym, constant, and type propagation [4]), taint analysis [386], symbolic execution, abstract interpretation;
3. vulnerability detection w.r.t. a database of patterns that define vulnerability criteria in IR terms.

This work does not consider formal verification methods, as they require a rarely available formal specification of the contract's intended functionality.

SmartCheck is a static analysis tool implemented in Java. It runs lexical and syntactical analysis on the Solidity source code. It uses ANTLR [289] and a custom Solidity grammar to generate an XML parse tree [4] as an intermediate representation (IR). The tool detects vulnerabilities by using XPath [234] queries on the IR. Thus, SmartCheck provides full coverage: the analyzed code is fully translated to the IR, and all its elements can be reached with XPath matching. Line numbers are stored as XML attributes and help localize findings in the source code. IR attributes can be enriched with more information, as new analysis methods are implemented. The tool can be extended to support other smart contact languages by adding the corresponding ANTLR grammar and a pattern database. The IR-level algorithms remain unchanged.

As an example, consider the Balance equality issue (Section 11.1.1.1). We aim to detect constructions that test the contract balance for equality, for instance:

```
if (this.balance == 42 ether){...}.
```

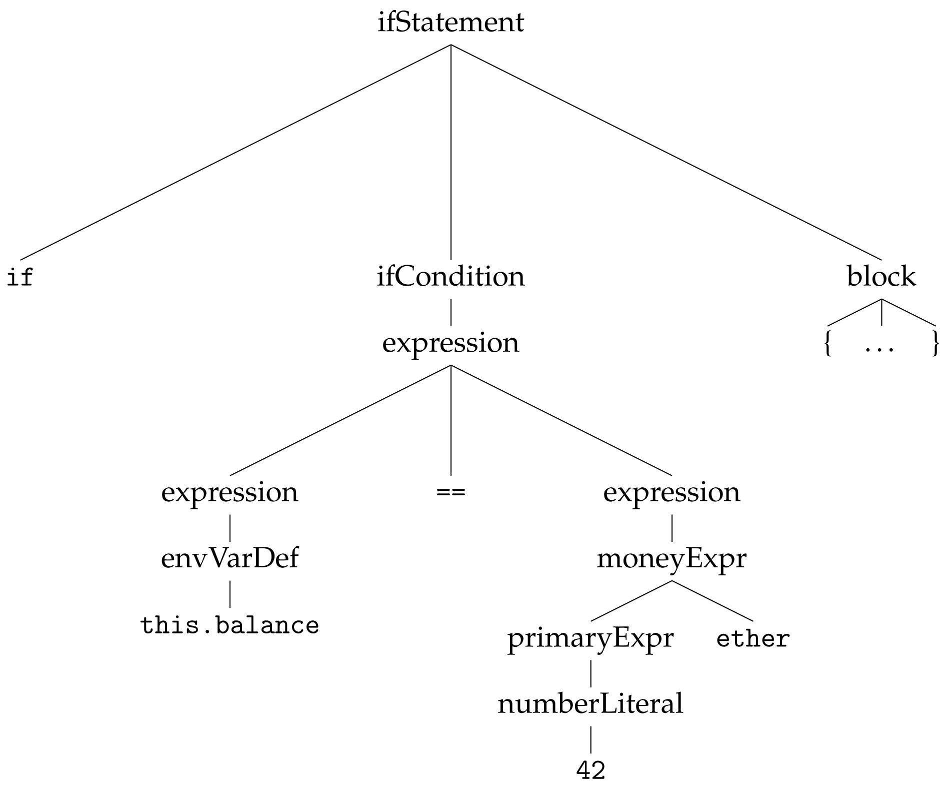


FIGURE 11.1: Parse tree for the Balance equality code example.

The parse tree of this construction is shown in Figure 11.1, and the corresponding XPath pattern is shown in Listing 11.1.

```
//expression[expression//envVarDef
[matches(text()[1],"^this.balance$")]]
[matches(text()[1],"^==|!=$")]
```

LISTING 11.1: XPath pattern for the Balance equality issue.

In this case, we do not expect false positives, as we can precisely describe the target construction in XPath.[7] More complex rules cannot be precisely described with XPath, which leads to false positives. Consider the Re-entrancy issue (Section 11.1.1.5). SmartCheck reports violations of the Checks-Effects-Interactions (CEI) pattern, which do not always lead to re-entrancy (Listing 11.2).

```
pragma solidity 0.4.19;
contract Foo {
  bool inBar = false;
  function bar(address someAddress) {
    if (inBar) throw;
    inBar = true;
    someAddress.transfer(0);
    inBar = false;
  }
}
```

LISTING 11.2: Violation of CEI not leading to re-entrancy.

[7]Assuming that ANTLR builds the AST correctly based on the Solidity grammar.

11.3 Experimental results

We evaluate SmartCheck in comparison to the results of manual audit (Section 11.3.2) and the three freely available vulnerability detection tools – Oyente, Remix, and Securify (Section 11.3.3).[8]

11.3.1 Definitions

We define a true finding as an issue (detected by a tool with manual verification or manually) that is a bad practice and should be fixed from our viewpoint. It may or may not be an exploitable vulnerability. All issues found by the tools are manually labeled as either true positive (TP) or false positive (FP). A false negative (FN) for each of the four tools (Oyente, Remix, Securify, and SmartCheck) is a true finding that is not detected by this tool.

For each tool, the false discovery rate (FDR) is the number of FPs for this tool divided by the number of all issues reported by this tool:

$$FDR = FP/(TP + FP)$$

False negative rate (FNR) is the number of FNs for this tool divided by the number of all true findings (found by any of the tools or manually):

$$FNR = FN/(TP + FN)$$

11.3.2 Case studies

We consider three contracts: Genesis ("the platform for the private trust management market" [176], source code [177], analyzed at commit `1ecf99d`), Hive ("the first crypto currency [sic] invoice financing platform" [199], source code [200], analyzed at commit `0d54699`), and Populous ("an online platform that matchmakes invoice sellers to invoice buyers hosted on the blockchain" [301], source code [302], analyzed at commit `10de4ae`). The FDR and FNR for each tool (in absolute numbers and as percentages) are presented in Table 11.2.

Oyente and Securify show no TPs on these three contracts. Remix detects TPs only in the Populous contract. Remix and SmartCheck show an overall FDR of 97% and 69% and an overall FNR of 92% and 47%, respectively. Overall, SmartCheck reports 87 issues in the three contracts.

Requirements for code analysis tools differ across platforms and domains. Due to special security requirements in smart contract programming, a low FN rate is crucial (a missed vulnerability can be disastrous), whereas a relatively high FP rate is tolerable. Most contracts contain a few hundreds of lines of code (see Section 11.3.3) and can be audited manually.

Though SmartCheck's FDR of 69% may seem high, it is not a serious issue in this domain. 47% is a reasonable level of FNR since many vulnerabilities in smart contracts are related to business logic and cannot be detected automatically. Most of SmartCheck's FNs are found manually (not by other tools).

SmartCheck detects a critical issue in one of the contracts. An attacker can create an unlimited number of internal entities and block the contract's normal operation. A public function (i.e., such that any Ethereum user can call it) adds an element to an internal array (Listing 11.3). Several critical functions then iterate through this

[8]For Securify, we only consider partial results from the publicly available version of the tool.

TABLE 11.2: Testing results on three selected projects.

Project		Oyente	Remix	Securify	SmartCheck
Genesis Vision	TP	0	0	0	7
	FP	6	40	19	22
	FN	10	10	10	3
	FDR (%)	100	100	100	75.86
	FNR (%)	100	100	100	30.00
Hive	TP	0	0	0	6
	FP	6	11	6	7
	FN	22	22	22	16
	FDR (%)	100	100	100	53.85
	FNR (%)	100	100	100	72.73
Populous	TP	0	4	0	14
	FP	7	60	45	31
	FN	19	15	19	5
	FDR (%)	100	93.75	100	68.89
	FNR (%)	100	78.95	100	26.32
Overall	TP	0	4	0	27
	FP	19	111	70	60
	FN	51	47	51	24
	FDR (%)	100	96.52	100	68.97
	FNR (%)	100	92.16	100	47.06

array (e.g., Listing 11.4). An attacker can make those functions permanently fail, as the function call would require more gas than the block gas limit.

```
function createGroup(string _name, uint _goal)
    onlyOpenAuction
    returns (uint8 err, uint groupIndex)
{
  if(checkDeadline() == false && _goal >= fundingGoal && _goal <=
      invoiceAmount) {
    groupIndex = groups.length++;
    groups[groupIndex].groupIndex = groupIndex;
    groups[groupIndex].name = _name;
    groups[groupIndex].goal = _goal;

    EventGroupCreated(groupIndex, _name, _goal);

    return (0, groupIndex);
  } else {
    return (1, 0);
  }
}
```

LISTING 11.3: Adding an element to an internal array.

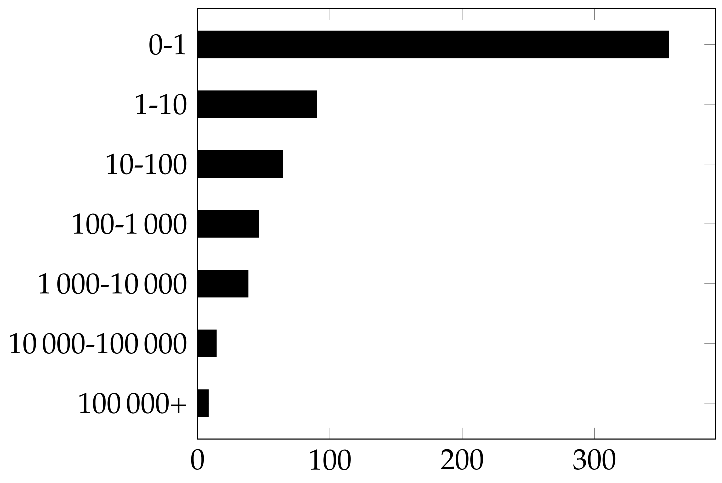


FIGURE 11.2: Distribution of non-zero contract balances (ether).

```
function findBidder(bytes32 bidderId) constant returns (uint8 err, uint
     groupIndex, uint bidderIndex) {
  for(groupIndex = 0; groupIndex < groups.length; groupIndex++) {
    for(bidderIndex = 0; bidderIndex < groups[groupIndex].bidders.
        length; bidderIndex++) {
      if (Utils.equal(groups[groupIndex].bidders[bidderIndex].bidderId,
           bidderId) == true) {
        return (0, groupIndex, bidderIndex);
      }
    }
  }
  return (1, 0, 0);
}
```

LISTING 11.4: Iterating through an internal array.

11.3.3 Testing on a massive sample

A *blockchain explorer* is a website that displays information about blockchain transactions. Etherscan [150] is a popular Ethereum blockchain explorer. Among other information, it offers *contract verification* as a service. A contract developer uploads the source code to Etherscan, which checks whether the deployed bytecode corresponds to the provided source code. We download the source code of 4 600 verified contracts (1 537 954 lines of code) from Etherscan as of 4 October 2017 using a Java library JSoup [210]. We then run SmartCheck on this dataset.

The contract balances differ significantly, with most contracts (3 984, or 86.6%) having a zero balance (Figure 11.2). One contract holds over one million ether (1 500 000, or 440 million USD at the time of testing), which accounts for 38.4% of the total balance of all contracts. Contracts have from 1 to 2 525 lines of code, with an average of 334 lines and a median of 221 lines.

SmartCheck analyzed the dataset in approximately 2 hours and 7 minutes.[9] As per SmartCheck, 99.9% of contracts have issues, 63.2% of contracts have critical vulnerabilities.[10] The findings are presented in Table 11.3 and Figure 11.3 (colors denote severity levels: black – high, dark gray – medium, light gray – low). The most

[9]7 644 seconds (437 lines per second) on Intel Core i5-4210M @ 2.60 GHz, 12 GB RAM, Windows 8.1 64 bit.

[10]The issues found by SmartCheck in the big dataset were not manually verified.

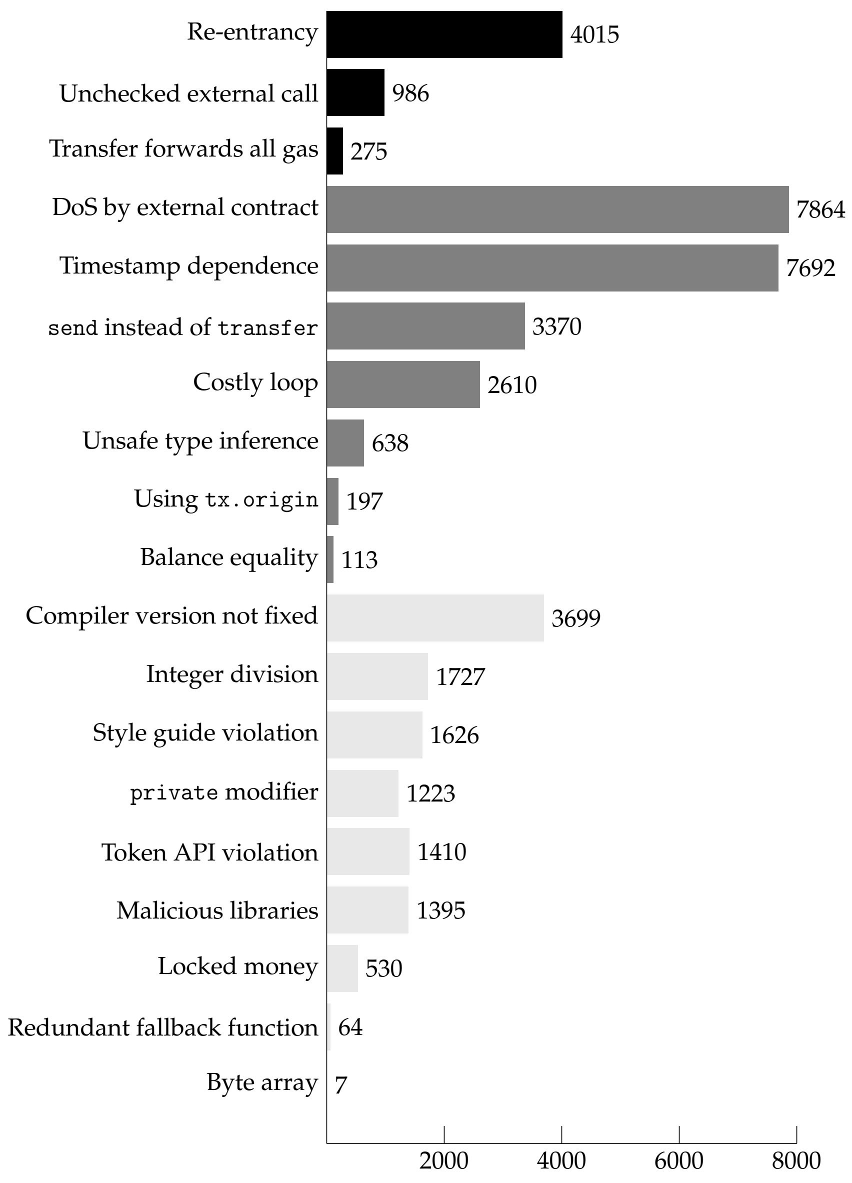


FIGURE 11.3: Findings on the big dataset (excluding Implicit visibility level).

TABLE 11.3: Code issues detected on the big dataset.

Severity	Pattern	Findings	% of all
high	Re-entrancy	4 015	3.329
	Unchecked external call	986	0.818
	Transfer forwards all gas	275	0.228
medium	DoS by external contract	7 864	6.521
	Timestamp dependence	7 692	6.378
	`send` instead of `transfer`	3 370	2.794
	Costly loop	2 610	2.164
	Unsafe type inference	638	0.529
	Using `tx.origin`	197	0.163
	Balance equality	113	0.094
low	Implicit visibility level	81 160	67.296
	Compiler version not fixed	3 699	3.067
	Integer division	1 727	1.432
	Style guide violation	1 626	1.348
	`private` modifier	1 223	1.014
	Token API violation	1 410	1.169
	Malicious libraries	1 395	1.157
	Locked money	530	0.439
	Redundant fallback function	64	0.053
	Byte array	7	0.006

prevalent issue, **Implicit visibility level** (detected 81 160 times, which accounts for 67.30% of all findings), is excluded from the figure for clarity.

11.4 Conclusion

We have provided a comprehensive overview and classification of code issues in Solidity – the primary high-level language for Ethereum smart contracts. We have implemented SmartCheck – an efficient static analysis tool for Solidity, which offers significant improvements over existing alternatives. We have tested our tool on a massive set of real-world contracts and detected code issues in most of them. The tool can be enhanced in multiple ways: improving the grammar,[11] making patterns more precise (e.g., the temporarily muted Unchecked math), adding new patterns, implementing more sophisticated static analysis methods, adding support for other languages.

Security is still an issue in blockchain development. We hope that SmartCheck will help solve this major challenge by providing smart contract developers with fast and relevant feedback on potentially problematic source code fragments.

[11] The currently used grammar failed to parse 0.16% of lines in our dataset.

Chapter 12

Privacy-preserving KYC on Ethereum

In this Chapter, we describe the current approach to identity management and propose KYCE – a privacy-preserving KYC scheme for token whitelisting on Ethereum.[1]

12.1 Identity

Digital identity is the information used by a computer system to represent a user. Access to services is controlled in two steps:

- authentication: a user proves that they are who they claim to be;
- authorization: the system ensures that the user has the right to perform the requested action.

To comply with regulations, financial institutions verify the identities of their customers. Modern finance depends on government-issued identities. Regulations in most jurisdictions demand that banks obtain proof of identity from customers before doing business with them – a procedure known as "know your customer," or KYC. "Anti money laundering" (AML) and "counter terrorist financing" (CTF) are related regulations that require banks to stop and report suspicious transactions.

Modern KYC practices weaken users' control over their personal information and threaten their privacy. Financial institutions store sensitive information in private databases, which become a target for corrupt employees or external hackers. Independent KYC/AML implementations lead to high compliance cost and multiply the risk of identity theft.

Open blockchains like Ethereum take a more decentralized approach to identity management. Users join these networks without any identification. Financial service providers establish consortia to apply blockchain technologies in their services [144, 203, 312]. To comply with regulation, they have to handle government-issued identities in a blockchain setting. This non-trivial task becomes more challenging, considering users' demands for stronger privacy protection. The European privacy regulation (GDPR [153]) that came into force in May 2018 poses more challenges for organizations that handle personal data.

[1]This Chapter is based on [39], which, in turn, described a hackathon project. The team consisting of Daniel Feher, Dmitry Khovratovich, Sergei Tikhomirov, Aleksei Udovenko, and Maciej Żurad implemented a proof-of-concept implementation in May 2017 during the Luxblock hackathon in Luxembourg and won a joint first prize. Contributions of the author of this thesis include: implementing parts of the hackathon project and writing the paper (except for Section 12.2.3).

In this work, we first explore the centralized and decentralized approaches to identity. We then propose KYCE – a privacy-preserving Ethereum-based KYC implementation. KYCE allows banks to implement KYC checks using an external smart contract – a KYC provider. Our scheme uses zero-knowledge proofs to check users' eligibility without disclosing their private information to anyone except the KYC provider. A smart contract stores the KYC whitelist in the form of a cryptographic accumulator. This construction allows users to be efficiently added to, removed from, and checked against the list without storing any plaintext data on the blockchain. We then discuss possible use cases and implementation challenges.

12.1.1 Centralized identity

In terms of asymmetric cryptography, identity I of user U is a public-private key pair $(pub_U, priv_U)$. The public key pub_U authenticates the user (or, equivalently, links their current action to some past actions). Public identifiers like username or address are derived from pub_U. The private key $priv_U$ allows U to sign messages on behalf of I. For the system, U is whoever possesses $priv_U$.

In the centralized identity model, prevalent on the Internet today, users delegate managing their private keys to a trusted party and use a password to access them when necessary. This approach is sub-optimal in many regards. First, users do not control their identities. The trusted party always has the technical ability to sign messages without the user's consent or to prevent the user from signing the message they want. Moreover, users' data is stored by a centralized entity, providing incentives for an attack. Finally, users have to create a new identity for each website they wish to register with. As a result, they adhere to a risky practice of reusing passwords. Third-party login protocols such as OAuth and OpenID [124] partially address this issue ("login with"). In this scheme, a website queries the service that holds the user's existing identity (e.g., Google) and asks for permission to access a subset of the user's data (e.g., name and email). This approach alleviates the password management problem but increases the impact of potential identity theft.

Though users can revoke access at any time, the "login with" scheme is still privacy-violating. Imagine a user who reveals their birth date to prove to a website that they are 18 years of age or older. If they later revoke the access, their date of birth will never change. Thus, they grant the third-party website effectively unlimited access to a piece of private information.

Maintaining correspondence between "real world" identities and public keys has long been a challenge. Widely deployed centralized solutions like PKI suffer from risks associated with centralization: a fraudulent authority can issue rogue certificates [7].

12.1.2 Decentralized identity and open blockchains

The PGP "web of trust" is a noteworthy attempt at creating a decentralized identity system [162]. It has not gained significant traction due in part to usability challenges [334] and concerns about the security of the long-term key model [394].

Bitcoin [273] eliminates the problem of connecting public keys to identities in a radical manner: a user may generate many public keys that *are* identities. Alternative blockchains such as Ethereum [70, 407] take a similar approach. The way open blockchains handle identity may come at odds with financial regulation. We propose a design that will simultaneously leverage the power of blockchain-based smart contracts, enable banks to implement KYC to comply with the law, and preserve users' privacy.

12.1.3 Financial and privacy regulation in the EU

The current EU legislation "on information accompanying transfers of funds" came into effect in 2015 [317]. In the wake of the rapid growth of cryptocurrencies, the EU is tightening its anti-money laundering regulations, stating that "virtual currency exchange platforms and custodian wallet providers will have to apply customer due diligence controls, ending the anonymity associated with such exchanges" [265]. See [395] for the analysis of virtual currencies under the EU AML law.

In 2018, two pieces of legislation came into force in the EU.

- The **Revised Payment Service Directive** (PSD2) obligates banks to provide access to their customers' accounts through open APIs [191]. This measure is meant to foster competition and give rise to third-party financial service providers. For instance, a unified banking API would simplify connecting banking infrastructure to open blockchains [138].
- The **General Data Protection Regulation** (GDPR) harmonizes data privacy laws across the EU [153] and introduces stricter rules for handling data of EU residents even for companies from outside the EU. We refer the reader to [31] describes possible implications of blockchain adoption from the viewpoint of the EU data protection regulation.

12.2 KYCE: a decentralized KYC-compliant exchange

KYC requirements differ depending on jurisdiction [310]. A typical KYC procedure links users' real-world identities to their accounts and checks users against a whitelist or a blacklist. The details of the KYC procedure do not affect our design.

12.2.1 Definitions and security properties

Definition 4 *A* ***KYC procedure*** *is a process that determines if a given user is eligible for a given transaction.*

Definition 5 *A* ***KYC provider*** *is an entity that performs a KYC procedure.*

Definition 6 *A* ***financial service*** *is an information system that allows users to exchange units of value.*

Definition 7 *A financial service is* ***KYC-compliant*** *w.r.t. the KYC procedure if and only if all users are eligible for all transactions they perform.*

Definition 8 *A KYC-compliant financial service is* ***privacy-preserving*** *if and only if only the KYC provider has access to the users' private data.*

12.2.2 Tokens and exchanges

Our KYC solution can be applied for any service. For concreteness, consider a token exchange as an example.

Definition 9 *A* ***token*** *is a transferable fungible unit of value maintained by a smart contract.*

ERC20 [397] is the de-facto standard API for implementing token contracts in Ethereum. A token contract keeps track of users' token balances and allows them to transfer tokens using the following functions:

- `transfer` sends a given amount of tokens to a given address;
- `approve` allows a given user to withdraw up to a given amount of tokens from the account of the user calling the function;
- `transferFrom` sends a given amount of tokens from one given address to another (the amount has to be `approved` beforehand).

Definition 10 *An* ***exchange*** *is a service that allows users to exchange tokens.*

Centralized exchanges, implemented as regular web services, are the most prevalent. We are mostly interested in decentralized, or on-chain exchanges, implemented as smart contracts.

An exchange without KYC support may be used as follows.

1. Alice creates an order to sell X A-tokens for Y B-tokens;
2. Bob creates an order to sell Y B-tokens for X A-tokens;
3. The exchange matches the two orders and transfers (by calling `transferFrom`) X A-tokens from Alice to Bob and Y B-tokens from Bob to Alice.

The transaction succeeds if Alice and Bob have `approved` the exchange with a sufficient amount of A- and B-tokens, respectively, before `transferFrom` is called. Users withdraw tokens from the exchange by calling `approve(exchangeAddress,0)`.

12.2.3 Privacy-preserving KYC

We propose KYCE – a privacy-preserving KYC design for Ethereum-based financial service providers. A KYC contract provides an API to other contracts so that external services can determine if a given user is KYC-approved for using a given token. A KYC provider (a governmental entity or company in charge of customer onboarding) performs the necessary checks for a new customer and adds their address to the whitelist.

A simple approach to implementing a KYC check with a separate contract would be the following. The KYC contract stores the whitelist of approved addresses. On every `transfer`, the token contract checks if the address belongs to the whitelist. This design has a fundamental privacy flaw: the contract stores all whitelisted addresses on-chain in plaintext. Moreover, users must use the same addresses they have registered with the KYC provider. Address reuse threatens privacy: an adversary can link the user's transactions using public blockchain data.

Our approach We use cryptographic techniques to design a privacy-preserving KYC solution. In KYCE, the KYC contract stores a **cryptographic accumulator** of the whitelisted addresses.

A cryptographic accumulator A absorbs certain algebraic objects and provides an interface to generate and verify zero-knowledge proofs that a given value has been accumulated. In our construction, to generate a proof for value $x \in A$, one needs a *witness* that depends on A and x. The accumulator owner provides the witness to the user who submitted x. We suggest an accumulator based on bilinear maps [75].

The KYC workflow is as follows. The KYC provider publishes a smart contract and initializes it with an empty accumulator. The User interacts with the KYC provider physically or online and provides the credentials needed to pass the verification. The User also generates their own master secret m and during the authenticated session, gives the provider a Pedersen commitment $g_1^m \cdot g_2^r$ to it. g_1 and g_2 are certain group generators,[2] and r is random. If the User passes the procedure, the provider updates the accumulator with user-dependent data and gives the User a witness. In every subsequent Ethereum transaction to KYCE, the User provides a proof that they have been registered in the accumulator, that this right has not been revoked, and that the proof owner and the transaction sender are one person. KYCE verifies the last statement. The KYC contract verifies the rest against the current accumulator value. If the checks pass, KYCE executes the requested action.

Details on the accumulator construction We construct an accumulator based on a pairing function $e(\cdot,\cdot)$ in some pairing setting.[3] The accumulator contains the serial numbers, possibly consecutive integers.[4]

The accumulator is constructed as follows. We assume a bilinear pairing $e: G \times G \rightarrow G_T$, where G, G_T are groups of order q. The KYC provider selects a generator g and a secret value $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q$. It also selects L as an upper bound of users enabled for KYC and computes $z = e(g,g)^{\gamma^{L+1}}$. It initialized the accumulator value A by 1.

Let us denote $g_i = g^{\gamma^i}$. The provider publishes A, $\{g_i\}_{1\leq i\leq L, L+2\leq i\leq 2L}$, the set of registered KYC indices $V = \emptyset$, and the parameters g, z needed for verification.

Every User who passes the KYC check is issued a new serial number i, the witness $w_i = \prod_{j\in V, j\neq i} g_{L+1-j+i}$, where V is the set of all issued serial numbers, and a signature σ_i of $g_i||i$ on the provider's private signature key. The witness is used to generate a proof of accumulating.[5] The KYC provider updates the accumulator with i by

$$\mathrm{A}_{\mathrm{V}\cup\{\mathrm{i}\}} \leftarrow \mathrm{A}_{\mathrm{V}} \cdot g_{L+1-i}$$

multiplying it by $g_{L+1-i} = g^{\gamma^{L+1-i}}$, and i is published as a new valid serial number. To prove that i has been committed to A and has not been revoked without disclosing it, the holder of w_i updates it[6] so that the following equation holds:

$$\frac{e(g_i, A)}{e(g, w_i)} = z.$$

[2]Here and in the further text all multiplications take place in the pre-selected group of prime order q, typically an elliptic-curve group.

[3]The original paper [75] uses type-1 pairings, but type-3 pairings can be adopted as well.

[4]It is possible to store public keys, but it would be less efficient.

[5]We refer an interested reader to [75] for the details.

[6]We omit the details, but the update can be performed just before the presentation, not necessarily after every accumulator update.

Note that revocation is also efficient. The KYC contract owner simply multiplies the accumulator value by the inverse of g_{L+1-i}. The witness value cannot be updated anymore.

Presentation When issuing a transaction to use the exchange (e.g., create an order), the user submits a **zero-knowledge proof** of the following statement:

- I know the private key of the current user address (`msg.sender`), and
- I know a signature σ_i and a witness w_i for some number i that has been accumulated in the accumulator A in the KYC contract.

This compound statement must be *atomic*, i.e., the sub-statements cannot be extracted as separate valid proofs, as this would make the transaction malleable.

The atomicity (and non-malleability) are ensured as follows. Let us denote the proof of knowledge for the witness and signature by PK_w. Then the Prover submits

$$P = \{PK_w \wedge PK_s\},$$

where PK_s is the proof of knowledge of the private key of the `msg.sender`'s ECDSA public key, which can be taken from [82]. The technique to make a composite proof of knowledge (PoK) is straightforward, as both PoKs are non-interactive, and is standard in complex PoK protocols:

1. The Prover collects a set $\mathcal{C}$ of commitments asserted in sub-proofs PK_w and PK_s.
2. The Prover makes necessary randomization of $\mathcal{C}$ to create t-values $\mathcal{T}$.
3. The Prover computes $c \leftarrow H(\mathcal{C}, \mathcal{T})$.
4. The Prover computes s-values $\mathcal{S}$ using $\mathcal{C}$, $\mathcal{T}$, and c.
5. The proof P is $(\mathcal{C}, \mathcal{S}, c)$. To verify it one computes asserted t-values $\widehat{\mathcal{T}}$ and verifies
$$c \stackrel{?}{=} H(\mathcal{C}, \widehat{\mathcal{T}}).$$

The resulting proof P is submitted as an Ethereum transaction argument. KYCE retrieves the current accumulator value and verifies P against it and the message sender's public key, available in the transaction metadata. If the proof is correct, the order is executed.

12.2.4 Use cases

Either the exchange contract or the token contract must be KYC-compliant – i.e., check the eligibility of transacting parties using the introduced cryptographic scheme using the KYC contract.

KYC-compliant exchange If the exchange is KYC-compliant, the tokens do not need to be aware of the KYC (Figure 12.1).

Consider an established exchange that trades dozens of tokens. It applies for official approval in a jurisdiction that requires all customers to pass the KYC procedure. The governmental body acts as a KYC provider, deploys a KYC contract, and publishes its address. The exchange adds KYC checks to its codebase and continues operation. Users who do not want to apply for KYC can withdraw their tokens from the exchange.

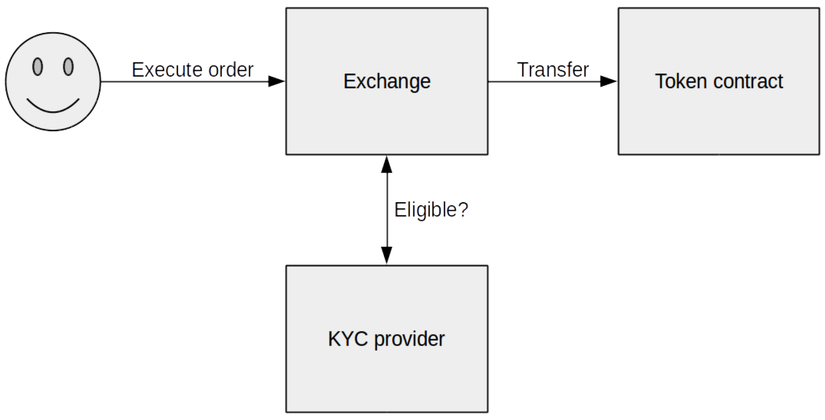


FIGURE 12.1: KYC-compliant exchange.

KYC-compliant token If the token is KYC-compliant, the exchange does not need to be aware of the KYC (Figure 12.2).

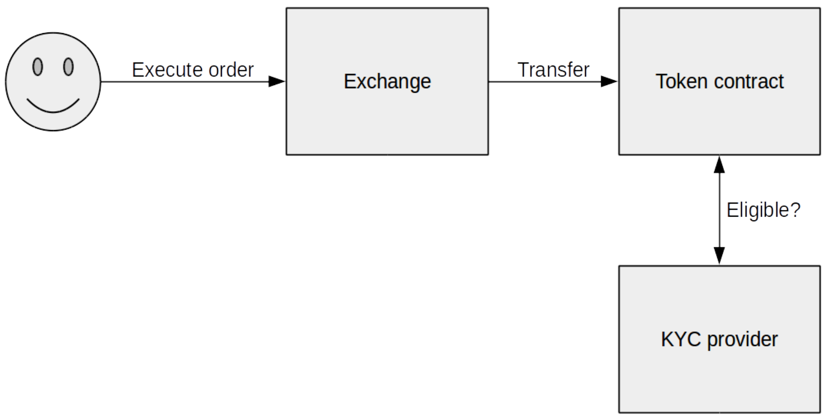


FIGURE 12.2: KYC-compliant token.

Consider a government that issues its own tokens.[7] KYC-approved users could use government tokens for tax payments, fees, and fines. Such a solution leverages smart contracts' flexibility and auditability while only allowing approved entities to use the token. The KYC-enabled government token can also be traded on exchanges, which would allow citizens to hold currency portfolios of their choice and only purchase government tokens to transact with the state.

Transaction-dependent checks Many jurisdictions impose restrictions that depend on the value of the transaction. E.g., the EU regulation [317] states that "the obligation to check whether information on the payer or the payee is accurate should [...] be imposed only in respect of individual transfers of funds that exceed 1 000 €". EU member states impose further restrictions for large transactions, e.g., exceeding 10 000 € in Belgium, 15 000 € in Germany and in the Netherlands [310]. Either the exchange contract or the token contract can perform such checks by storing the following mappings:

[7]Bank of England [110] and the Monetary Authority of Singapore [264] have already researched this direction.

- address => accumulated transaction volume in the current period (day, month, year);
- address => timestamp of the latest transaction.

12.3 Implementation details

We have created an initial (not privacy-preserving) implementation of the proposed design. Our project consists of two smart contracts written in Solidity: KycProvider and KyceToken. KycProvider maintains a 2-dimensional boolean array that stores the eligibility status across users and tokens. On initialization, the address that deploys the contract is appointed as the *owner*, allowing it to add and remove users from the whitelist. The ownership may be transferred (using the functionality inherited from the standard `Ownable` contract).

The KycProvider exposes the following API:

- `add(address _user, address _token)` makes the user eligible for using the token (callable only by the owner);
- `remove(address _user, address _token)` makes the user not eligible for using the token (callable only by the owner);
- `isEligible(address _user, address _token)` checks if the user is eligible for using the token.

KyceToken adheres to the de-facto standard token API in Ethereum – ERC20. To minimize the risk of security issues due to implementation subtleties, we inherit a widely used and tested ERC20 implementation by OpenZeppelin. We override the functions `approve`, `transfer`, and `transferFrom` to check if the given user (`msg.sender`) is eligible for using this token. If `isEligible` returns `false`, the execution stops. If it returns `true`, the corresponding function of the superclass is invoked.

The implementation of the proposed scheme requires certain cryptographic primitives. Some of them are partially available in Ethereum as pre-compiled contracts (elliptic curve addition, scalar multiplication, and pairing checks). For the proposed scheme to be fully implemented, pairing evaluation is also required.[8]

12.4 Related work

José Parra Moyano and Omri Ross use distributed ledgers to improve the KYC process [270]. Their proposal can be summarized as follows:

- the regulator maintains a database with all users' private data;
- the user signs a contract with their first bank (the *home bank*);
- the home bank stores hashes of the user's documents in a smart contract in a permissioned blockchain;
- when the user signs a contract with another bank it obtains the user's documents from the database and looks up the hash to ensure that the user has been KYC-approved;

[8] As of 2020, Ethereum does not support pairing evaluation.

- the identity of the home bank is not revealed;
- a cost-sharing mechanism for banks allows them to proportionally share the cost of the initial KYC approval.

In this design, all banks store users' private data – contrary to our solution, where it is stored only with the KYC provider. The authors also propose a more decentralized design but claim it to be of lesser practical relevance.

Clare Sullivan and Eric Burger investigate possible implications of further development of the Estonian e-residency program using blockchain technology [364]. E-residency of Estonia is a governmental program that provides applicants with a digital identity that can be used, e.g., to register a company and open a bank account. Estonian e-residency disconnects a digital identity from citizenship or physical residence. Within the e-residency program, Estonia collaborates with a blockchain project Bitnation [51, 130]. Provable (previously known as Oraclize) offers a connector that lets Ethereum contracts handle e-residency identities [307].

A project [282] similar to ours implements a KYC scheme using Ethereum, but stores the KYC status on-chain in plaintext. Multiple projects aim at easing customer onboarding for banks [74, 226, 355, 383]. Blockchain consortium R3 has developed a proof-of-concept implementation of a shared KYC between ten banks based on its blockchain platform Corda [6]. Multiple Ethereum-based identity projects have been proposed [255, 359, 390].

12.5 Conclusion

We have proposed a modular design of an Ethereum-based financial service with an external KYC check, which benefits all participants.

- **Users** obtain a unified identity that they can use with multiple financial services. Users' data is stored only with the KYC provider and can be easily updated. Personal data is neither stored on the blockchain nor transmitted to third parties.
- **Financial services** greatly simplify the KYC process: it boils down to a single API call. Our design lets them cut KYC costs while at the same time diminishing risks of handling sensitive data.
- **Governments** get an opportunity to stimulate innovation in the financial sector by providing a unified and simple KYC API, which is especially relevant in rapidly growing fintech and blockchain industries.

Our design is agnostic to the nature of the entity behind the KYC contract. It does not have to be a government body. The proposed solution can be used in any setting where a smart contract based service wants to limit the set of its users. For instance, many jurisdictions (e.g., the US [391]) only allow certain investments to be offered to "accredited investors." These are typically high-net-worth individuals and financial institutions. This logic can be replicated in a blockchain setting. Consider a blockchain-based financial service that only accepts cryptocurrency users who possess more than 10 000 USD and have done their first transaction before 2017. The "accrediting" functionality is delegated to a third party KYC provider. Proving net worth and previous activity on the blockchain is straightforward. More checks can be added. Once accredited, an investor can use multiple "restricted" services without revealing any personal details to their developers.

Bibliography

[1] 1ML. *Lightning Nodes - Top Channel Count*. 2019. URL: https://1ml.com/node?order=channelcount.

[2] 1ML. *Litecoin Lightning Network*. 2019. URL: https://1ml.com/litecoin/.

[3] 8BTCStaff. *Chinese Bitcoin miners headed to Central Asia?* 2020. URL: https://decrypt.co/20404/chinese-bitcoin-miners-headed-to-central-asia.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986. ISBN: 0-201-10088-6. URL: https://www.worldcat.org/oclc/12285707.

[5] J. Ayo Akinyele. *zkChannels for Bitcoin*. 2020. URL: https://medium.com/boltlabs/zkchannels-for-bitcoin-f1bbf6e3570e.

[6] Ian Allison. *R3 develops proof-of-concept for shared KYC service with 10 global banks*. 2016. URL: https://www.ibtimes.co.uk/r3-develops-proof-concept-shared-kyc-service-10-global-banks-1590908.

[7] Johanna Amann et al. "Mission accomplished?: HTTPS security after diginotar". In: *Proceedings of the 2017 Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017*. Ed. by Steve Uhlig and Olaf Maennel. ACM, 2017, pp. 325–340. DOI: 10.1145/3131365.3131401.

[8] Enrique Amigó et al. "A comparison of extrinsic clustering evaluation metrics based on formal constraints". In: *Inf. Retr.* 12.4 (2009), pp. 461–486. DOI: 10.1007/s10791-008-9066-8.

[9] Oleg Andreev. *Proof That Proof-of-Work is the Only Solution to the Byzantine Generals' Problem*. 2014. URL: https://nakamotoinstitute.org/mempool/proof-that-proof-of-work-is-the-only-solution-to-the-byzantine-generals-problem/ (visited on 2017-07-09).

[10] Android. *Permissions overview*. URL: https://developer.android.com/guide/topics/permissions/overview.

[11] Elli Androulaki et al. "Evaluating User Privacy in Bitcoin". In: *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Ed. by Ahmad-Reza Sadeghi. Vol. 7859. Lecture Notes in Computer Science. Springer, 2013, pp. 34–51. DOI: 10.1007/978-3-642-39884-1_4.

[12] Andreas Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, 2014. URL: https://bitcoinbook.info/.

[13] Andreas Antonopoulos, Olaoluwa Osuntokun, and René Pickhardt. *Mastering the Lightning Network*. O'Reilly Media, 2020. URL: https://lnbook.info/.

[14] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 375–392. DOI: 10.1109/SP.2017.29.

[15] Steven Arzt et al. “FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps”. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*. Ed. by Michael F. P. O'Boyle and Keshav Pingali. ACM, 2014, pp. 259–269. DOI: 10.1145/2594291.2594299.

[16] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. “A Survey of Attacks on Ethereum Smart Contracts (SoK)”. In: *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Ed. by Matteo Maffei and Mark Ryan. Vol. 10204. Lecture Notes in Computer Science. Springer, 2017, pp. 164–186. DOI: 10.1007/978-3-662-54455-6_8.

[17] *Augur*. 2017. URL: https://augur.net/ (visited on 2017-09-25).

[18] *Aztec Protocol*. 2020. URL: https://www.aztecprotocol.com/.

[19] Adam Back. *A partial hash collision based postage scheme*. 1997. URL: http://www.hashcash.org/papers/announce.txt.

[20] Vivek Kumar Bagaria, Joachim Neu, and David Tse. “Boomerang: Redundancy Improves Latency and Throughput in Payment Networks”. In: *CoRR* abs/1910.01834 (2019). URL: https://arxiv.org/abs/1910.01834.

[21] Marshall Ball et al. “Proofs of Useful Work”. In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 203. URL: https://eprint.iacr.org/2017/203.

[22] Shehar Bano et al. “SoK: Consensus in the Age of Blockchains”. In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*. ACM, 2019, pp. 183–198. DOI: 10.1145/3318041.3355458.

[23] Massimo Bartoletti and Livio Pompianu. “An Analysis of Bitcoin OP_RETURN Metadata”. In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017, pp. 218–230. DOI: 10.1007/978-3-319-70278-0_14.

[24] Massimo Bartoletti and Livio Pompianu. “An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns”. In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017, pp. 494–509. DOI: 10.1007/978-3-319-70278-0_31.

[25] *BEAM*. 2020. URL: https://beam.mw/.

[26] Adam L. Beberg et al. “Folding@home: Lessons from eight years of volunteer distributed computing”. In: *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*. IEEE, 2009, pp. 1–8. DOI: 10.1109/IPDPS.2009.5160922.

[27] Eli Ben-Sasson et al. “Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture”. In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, 2014, pp. 781–796. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.

[28] Eli Ben-Sasson et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[29] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. “Cryptocurrencies Without Proof of Work”. In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Ed. by Jeremy Clark et al. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 142–157. DOI: 10.1007/978-3-662-53357-4_10.

[30] Iddo Bentov, Rafael Pass, and Elaine Shi. “Snow White: Provably Secure Proofs of Stake”. In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 919. URL: https://eprint.iacr.org/2016/919.

[31] M. Berberich and M. Steiner. “Practitioner’s Corner. Blockchain Technology and the GDPR - How to Reconcile Privacy and Distributed Ledgers?” In: 2 (2016), pp. 422–426. ISSN: 2364-2831. DOI: 10.21552/edpl/2016/3/21.

[32] Ferenc Béres, István András Seres, and András A. Benczúr. “A Cryptoeconomic Traffic Analysis of Bitcoins Lightning Network”. In: *CoRR* abs/1911.09432 (2019). arXiv: 1911.09432. URL: https://arxiv.org/abs/1911.09432.

[33] Karthikeyan Bhargavan et al. “Formal Verification of Smart Contracts: Short Paper”. In: *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2016, Vienna, Austria, October 24, 2016*. Ed. by Toby C. Murray and Deian Stefan. ACM, 2016, pp. 91–96. DOI: 10.1145/2993600.2993611.

[34] Alex Biryukov and Daniel Feher. “Portrait of a Miner in a Landscape”. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 2019, pp. 638–643. DOI: 10.1109/INFCOMW.2019.8845201.

[35] Alex Biryukov and Daniel Feher. “Privacy and Linkability of Mining in Zcash”. In: *7th IEEE Conference on Communications and Network Security, CNS 2019, Washington, DC, USA, June 10-12, 2019*. IEEE, 2019, pp. 118–123. DOI: 10.1109/CNS.2019.8802711.

[36] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. “Privacy Aspects and Subliminal Channels in Zcash”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, pp. 1795–1811. DOI: 10.1145/3319535.3345663.

[37] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. “Deanonymisation of Clients in Bitcoin P2P Network”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM, 2014, pp. 15–29. DOI: 10.1145/2660267.2660379.

[38] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. “Findel: Secure Derivative Contracts for Ethereum”. In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017,

pp. 453–467. DOI: 10.1007/978-3-319-70278-0_28. URL: https://hdl.handle.net/10993/30975.

[39] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. “Privacy-preserving KYC on Ethereum”. In: *Proceedings of 1st ERCIM Blockchain Workshop 2018*. European Society for Socially Embedded Technologies (EUSSET), 2018. DOI: 10.18420/blockchain2018_09. URL: https://hdl.handle.net/10993/35915.

[40] Alex Biryukov and Ivan Pustogarov. “Bitcoin over Tor isn’t a Good Idea”. In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 122–134. DOI: 10.1109/SP.2015.15.

[41] Alex Biryukov and Sergei Tikhomirov. “Deanonymization and Linkability of Cryptocurrency Transactions Based on Network Analysis”. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 172–184. DOI: 10.1109/EuroSP.2019.00022. URL: https://hdl.handle.net/10993/39724.

[42] Alex Biryukov and Sergei Tikhomirov. “Security and privacy of mobile wallet users in Bitcoin, Dash, Monero, and Zcash”. In: *Pervasive Mob. Comput.* 59 (2019). DOI: 10.1016/j.pmcj.2019.101030. URL: https://hdl.handle.net/10993/39729.

[43] Alex Biryukov and Sergei Tikhomirov. “Transaction Clustering Using Network Traffic Analysis for Bitcoin and Derived Blockchains”. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 2019, pp. 204–209. DOI: 10.1109/INFCOMW.2019.8845213. URL: https://hdl.handle.net/10993/39728.

[44] *Bitcoin Wallet*. URL: https://bitcoin.org/en/wallets/mobile/android/bitcoinwallet/.

[45] *Bitcoin wallet privacy policy*. 2018. URL: https://github.com/bitcoin-wallet/bitcoin-wallet/wiki/PrivacyPolicy (visited on 2018-08-26).

[46] BitcoinCore. *Max standard tx weight*. 2017. URL: https://github.com/bitcoin/bitcoin/blob/c536dfbcb00fb15963bf5d507b7017c241718bf6/src/policy/policy.h.

[47] *BitcoinJ documentaion. Working with micropayment channels*. URL: https://bitcoinj.github.io/working-with-micropayments.

[48] BitcoinWiki. *Bitcoin protocol documentation*. URL: https://en.bitcoin.it/wiki/Protocol_documentation.

[49] *Bither wallet privacy policy*. 2018. URL: https://github.com/bither/bither-android/wiki/PrivacyPolicy (visited on 2018-08-26).

[50] BitMEX. *Lightning Network (Part 3) – Where Is The Justice?* URL: https://blog.bitmex.com/lightning-network-justice/.

[51] Bitnation. *Estonia e-residency program & Bitnation DAO public notary partnership*. 2015. URL: ttps://bitnation.co/blog/pressrelease-estonia-bitnation-public-notary-partnership/.

[52] Bitnodes. *Global Bitcoin nodes distribution*. URL: https://bitnodes.earn.com/.

[53] The Block. *Person behind 40% of LN's capacity: "I have no doubt in Bitcoin and the Lightning Network"*. 2019. URL: https://bit.ly/39dDpbF.

[54] Karl Bode. *The Rise of Netflix Competitors Has Pushed Consumers Back Toward Piracy*. Vice. Oct. 2018. URL: https://www.vice.com/en_us/article/d3q45v/bittorrent-usage-increases-netflix-streaming-sites.

[55] Ivan Bogatyy. *Linking 96% of Grin transactions*. 2019. URL: https://github.com/bogatyy/grin-linkability.

[56] BOLT. *BOLT 2: Peer Protocol for Channel Management*. 2019. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md.

[57] BOLT. *BOLT 3: : Bitcoin Transaction and Script Formats. Trimmed outputs*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc/blob/dcbf8583976df087c79c3ce0b535311212e6812d/03-transactions.md.

[58] BOLT. *BOLT 4: Onion Routing Protocol*. 2019. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md.

[59] BOLT. *Lightning Network Specifications*. 2019. URL: https://github.com/lightningnetwork/lightning-rfc.

[60] Joseph Bonneau et al. "Coda: Decentralized Cryptocurrency at Scale". In: *IACR Cryptology ePrint Archive* 2020 (2020), p. 352. URL: https://eprint.iacr.org/2020/352.

[61] Joseph Bonneau et al. "Mixcoin: Anonymity for Bitcoin with Accountable Mixes". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 486–504. DOI: 10.1007/978-3-662-45472-5_31.

[62] Joseph Bonneau et al. "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 104–121. DOI: 10.1109/SP.2015.14.

[63] *BRD privacy policy*. 2018. URL: https://brd.com/privacy (visited on 2018-08-26).

[64] Jerry Brito. *The Case for Electronic Cash*. 2019. URL: https://coincenter.org/entry/the-case-for-electronic-cash.

[65] Jonah Brown-Cohen et al. "Formal Barriers to Longest-Chain Proof-of-Stake Protocols". In: *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*. Ed. by Anna Karlin, Nicole Immorlica, and Ramesh Johari. ACM, 2019, pp. 459–473. DOI: 10.1145/3328526.3329567.

[66] Eric Lombrozo BtcDrak Mark Friedenbach. *BIP-112. CHECKSEQUENCEVERIFY*. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki.

[67] Ethan Buchman. *Understanding the Ethereum trie*. 2014. URL: https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/.

[68] Benedikt Bünz et al. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[69] Conrad Burchert, Christian Decker, and Roger Wattenhofer. “Scalable Funding of Bitcoin Micropayment Channel Networks - Regular Submission”. In: *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*. Ed. by Paul G. Spirakis and Philippas Tsigas. Vol. 10616. Lecture Notes in Computer Science. Springer, 2017, pp. 361–377. DOI: 10.1007/978-3-319-69084-1_26.

[70] Vitalik Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform*. 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[71] Vitalik Buterin. *Design Rationale*. 2017. URL: https://github.com/ethereum/wiki/wiki/Design-Rationale.

[72] Vitalik Buterin. *New experimental programming language*. 2017. URL: https://github.com/ethereum/viper (visited on 2017-07-06).

[73] *c-lightning*. URL: https://github.com/ElementsProject/lightning.

[74] *Cambridge Blockchain*. URL: https://cambridge-blockchain.com/.

[75] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. “An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials”. In: *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*. Ed. by Stanislaw Jarecki and Gene Tsudik. Vol. 5443. Lecture Notes in Computer Science. Springer, 2009, pp. 481–500. DOI: 10.1007/978-3-642-00468-1_27.

[76] James Cameron. *What privacy issues did Monero have and still has?* Sept. 2016. URL: https://monero.stackexchange.com/q/1495/4089.

[77] Nic Carter. *The Last Word on Bitcoin’s Energy Consumption*. 2020. URL: https://www.coindesk.com/the-last-word-on-bitcoins-energy-consumption.

[78] Michael del Castillo. *JP Morgan, Credit Suisse Among 8 in Latest Bank Blockchain Test*. Oct. 2016. URL: https://www.coindesk.com/jp-morgan-credit-suisse-among-8-in-latest-bank-blockchain-test/.

[79] Miguel Castro and Barbara Liskov. “Practical byzantine fault tolerance and proactive recovery”. In: *ACM Trans. Comput. Syst.* 20.4 (2002), pp. 398–461. DOI: 10.1145/571637.571640.

[80] *Chainlink*. 2020. URL: https://chain.link/.

[81] Brad Chase and Ethan MacBrough. “Analysis of the XRP Ledger Consensus Protocol”. In: *CoRR* abs/1802.07242 (2018). URL: http://arxiv.org/abs/1802.07242.

[82] Melissa Chase, Chaya Ganesh, and Payman Mohassel. “Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials”. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. Lecture Notes in Computer Science. Springer, 2016, pp. 499–530. DOI: 10.1007/978-3-662-53015-3_18.

[83] Shahbaz Chaudhary. *Adventures in financial and software engineering*. 2015. URL: https://falconair.github.io/2015/01/30/composingcontracts.html.

[84] David Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, 1982, pp. 199–203. DOI: 10.1007/978-1-4757-0602-4_18.

[85] David Chaum. *Epicenter podcast, episode 304. The Forefather of Cryptocurrencies and the Cypherpunk Movement*. 2019. URL: https://epicenter.tv/episodes/304/.

[86] David Chaum, Amos Fiat, and Moni Naor. "Untraceable Electronic Cash". In: *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*. Ed. by Shafi Goldwasser. Vol. 403. Lecture Notes in Computer Science. Springer, 1988, pp. 319–327. DOI: 10.1007/0-387-34799-2_25.

[87] Jing Chen and Silvio Micali. "Algorand: A secure and efficient distributed ledger". In: *Theor. Comput. Sci.* 777 (2019), pp. 155–183. DOI: 10.1016/j.tcs.2019.02.001.

[88] Ting Chen et al. "Under-optimized smart contracts devour your money". In: *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*. Ed. by Martin Pinzger, Gabriele Bavota, and Andrian Marcus. IEEE Computer Society, 2017, pp. 442–446. DOI: 10.1109/SANER.2017.7884650.

[89] Tarun Chitra. "Competitive equilibria between staking and on-chain lending". In: *CoRR* abs/2001.00919 (2020). URL: https://arxiv.org/abs/2001.00919.

[90] Christopher D. Clack, Vikram A. Bakshi, and Lee Braine. "Smart Contract Templates: foundations, design landscape and research directions". In: *CoRR* abs/1608.00771 (2016). URL: https://arxiv.org/abs/1608.00771.

[91] Jeremy Clark, Didem Demirag, and Seyedehmahsa Moosavi. "Demystifying Stablecoins". In: *ACM Queue* 18.1 (2020), pp. 39–60. DOI: 10.1145/3387945.3388781.

[92] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks". In: *Physical review E* 70.6 (2004), p. 066111. URL: https://arxiv.org/abs/cond-mat/0408187.

[93] *clightning Plugins*. 2020. URL: https://github.com/ElementsProject/lightning/blob/master/doc/PLUGINS.md (visited on 2020-03-05).

[94] Coindesk. *ICO Tracker*. 2017. URL: https://www.coindesk.com/ico-tracker/ (visited on 2017-09-18).

[95] Justin Connell. *How Much Does it Cost to Run a Full Bitcoin Node?* 2017. URL: https://news.bitcoin.com/cost-full-bitcoin-node/.

[96] Marco Conoscenti, Antonio Vetrò, and Juan Carlos De Martin. "Hubs, Rebalancing and Service Providers in the Lightning Network". In: *IEEE Access* 7 (2019), pp. 132828–132840. DOI: 10.1109/ACCESS.2019.2941448.

[97] Consensys. *Ethereum Contract Security Techniques and Tips*. 2016. URL: https://github.com/ConsenSys/smart-contract-best-practices.

[98] Bitcoin Core. *Bitcoin Core version 0.10.0 released*. Feb. 2015. URL: https://bitcoin.org/en/release/v0.10.0.

[99] Bitcoin Core. *Compact Blocks FAQ*. June 2016. URL: https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/.

[100] *Counterparty Protocol Specification*. 2019. URL: https://counterparty.io/docs/protocol_specification/.

[101] John C. Cox, Stephen A. Ross, and Mark Rubinstein. "Option pricing: A simplified approach". In: *Journal of Financial Economics* 7 (3 1979), pp. 229–263. ISSN: 0304-405X. DOI: 10.1016/0304-405x(79)90015-1.

[102] Kyle Croman et al. "On Scaling Decentralized Blockchains - (A Position Paper)". In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Ed. by Jeremy Clark et al. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 106–125. DOI: 10.1007/978-3-662-53357-4_8.

[103] Suhas Daftuar. *Bitcoin Core. Commit 5add7a74. Track transaction packages in CTxMemPoolEntry*. 2015. URL: https://github.com/bitcoin/bitcoin/commit/5add7a74.

[104] Suhas Daftuar. *p2p: Add 2 outbound block-relay-only connections*. 2019. URL: https://github.com/bitcoin/bitcoin/pull/15759.

[105] Wei Dai. *B-money*. 1998. URL: http://www.weidai.com/bmoney.txt.

[106] Philip Daian et al. "Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges". In: *CoRR* abs/1904.05234 (2019). URL: https://arxiv.org/abs/1904.05234.

[107] Gijs van Dam et al. "Improvements of the Balance Discovery Attack on Lightning Network Payment Channels". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 1385. URL: https://eprint.iacr.org/2019/1385.

[108] Coin Dance. *Bitcoin Nodes Summary*. 2019. URL: https://coin.dance/nodes.

[109] Pranav Dandekar et al. "Liquidity in credit networks: a little trust goes a long way". In: *Proceedings 12th ACM Conference on Electronic Commerce (EC-2011), San Jose, CA, USA, June 5-9, 2011*. Ed. by Yoav Shoham, Yan Chen, and Tim Roughgarden. ACM, 2011, pp. 147–156. DOI: 10.1145/1993574.1993597.

[110] George Danezis and Sarah Meiklejohn. "Centrally Banked Cryptocurrencies". In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL: https://eprint.iacr.org/2015/502.

[111] Dash. *Dash website*. URL: https://www.dash.org/.

[112] *Dash wallet privacy policy*. 2018. URL: https://github.com/HashEngineering/dash-wallet/wiki/PrivacyPolicy (visited on 2018-08-26).

[113] dEBRUYNE and ErCiccione. *Monero. Network upgrade and release 0.15*. 2019. URL: https://web.getmonero.org/2019/10/01/announcement-release-0-15.html.

[114] Christian Decker. *BIP-118. SIGHASH_NOINPUT*. 2017. URL: https://github.com/bitcoin/bips/blob/master/bip-0118.mediawiki.

[115] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. *eltoo: A simple layer2 protocol for bitcoin*. URL: https://blockstream.com/eltoo.pdf.

[116] Christian Decker and Roger Wattenhofer. “A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels”. In: *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*. Ed. by Andrzej Pelc and Alexander A. Schwarzmann. Vol. 9212. Lecture Notes in Computer Science. Springer, 2015, pp. 3–18. DOI: 10.1007/978-3-319-21741-3_1.

[117] Kevin Delmolino et al. “Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab”. In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Ed. by Jeremy Clark et al. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 79–94. DOI: 10.1007/978-3-662-53357-4_6.

[118] Tuur Demeester. *The Bitcoin Reformation*. 2019. URL: https://docsend.com/view/ijd8qrs.

[119] Inderjit S. Dhillon. “Co-clustering documents and words using bipartite spectral graph partitioning”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*. Ed. by Doheon Lee et al. ACM, 2001, pp. 269–274. URL: https://portal.acm.org/citation.cfm?id=502512.502550.

[120] Claudia Díaz et al. “Towards Measuring Anonymity”. In: *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*. Ed. by Roger Dingledine and Paul F. Syverson. Vol. 2482. Lecture Notes in Computer Science. Springer, 2002, pp. 54–68. DOI: 10.1007/3-540-36467-6_5.

[121] Whitfield Diffie and Martin E. Hellman. “New directions in cryptography”. In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[122] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. “Tor: The Second-Generation Onion Router”. In: *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*. Ed. by Matt Blaze. USENIX, 2004, pp. 303–320. URL: https://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html.

[123] Ethereum documentation. *Mining*. 2017. URL: https://ethdocs.org/en/latest/mining.html.

[124] Kavindu Dodanduwa and Ishara Kaluthanthri. “Role of Trust in OAuth 2.0 and OpenID Connect”. In: *CoRR* abs/1808.10624 (2018). URL: https://arxiv.org/abs/1808.10624.

[125] Maya Dotan et al. “Survey on Cryptocurrency Networking: Context, State-of-the-Art, Challenges”. In: *CoRR* abs/2008.08412 (2020). URL: https://arxiv.org/abs/2008.08412.

[126] dpzz. *What’s the difference between “balance” and “unlocked balance”?* Jan. 2017. URL: https://monero.stackexchange.com/q/3262/4089.

[127] Thaddeus Dryja. *Discreet Log Contracts*. URL: https://adiabat.github.io/dlc.pdf.

[128] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147. DOI: 10.1007/3-540-48071-4_10.

[129] Stefan Dziembowski et al. "PERUN: Virtual Payment Channels over Cryptographic Currencies". In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 635. URL: https://eprint.iacr.org/2017/635.

[130] e-Estonia. *New Possibilities for e-residents*. 2015. URL: https://e-estonia.com/new-possibilities-for-e-residents/.

[131] *Echidna: A Fast Smart Contract Fuzzer*. URL: https://github.com/crytic/echidna.

[132] *Eclair*. URL: https://github.com/ACINQ/eclair.

[133] Benjamin Egelund-Müller et al. "Automated Execution of Financial Contracts on Blockchains". In: *Bus. Inf. Syst. Eng.* 59.6 (2017), pp. 457–467. DOI: 10.1007/s12599-017-0507-z.

[134] *EIP 1884: Repricing for trie-size-dependent opcodes*. 2019. URL: https://eips.ethereum.org/EIPS/eip-1884.

[135] *Electrum*. URL: https://electrum.org/.

[136] Electrum. *The next release of Electrum will support Lightning payments*. 2019. URL: https://twitter.com/ElectrumWallet/status/1183706431473815552.

[137] *Electrum privacy policy*. 2018. URL: https://electrum.org/\#privacy (visited on 2018-08-26).

[138] Meghan Elison. *Christopher Kong: PSD2 Means Opportunity*. 2016. URL: https://ripple.com/insights/christopher-kong-psd2/.

[139] *Elliptic*. 2020. URL: https://www.elliptic.co/.

[140] Daniel Ellison. *An Introduction to LLL for Ethereum Smart Contract Development*. 2017. URL: https://media.consensys.net/an-introduction-to-lll-for-ethereum-smart-contract-development-e26e38ea6c23.

[141] EmelyanenkoK. *Payment channel congestion via spam-attack*. 2017. URL: https://github.com/lightningnetwork/lightning-rfc/issues/182.

[142] Felix Engelmann et al. "Towards an economic analysis of routing in payment channel networks". In: *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, SERIAL@Middleware 2017, Las Vegas, NV, USA, December 11-15, 2017*. ACM, 2017, 2:1–2:6. DOI: 10.1145/3152824.3152826.

[143] *ENS*. 2017. URL: https://ens.domains/ (visited on 2017-09-25).

[144] *Enterprise Ethereum Alliance*. 2017. URL: https://entethalliance.org/.

[145] ErCiccione. *Another privacy-enhancing technology added to Monero: Dandelion++*. 2020. URL: https://web.getmonero.org/2020/04/18/dandelion-implemented.html.

[146] *Ethash*. 2017. URL: https://github.com/ethereum/wiki/wiki/Ethash (visited on 2017-07-05).

[147] *Ethereum Classic*. URL: https://ethereumclassic.org/.

[148] *Ethereum Contract Security Techniques and Tips*. 2016. URL: https://github.com/ConsenSys/smart-contract-best-practices (visited on 2017-07-06).

[149] *Ethereus Gas Station*. 2020. URL: https://ethgasstation.info/.

[150] Etherscan. *Contracts With Verified Source Codes Only*. URL: https://etherscan.io/contractsVerified.

[151] Ethhub. *Monetary Policy*. 2020. URL: https://docs.ethhub.io/ethereum-basics/monetary-policy/.

[152] Ethstats. *Ethstats*. URL: https://ethstats.net/.

[153] EUGDPR. *EU General Data Protection Regulation*. 2016. URL: https://www.eugdpr.org/.

[154] expez. *In what ways can a wallet connected to a malicious remote node be abused?* Sept. 2016. URL: https://monero.stackexchange.com/q/2962/4089.

[155] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: bitcoin mining is vulnerable". In: *Commun. ACM* 61.7 (2018), pp. 95–102. DOI: 10.1145/3212998.

[156] *F-Droid*. URL: https://f-droid.org/en/.

[157] FALCON. *The Falcon Project*. URL: https://www.falcon-net.org/.

[158] Giulia C. Fanti and Pramod Viswanath. "Anonymity Properties of the Bitcoin P2P Network". In: *CoRR* abs/1703.08761 (2017). URL: https://arxiv.org/abs/1703.08761.

[159] Giulia C. Fanti et al. "Compounding of Wealth in Proof-of-Stake Cryptocurrencies". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 42–61. DOI: 10.1007/978-3-030-32101-7_3.

[160] Giulia C. Fanti et al. "Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees". In: *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2018, Irvine, CA, USA, June 18-22, 2018*. Ed. by Konstantinos Psounis, Aditya Akella, and Adam Wierman. ACM, 2018, pp. 5–7. DOI: 10.1145/3219617.3219620.

[161] Daniel Feher. *Empirical analysis of the Zcash blockchain. A grant proposal*. 2017. URL: https://github.com/ZcashFoundation/GrantProposals-2017Q4/issues/24.

[162] Patrick Feisthammel. *Explanation of the web of trust of PGP*. 2017. URL: https://www.rubin.ch/pgp/weboftrust.en.html.

[163] fiatjaf. *lnchannels. history of the open network*. 2020. URL: https://ln.bigsun.xyz/.

[164] FIBRE. *The Fast Internet Bitcoin Relay Engine*. URL: https://bitcoinfibre.org/.

[165] Karl Floersch. *Ethereum Smart Contracts in L2: Optimistic Rollup*. 2019. URL: https://medium.com/plasma-group/ethereum-smart-contracts-in-l2-optimistic-rollup-2c1cef2ec537.

[166] FlowDroid. *FlowDroid Static Data Flow Tracker (source code)*. URL: https://github.com/secure-software-engineering/FlowDroid/.

[167] David Floyd. *Bitmain's Latest Crypto ASIC Can Mine Zcash*. 2018. URL: https://www.coindesk.com/bitmains-latest-crypto-asic-can-mine-zcash.

[168] Michael Folkson. *How is a "standard" Bitcoin transaction defined?* 2017. URL: https://bitcoin.stackexchange.com/q/52528/31712.

[169] Simon Frankau et al. "Commercial uses: Going functional on exotic trades". In: *Journal of Functional Programming* 19.1 (2009), pp. 27–45. URL: http://arbitrary.name/papers/fpf.pdf.

[170] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. "A Decentralized Public Key Infrastructure with Identity Retention". In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 803. URL: https://eprint.iacr.org/2014/803.

[171] *Functional Programming for Ethereum*. 2017. URL: https://github.com/evm-lang-design/evm-lang-design.

[172] Dipl-Inf Jean-Marie Gaillourdet. "A software language approach to derivative contracts in finance". In: *CEUR Workshop Proceedings*. Vol. 750. 2011, pp. 39–43. URL: https://softech.cs.uni-kl.de/homepage/publications/Gaillourdet11software.pdf.

[173] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10.

[174] Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Stake-Bleeding Attacks on Proof-of-Stake Blockchains". In: *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018*. IEEE, 2018, pp. 85–92. DOI: 10.1109/CVCBT.2018.00015.

[175] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. "Service-Oriented Sharding with Aspen". In: *CoRR* abs/1611.06816 (2016). URL: https://arxiv.org/abs/1611.06816.

[176] *Genesis*. URL: https://genesis.vision/.

[177] *Genesis Github repository*. URL: https://github.com/GenesisVision/ico-contracts/.

[178] Ryan Gentry and Matt Shapiro. *Privacy Is a Feature, Not a Product*. 2019. URL: https://multicoin.capital/2019/09/24/privacy-is-a-feature/.

[179] Arthur Gervais et al. "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*. Ed. by Charles N. Payne Jr. et al. ACM, 2014, pp. 326–335. DOI: 10.1145/2664243.2664267.

[180] Jeremy Gibbons. *Functional Programming for Domain-Specific Languages*. 2015. DOI: 10.1007/978-3-319-15940-9_1.

[181] Alex Gluchowski. *Optimistic vs. ZK Rollup: Deep Dive*. 2019. URL: https://medium.com/matter-labs/optimistic-vs-zk-rollup-deep-dive-ea141e71e075.

[182] *Gnosis*. 2017. URL: https://gnosis.pm/ (visited on 2017-09-25).

[183] *Golem*. 2017. URL: https://golem.network/ (visited on 2017-09-25).

[184] Matthew Green and Ian Miers. "Bolt: Anonymous Payment Channels for Decentralized Currencies". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani M. Thuraisingham et al. ACM, 2017, pp. 473–489. DOI: 10.1145/3133956.3134093.

[185] Gideon Greenspan. *Why Many Smart Contract Use Cases Are Simply Impossible*. 2016. URL: https://www.coindesk.com/three-smart-contract-misconceptions.

[186] *Grin*. 2020. URL: https://grin.mw/.

[187] Cyril Grunspan and Ricardo Pérez-Marco. "Ant routing algorithm for the Lightning Network". In: *CoRR* abs/1807.00151 (2018). URL: https://arxiv.org/abs/1807.00151.

[188] Lewis Gudgeon et al. "SoK: Off The Chain Transactions". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 360. URL: https://eprint.iacr.org/2019/360.

[189] David A. Harding. *Are Micropayment channels still subject to malleability after BIP65?* 2016. URL: https://bitcoin.stackexchange.com/a/48546/31712.

[190] Mike Hearn. *Anti DoS for tx replacement*. 2013. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html.

[191] Viola Hellström. *PSD2 – the directive that will change banking as we know it*. 2017. URL: https://www.evry.com/en/news/articles/psd2-the-directive-that-will-change-banking-as-we-know-it/.

[192] Kevlin Henney. "Inside requirements". In: *Application development advisor* (May 2003). URL: https://www.slideshare.net/Kevlin/inside-requirements.

[193] Sebastian A. Henningsen et al. "Eclipsing Ethereum Peers with False Friends". In: *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 300–309. DOI: 10.1109/EuroSPW.2019.00040.

[194] Jordi Herrera-Joancomartí et al. "On the Difficulty of Hiding the Balance of Lightning Network Channels". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*. Ed. by Steven D. Galbraith et al. ACM, 2019, pp. 602–612. DOI: 10.1145/3321705.3329812.

[195] Alyssa Hertig. *Bitcoin's Future: Exactly How a Coming Upgrade Could Improve Privacy and Scaling*. 2020. URL: https://www.coindesk.com/bitcoins-future-exactly-how-a-coming-upgrade-could-improve-privacy-and-scaling.

[196] Everett Hildenbrandt et al. "KEVM: A Complete Formal Semantics of the Ethereum Virtual Machine". In: *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 204–217. DOI: 10.1109/CSF.2018.00022.

[197] Yoichi Hirai. *Bamboo: a morphing smart contract language*. 2017. URL: https://github.com/pirapira/bamboo (visited on 2017-07-06).

[198] Yoichi Hirai. *Formal Verification of Ethereum Contracts*. 2017. URL: https://github.com/pirapira/ethereum-formal-verification-overview.

[199] *Hive*. URL: https://bitcointalk.org/index.php?topic=1959159.0.

[200] *Hive Github repository*. URL: https://github.com/HiveProjectLtd/HVNTokenBasic/.

[201] Diara Hopwood et al. *Zcash protocol specification*. Mar. 2020. URL: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[202] Tom Hvitved. "A survey of formal languages for contracts". In: *Formal Languages and Analysis of Contract-Oriented Software* (2010), pp. 29–32. URL: https://curis.ku.dk/ws/files/172435414/Hvitved_2010_A_survey_of_formal.pdf.

[203] Hyperledger. *Hyperledger Business Blockchain Technologies*. URL: https://www.hyperledger.org/.

[204] Eric Jackson and Christopher Grey. *Cryptocurrency is accomplishing PayPal's original mission*. 2017. URL: https://venturebeat.com/2017/12/02/cryptocurrency-is-accomplishing-paypals-original-mission/.

[205] Joost Jager. *Circuit Breaker*. 2020. URL: https://github.com/lightningequipment/circuitbreaker.

[206] Tom Elvis Jedusor. *MimbleWimble*. 2016. URL: https://scalingbitcoin.org/papers/mimblewimble.txt.

[207] Bo Jiang, Ye Liu, and W. K. Chan. "ContractFuzzer: fuzzing smart contracts for vulnerability detection". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. Ed. by Marianne Huchard, Christian Kästner, and Gordon Fraser. ACM, 2018, pp. 259–269. DOI: 10.1145/3238147.3238177.

[208] jnnk. *What is Gas Limit in Ethereum?* 2015. URL: https://bitcoin.stackexchange.com/a/39197.

[209] Simon L. Peyton Jones and Jean-Marc Eber. *How to write a financial contract*. 2003.

[210] jsoup. *Java HTML Parser*. URL: https://jsoup.org/.

[211] Harry A. Kalodner et al. "Arbitrum: Scalable, private smart contracts". In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 1353–1370. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner.

[212] Harry A. Kalodner et al. "BlockSci: Design and applications of a blockchain analysis platform". In: *CoRR* abs/1709.02489 (2017). URL: https://arxiv.org/abs/1709.02489.

[213] George Kappos et al. "An Empirical Analysis of Anonymity in Zcash". In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 463–477. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/kappos.

[214] George Kappos et al. "An Empirical Analysis of Privacy in the Lightning Network". In: *CoRR* abs/2003.12470 (2020). URL: https://arxiv.org/abs/2003.12470.

[215] Dmitry Khovratovich. *debt.sol*. 2016. URL: https://gist.github.com/khovratovich/45f68082b556b45eb64e8e1c3eb82892.

[216] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. "A Composable Security Treatment of the Lightning Network". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 778. URL: https://eprint.iacr.org/2019/778.

[217] Aggelos Kiayias et al. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 357–388. DOI: 10.1007/978-3-319-63688-7_12.

[218] Christine Kim. *Inside Monero's 'Last Ditch Effort' to Block Crypto Mining ASICs*. 2019. URL: https://www.coindesk.com/inside-moneros-last-ditch-effort-to-block-crypto-mining-asics.

[219] Christine Kim. *The Rise of ASICs: A Step-by-Step History of Bitcoin Mining*. 2020. URL: https://www.coindesk.com/rise-of-asics-bitcoin-mining-history.

[220] Ariah Klages-Mundt et al. "Stablecoins 2.0: Economic Foundations and Risk-based Models". In: *CoRR* abs/2006.12388 (2020). URL: https://arxiv.org/abs/2006.12388.

[221] Nadav Kohen. *Payment Points Part 1: Replacing HTLCs*. 2019. URL: https://suredbits.com/payment-points-part-1/.

[222] Philip Koshy, Diana Koshy, and Patrick D. McDaniel. "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 469–485. DOI: 10.1007/978-3-662-45472-5_30.

[223] Kovri. *The Kovri Project*. URL: https://gitlab.com/kovri-project/kovri.

[224] Joshua Kroll, Ian Davey, and Edward Felten. "The Economics of Bitcoin Mining, or Bitcoin in the Presence of Adversaries". In: *Proceedings of WEIS*. 2013. URL: https://www.econinfosec.org/archive/weis2013/papers/KrollDaveyFeltenWEIS2013.pdf.

[225] Yujin Kwon et al. "Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash?" In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 935–951. DOI: 10.1109/SP.2019.00075.

[226] *KYC-Chain*. URL: https://kyc-chain.com/.

[227] scikit learn. *Biclustering*. 2018. URL: https://scikit-learn.org/stable/modules/biclustering.html (visited on 2018-07-19).

[228] Ari Levy and Lorie Konish. *The five biggest tech companies now make up 17.5% of the S&P 500*. 2020. URL: https://www.cnbc.com/2020/01/28/sp-500-dominated-by-apple-microsoft-alphabet-amazon-facebook.html.

[229] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. "Inclusive Block Chain Protocols". In: *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 528–547. DOI: 10.1007/978-3-662-47854-7_33.

[230] Karen Lewison and Francisco Corella. *Backing rich credentials with a blockchain PKI*. 2016. URL: https://pomcor.com/techreports/BlockchainPKI.pdf.

[231] LexiFi. *Frequently asked questions*. URL: https://www.lexifi.com/faq/.

[232] *lit*. URL: https://github.com/mit-dci/lit.

[233] Bingchang Liu et al. “Software vulnerability discovery techniques: A survey”. In: *2012 fourth international conference on multimedia information networking and security*. IEEE. 2012, pp. 152–156.

[234] “W3C XML Path Language”. In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Springer US, 2009, p. 3441. DOI: 10.1007/978-0-387-39940-9_3982.

[235] *LND*. URL: https://github.com/lightningnetwork/lnd.

[236] *LND documentation. INSTALL*. 2020. URL: https://github.com/lightningnetwork/lnd/blob/97da7b344498d3f5ec8b4760018d2d2eaf3f634b/docs/INSTALL.md.

[237] lpd. *Lightning Peach Node in rust*. URL: https://github.com/LightningPeach/lpd.

[238] Loi Luu et al. “A Secure Sharding Protocol For Open Blockchains”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 17–30. DOI: 10.1145/2976749.2978389.

[239] Loi Luu et al. “Making Smart Contracts Smarter”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 254–269. DOI: 10.1145/2976749.2978309.

[240] Giulio Malavolta et al. “Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability”. In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/.

[241] Giulio Malavolta et al. “Concurrency and Privacy with Payment-Channel Networks”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani M. Thuraisingham et al. ACM, 2017, pp. 455–471. DOI: 10.1145/3133956.3134096.

[242] Giulio Malavolta et al. “SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks”. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/silentwhispers-enforcing-security-and-privacy-decentralized-credit-networks/.

[243] manontheinside. *Does Monero protect against timing analysis?* Nov. 2016. URL: https://monero.stackexchange.com/q/2765/4089.

[244] *Manticore*. URL: https://github.com/trailofbits/manticore.

[245] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. “Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network”. In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 236. URL: https://eprint.iacr.org/2018/236.

[246] Stefano Martinazzi. “The evolution of Lightning Network’s Topology during its first year and the influence over its core values”. In: *CoRR* abs/1902.07307 (2019). URL: https://arxiv.org/abs/1902.07307.

[247] Gregory Maxwell. *Bitcoin Core. Commit f692fce8. Make RelayWalletTransaction attempt to AcceptToMemoryPool*. 2016. URL: https://github.com/bitcoin/bitcoin/pull/9290/commits/f692fce8.

[248] Gregory Maxwell. *CoinJoin: Bitcoin privacy for the real world*. 2013. URL: https://bitcointalk.org/index.php?topic=279249.

[249] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Ed. by Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer, 2002, pp. 53–65. DOI: 10.1007/3-540-45748-8_5.

[250] David Mazières. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus*. 2014. URL: https://www.stellar.org/papers/stellar-consensus-protocol.pdf (visited on 2017-07-06).

[251] Darryl McAdams. *An Ontology for Smart Contracts*. 2017. URL: https://iohk.io/en/research/library/papers/an-ontology-for-smart-contracts/.

[252] Patrick McCorry et al. "Pisa: Arbitration Outsourcing for State Channels". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*. ACM, 2019, pp. 16–30. DOI: 10.1145/3318041.3355461.

[253] Patrick McCorry et al. "Towards Bitcoin Payment Networks". In: *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I*. Ed. by Joseph K. Liu and Ron Steinfeld. Vol. 9722. Lecture Notes in Computer Science. Springer, 2016, pp. 57–76. DOI: 10.1007/978-3-319-40253-6_4.

[254] Sarah Meiklejohn et al. "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names". In: *login Usenix Mag.* 38.6 (2013). URL: https://www.usenix.org/publications/login/december-2013-volume-38-number-6/fistful-bitcoins-characterizing-payments-among.

[255] Elena Mesropyan. *21 Companies Leveraging Blockchain for Identity Management and Authentication*. 2017. URL: https://letstalkpayments.com/22-companies-leveraging-blockchain-for-identity-management-and-authentication/.

[256] Ian Miers et al. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 397–411. DOI: 10.1109/SP.2013.34.

[257] Andrew Miller et al. *Discovering Bitcoin's public topology and influential nodes*. 2015. URL: https://www.cs.umd.edu/projects/coinscope/coinscope.pdf.

[258] Andrew Miller et al. "Sprites and State Channels: Payment Networks that Go Faster Than Lightning". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 508–526. DOI: 10.1007/978-3-030-32101-7_30.

[259] Makiko Mita et al. "What is Stablecoin?: A Survey on Price Stabilization Mechanisms for Decentralized Payment Systems". In: *8th International Congress on Advanced Applied Informatics, IIAI-AAI 2019, Toyama, Japan, July 7-11, 2019*. IEEE, 2019, pp. 60–66. DOI: 10.1109/IIAI-AAI.2019.00023.

[260] Ayelet Mizrahi and Aviv Zohar. “Congestion Attacks in Payment Channel Networks”. In: *CoRR* abs/2002.06564 (2020). URL: https://arxiv.org/abs/2002.06564.

[261] Monero. *Monero website*. URL: https://web.getmonero.org/.

[262] MoneroHash. *Monero Active Nodes Distribution*. URL: https://monerohash.com/nodes-distribution.html.

[263] *Monerujo privacy policy*. 2018. URL: https://www.monerujo.io/privacy-policy.html (visited on 2018-08-26).

[264] *Monetary authority of Singapore. The future is here. Project Ubin: SGD on Distributed Ledger*. 2017. URL: https://www.mas.gov.sg/Singapore-Financial-Centre/Smart-Financial-Centre/Project-Ubin.aspx.

[265] *Money laundering and terrorist financing: Council agrees its negotiating stance*. 2016. URL: https://www.consilium.europa.eu/en/press/press-releases/2016/12/20-money-laundering-and-terrorist-financing/.

[266] Alex Morcos. *Bitcoin Core. Commit 971a4e6b. Lower default policy limits*. 2015. URL: https://github.com/bitcoin/bitcoin/commit/971a4e6b.

[267] Pedro Moreno-Sanchez et al. “Mind Your Credit: Assessing the Health of the Ripple Credit Network”. In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. Ed. by Pierre-Antoine Champin et al. ACM, 2018, pp. 329–338. DOI: 10.1145/3178876.3186099.

[268] Simon Morris. *If you’re not Breaking Rules you’re Doing it Wrong — Bittorrent Lessons for Crypto (2 of 4)*. 2018. URL: https://medium.com/@simonhmorris/if-youre-not-breaking-rules-you-re-doing-it-wrong-bittorrent-lessons-for-crypto-2-of-4-72c68227fe69.

[269] Malte Möser et al. “An Empirical Analysis of Traceability in the Monero Blockchain”. In: *PoPETs* 2018.3 (2018), pp. 143–163.

[270] José Parra Moyano and Omri Ross. “KYC Optimization Using Distributed Ledger Technology”. In: *Bus. Inf. Syst. Eng.* 59.6 (2017), pp. 411–423. DOI: 10.1007/s12599-017-0504-2.

[271] *Mythril*. URL: https://github.com/ConsenSys/mythril.

[272] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. 2008. URL: \href{https://bitcoin.org/bitcoin.pdf}.

[273] Satoshi Nakamoto. *Governments are good at cutting off the heads of a centrally controlled networks like Napster...* The Cryptography Mailing List. 2008. URL: https://www.metzdowd.com/pipermail/cryptography/2008-November/014823.html.

[274] Arvind Narayanan and Jeremy Clark. “Bitcoin’s academic pedigree”. In: *Commun. ACM* 60.12 (2017), pp. 36–45. DOI: 10.1145/3132259.

[275] Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016. ISBN: 978-0-691-17169-2. URL: https://press.princeton.edu/titles/10908.html.

[276] Gleb Naumenko. *p2p: supplying and using asmap to improve IP bucketing in addrman*. 2019. URL: https://github.com/bitcoin/bitcoin/pull/16702.

[277] Gleb Naumenko et al. "Erlay: Efficient Transaction Relay for Bitcoin". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, pp. 817–831. DOI: 10.1145/3319535.3354237.

[278] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. "Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*. IEEE Computer Society, 2016, pp. 358–367. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070.

[279] Till Neudecker and Hannes Hartenstein. "Could Network Information Facilitate Address Clustering in Bitcoin?" In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017, pp. 155–169. DOI: 10.1007/978-3-319-70278-0_9.

[280] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. "Structure and Anonymity of the Bitcoin Transaction Graph". In: *Future Internet* 5.2 (2013), pp. 237–250. DOI: 10.3390/fi5020237.

[281] Russell O'Connor. "Simplicity: A New Language for Blockchains". In: *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2017, Dallas, TX, USA, October 30, 2017*. ACM, 2017, pp. 107–120. DOI: 10.1145/3139337.3139340.

[282] Mikko Ohtamaa. *Know Your Customer partner integration*. 2016. URL: ttps://github.com/TokenMarketNet/ethereum-tokens/blob/master/KYC.rst.

[283] Rachel Rose O'Leary. *Ethereum ASICs Are Here: What the New Miners Mean and What's Next*. 2018. URL: https://www.coindesk.com/ethereum-asics-means-whats-next.

[284] Rachel Rose O'Leary. *Ethereum Developers Give 'Tentative' Greenlight to ASIC-Blocking Code*. 2019. URL: https://www.coindesk.com/ethereum-developers-give-tentative-greenlight-to-asic-blocking-code.

[285] OpenZeppelin. *Blog posts tagged "security"*. 2017. URL: https://blog.zeppelin.solutions/tagged/security.

[286] OpenZeppelin. *SafeMath*. 2017. URL: https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol.

[287] Olaoluwa Osuntokun. *AMP: Atomic Multi-Path Payments over Lightning*. 2018. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

[288] Santiago Palladino. *The Parity Wallet Hack Reloaded*. 2017. URL: https://blog.openzeppelin.com/parity-wallet-hack-reloaded/.

[289] Terence Parr. *ANTLR*. URL: https://www.antlr.org/.

[290] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[291] Cristina Pérez-Solà et al. "LockDown: Balance Availability Attack against Lightning Network Channels". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 1149. URL: https://eprint.iacr.org/2019/1149.

[292] Jack Pettersson and Robert Edström. "Safer smart contracts through type-driven development". MA thesis. 2016. URL: https://hdl.handle.net/20.500.12380/234939.

[293] Simon L. Peyton Jones, Jean-Marc Eber, and Julian Seward. "Composing contracts: an adventure in financial engineering, functional pearl". In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*. Ed. by Martin Odersky and Philip Wadler. ACM, 2000, pp. 280–292. DOI: 10.1145/351240.351267.

[294] Dmytro Piatkivskyi and Mariusz Nowostawski. "Split Payments in Payment Networks". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*. Ed. by Joaquín García-Alfaro et al. Vol. 11025. Lecture Notes in Computer Science. Springer, 2018, pp. 67–75. DOI: 10.1007/978-3-030-00305-0_5.

[295] René Pickhardt. *Just in Time Routing (JIT-Routing) and a channel rebalancing heuristic as an add on for improved routing success in BOLT 1.0*. 2019. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-March/001891.html.

[296] René Pickhardt. *random short channel id for private channels*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc/issues/675.

[297] René Pickhardt and Mariusz Nowostawski. "Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network". In: *CoRR* abs/1912.09555 (2019). URL: https://arxiv.org/abs/1912.09555.

[298] Andrew Poelstra. *On stake and consensus*. 2015. URL: https://nakamotoinstitute.org/static/docs/on-stake-and-consensus.pdf.

[299] Joseph Poon and Vitalik Buterin. *Plasma: Scalable Autonomous Smart Contracts*. 2017. URL: https://plasma.io/plasma.pdf.

[300] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Tech. rep. 2016. URL: https://lightning.network/lightning-network-paper.pdf.

[301] *Populous*. URL: https://populous.world/.

[302] *Populous Github repository*. URL: https://github.com/bitpopulous/populous-smartcontracts.

[303] Simone Porru et al. "Blockchain-oriented software engineering: challenges and new directions". In: *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*. Ed. by Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard. IEEE Computer Society, 2017, pp. 169–171. DOI: 10.1109/ICSE-C.2017.142.

[304] Johan A. Pouwelse et al. "The Bittorrent P2P File-Sharing System: Measurements and Analysis". In: *Peer-to-Peer Systems IV, 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005, Revised Selected Papers*. Ed. by Miguel Castro and Robbert van Renesse. Vol. 3640. Lecture Notes in Computer Science. Springer, 2005, pp. 205–216. DOI: 10.1007/11558989_19.

[305] Pavel Prihodko et al. *Flare: An approach to routing in lightning network*. 2016. URL: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.

[306] *Protecting the integrity of digital assets*. 2020. URL: https://www.chainalysis.com/.

[307] Provable. *The Provable blockchain oracle for modern DApps*. URL: https://provable.xyz/.

[308] *Ptarmigan*. URL: https://github.com/nayutaco/ptarmigan.

[309] Ivan Pustogarov. *Bitcoin Network Probing Tool*. 2017. URL: https://github.com/ivanpustogarov/bcclient (visited on 2018-01-31).

[310] PWC. *Know your customer: quick reference guide*. 2015. URL: https://www.pwc.lu/en/anti-money-laundering/docs/pwc-aml-know-your-customer-2015.pdf.

[311] Jeffrey Quesnelle. "On the linkability of Zcash transactions". In: *CoRR* abs/1712.01210 (2017). URL: https://arxiv.org/abs/1712.01210.

[312] *R3*. URL: https://www.r3.com/.

[313] Raiden. *Raiden network: high speed asset transfers for Ethereum*. 2017. URL: https://raiden.network/.

[314] RANDAO. *A DAO working as RNG of Ethereum*. 2017. URL: https://github.com/randao/randao.

[315] Michel Rauchs, Apolline Blandin, and Anton Dek. *Cambridge Bitcoin Electricity Consumption Index*. 2020. URL: https://www.cbeci.org/.

[316] Realitio. *Crowd-sourced verification for smart contracts*. URL: https://realit.io/.

[317] *Regulation (EU) 2015/847 of the European Parliament and of the Council of 20 May 2015 on information accompanying transfers of funds and repealing Regulation (EC) No 1781/2006 (Text with EEA relevance)*. 2015. URL: https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32015R0847.

[318] Alf Rehn. "The politics of contraband: The honor economies of the warez scene". In: *The journal of socio-economics* 33.3 (2004), pp. 359–374.

[319] Fergal Reid and Martin Harrigan. "An Analysis of Anonymity in the Bitcoin System". In: *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*. IEEE Computer Society, 2011, pp. 1318–1326. DOI: 10.1109/PASSAT/SocialCom.2011.79.

[320] *Remix: online Solidity compiler*. https://remix.ethereum.org/.

[321] BitMEX Research. *Lightning Network (Part 7) – Proportion Of Public vs Private Channels*. 2020. URL: https://blog.bitmex.com/lightning-network-part-7-proportion-of-public-vs-private-channels/.

[322] Raine Rupert Revere. *What is the difference between transaction cost and execution cost in browser solidity?* 2016. URL: https://ethereum.stackexchange.com/q/5812/5113.

[323] Antoine Riard and Gleb Naumenko. "Time-Dilation Attacks on the Lightning Network". In: *CoRR* abs/2006.01418 (2020). URL: https://arxiv.org/abs/2006.01418.

[324] Dan Robinson. *HTLCs considered harmful*. Transcript by Bryan Bishop. 2019. URL: https://diyhpl.us/wiki/transcripts/stanford-blockchain-conference/2019/htlcs-considered-harmful/.

[325] Dan Robinson and Georgios Konstantopoulos. *Ethereum is a Dark Forest*. 2020. URL: https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff.

[326] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. "Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks". In: *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 347–356. DOI: 10.1109/EuroSPW.2019.00045.

[327] Elias Rohrer and Florian Tschorsch. "Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks". In: *CoRR* abs/2006.12143 (2020). URL: https://arxiv.org/abs/2006.12143.

[328] Dorit Ron and Adi Shamir. "Quantitative Analysis of the Full Bitcoin Transaction Graph". In: *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Ed. by Ahmad-Reza Sadeghi. Vol. 7859. Lecture Notes in Computer Science. Springer, 2013, pp. 6–24. DOI: 10.1007/978-3-642-39884-1_2.

[329] Stefanie Roos et al. "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL: https://arxiv.org/abs/1709.05748.

[330] Meni Rosenfeld. *Overview of colored coins*. 2012. URL: https://bitcoil.co.il/BitcoinX.pdf.

[331] Jeremy Rubin. *CHECKTEMPLATEVERIFY*. 2020. URL: https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki.

[332] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin". In: *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*. Ed. by Miroslaw Kutylowski and Jaideep Vaidya. Vol. 8713. Lecture Notes in Computer Science. Springer, 2014, pp. 345–364. DOI: 10.1007/978-3-319-11212-1_20.

[333] Rubén Cuevas Rumín et al. "Is content publishing in BitTorrent altruistic or profit-driven?" In: *Proceedings of the 2010 ACM Conference on Emerging Networking Experiments and Technology, CoNEXT 2010, Philadelphia, PA, USA, November 30 - December 03, 2010*. Ed. by Jaudelice Cavalcante de Oliveira et al. ACM, 2010, p. 11. DOI: 10.1145/1921168.1921183.

[334] Scott Ruoti et al. "Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client". In: *CoRR* abs/1510.08555 (2015). URL: https://arxiv.org/abs/1510.08555.

[335] *Rust-Lightning*. URL: https://github.com/rust-bitcoin/rust-lightning.

[336] Nicolas Van Saberhagen. *CryptoNote v 2.0*. 2013. URL: https://cryptonote.org/whitepaper.pdf.

[337] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. “Measuring and analyzing the characteristics of Napster and Gnutella hosts”. In: *Multimedia Syst.* 9.2 (2003), pp. 170–184. DOI: 10.1007/s00530-003-0088-1.

[338] Todd Schiller. *Financial DSL Listing*. 2013. URL: https://www.dslfin.org/resources.html.

[339] Holger Schinzel and UdjinM6. *Dash 0.12.0 Release notes*. 2015. URL: https://github.com/dashpay/dash/blob/master/doc/release-notes/dash/release-notes-0.12.0.md.

[340] Bruce Schneier. *Debating Full Disclosure*. 2007. URL: https://www.schneier.com/blog/archives/2007/01/debating_full_d.html.

[341] Claus P. Schnorr. *Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system*. US Patent US4995082A. 1989. URL: https://patents.google.com/patent/US4995082.

[342] Steffen Schuldenzucker. *An Axiomatic Framework for No-Arbitrage Relationships in Financial Derivatives Markets*. 2016. URL: https://www.ifi.uzh.ch/ce/publications/LPT.pdf.

[343] Steffen Schuldenzucker. “Decomposing contracts”. MA thesis. University of Bonn, 2014. URL: https://www.ifi.uzh.ch/ce/people/schuldenzucker/decomposingcontracts.pdf.

[344] Pablo Lamela Seijas, Simon J. Thompson, and Darryl McAdams. “Scripting smart contracts for distributed ledger technology”. In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 1156. URL: https://eprint.iacr.org/2016/1156.

[345] István András Seres et al. “Topological Analysis of Bitcoin’s Lightning Network”. In: *CoRR* abs/1901.04972 (2019). URL: https://arxiv.org/abs/1901.04972.

[346] Ilya Sergey and Aquinas Hobor. “A Concurrent Perspective on Smart Contracts”. In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017, pp. 478–493. DOI: 10.1007/978-3-319-70278-0_30.

[347] *Serpent*. 2017. URL: https://github.com/ethereum/wiki/wiki/Serpent.

[348] *Sharding FAQ*. 2016. URL: https://github.com/ethereum/wiki/wiki/Sharding-FAQ (visited on 2017-07-05).

[349] *Simple Bitcoin*. 2018. URL: https://github.com/btcontract/wallet (visited on 2018-08-20).

[350] Emin Gün Sirer. *Thoughts on The DAO Hack*. 2016. URL: https://hackingdistributed.com/2016/06/17/thoughts-on-the-dao-hack/.

[351] Vibhaalakshmi Sivaraman et al. “Routing Cryptocurrency with the Spider Network”. In: *Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018*. ACM, 2018, pp. 29–35. DOI: 10.1145/3286062.3286067.

[352] *Slither, the Solidity source analyzer*. URL: https://github.com/crytic/slither.

[353] *SmartCheck source code*. URL: https://github.com/smartdec/smartcheck.

[354] SmartDec. *SmartDec Scanner*. 2018. URL: https://smartdecscanner.com/.

[355] *SnapSwap*. URL: https://snapswap.eu/.

[356] *Solidity official documentation*. URL: https://solidity.readthedocs.io/.

[357] *Solidity v0.5.0 Breaking Changes*. 2018. URL: https://solidity.readthedocs.io/en/v0.5.0/050-breaking-changes.html.

[358] Yonatan Sompolinsky and Aviv Zohar. “Accelerating Bitcoin’s Transaction Processing. Fast Money Grows on Trees, Not Chains”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 881. URL: https://eprint.iacr.org/2013/881.

[359] *Sovrin*. URL: https://www.sovrin.org/.

[360] Jeremy Spillman. *Anti DoS for tx replacement*. 2013. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html.

[361] Chris Stewart. *What is the tradeoff between privacy and implementation complexity of Dandelion (BIP156)*. 2018. URL: https://bitcoin.stackexchange.com/q/81503/31712.

[362] *Storj*. 2017. URL: https://storj.io/ (visited on 2017-09-25).

[363] Anton van Straaten. *Composing Contracts*. 2007. URL: https://web.archive.org/web/20130814194431/http://contracts.scheming.org.

[364] Clare Sullivan and Eric Burger. “E-residency and blockchain”. In: *Computer Law and Security Review* 33.4 (2017), pp. 470–481. ISSN: 0267-3649. DOI: 10.1016/j.clsr.2017.03.016.

[365] Alex Sunnarborg. *ICO Investments Pass VC Funding in Blockchain Market First*. 2017. URL: https://www.coindesk.com/ico-investments-pass-vc-funding-in-blockchain-market-first/.

[366] Nick Szabo. *A Formal Language for Analyzing Contracts*. 2002. URL: https://nakamotoinstitute.org/contract-language/.

[367] Nick Szabo. *Bit gold*. 2005. URL: https://unenumerated.blogspot.com/2005/12/bit-gold.html.

[368] Nick Szabo. “Formalizing and Securing Relationships on Public Networks”. In: *First Monday* 2.9 (1997). URL: https://firstmonday.org/ojs/index.php/fm/article/view/548.

[369] Paul Sztorc. *Nothing is Cheaper than Proof of Work*. 2015. URL: https://www.truthcoin.info/blog/pow-cheapest/ (visited on 2017-07-09).

[370] Weizhao Tang et al. “Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks”. In: *CoRR* abs/1909.02717 (2019). URL: https://arxiv.org/abs/1909.02717.

[371] Weizhao Tang et al. “Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks”. In: *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Boston, MA, USA, June, 8-12, 2020*. Ed. by Edmund Yeh, Athina Markopoulou, and Y. C. Tay. ACM, 2020, pp. 81–82. DOI: 10.1145/3393691.3394213.

[372] Jason Teutsch and Christian Reitwiessner. *A scalable verification solution for blockchains*. 2017. URL: https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf.

[373] Sergei Tikhomirov. *Assessing the Security and Anonymity of the Lightning Network - accompanying website*. 2019. URL: https://sites.google.com/view/lightning-privacy.

[374] Sergei Tikhomirov. "Ethereum: State of Knowledge and Research Perspectives". In: *Foundations and Practice of Security - 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers*. Ed. by Abdessamad Imine et al. Vol. 10723. Lecture Notes in Computer Science. Springer, 2017, pp. 206–221. DOI: 10.1007/978-3-319-75650-9_14. URL: https://hdl.handle.net/10993/32468.

[375] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. "A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network". In: *2020 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, September 7-11, 2020*. IEEE, 2020. URL: https://eprint.iacr.org/2020/303.

[376] Sergei Tikhomirov et al. "Probing Channel Balances in the Lightning Network". In: *CoRR* abs/2004.00333 (2020). URL: https://arxiv.org/abs/2004.00333.

[377] Sergei Tikhomirov et al. "SmartCheck: Static Analysis of Ethereum Smart Contracts". In: *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018, Gothenburg, Sweden, May 27 - June 3, 2018*. ACM, 2018, pp. 9–16. URL: https://hdl.handle.net/10993/35862.

[378] TLSNotary. *A new kind of auditing - cryptographic proof of online accounts*. URL: https://tlsnotary.org/.

[379] Saar Tochner, Stefan Schmid, and Aviv Zohar. "Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff". In: *CoRR* abs/1909.06890 (2019). URL: https://arxiv.org/abs/1909.06890.

[380] Peter Todd. *BIP-65. OP_CHECKLOCKTIMEVERIFY*. 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki.

[381] Tor. *Tor website*. URL: https://www.torproject.org/.

[382] *Tornado Cash*. 2020. URL: https://tornado.cash/.

[383] *Tradle*. URL: https://tradle.io/.

[384] Florian Tramèr, Dan Boneh, and Kenneth G. Paterson. "Remote Side-Channel Attacks on Anonymous Transactions". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 220. URL: https://eprint.iacr.org/2020/220.

[385] Lawrence J. Trautman. "Virtual Currencies: Bitcoin and What Now after Liberty Reserve and Silk Road?" In: *Richmond Journal of Law and Technology* 20 (4 2014). ISSN: 1556-5068. DOI: 10.2139/ssrn.2393537.

[386] Omer Tripp et al. "TAJ: effective taint analysis of web applications". In: *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*. Ed. by Michael Hind and Amer Diwan. ACM, 2009, pp. 87–97. DOI: 10.1145/1542476.1542486.

[387] Petar Tsankov et al. "Securify: Practical Security Analysis of Smart Contracts". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 67–82. DOI: 10.1145/3243734.3243780.

[388] Florian Tschorsch and Björn Scheuermann. "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies". In: *IEEE Commun. Surv. Tutorials* 18.3 (2016), pp. 2084–2123. DOI: 10.1109/COMST.2016.2535718.

[389] tyzbit. *Samourai Wallet needs access to your node's JSON-RPC port if you're using a Trusted Node... why? - Reddit*. 2017. URL: https://www.reddit.com/r/Bitcoin/comments/7qalbo/ (visited on 2018-12-05).

[390] *Uport*. URL: https://www.uport.me/.

[391] *US securities and exchange comission. Accredited Investors*. URL: https://www.sec.gov/fast-answers/answers-accredhtm.html.

[392] user3643. *Is Monero in I2P secure now, and how do I do it?* Oct. 2017. URL: https://monero.stackexchange.com/q/6264/4089.

[393] Luke Valenta and Brendan Rowan. "Blindcoin: Blinded, Accountable Mixes for Bitcoin". In: *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 8976. Lecture Notes in Computer Science. Springer, 2015, pp. 112–126. DOI: 10.1007/978-3-662-48051-9_9.

[394] Filippo Valsorda. *I'm giving up on PGP*. 2016. URL: https://blog.filippo.io/giving-up-on-long-term-pgp/.

[395] Niels Vandezande. "Virtual currencies under EU anti-money laundering law". In: 33 (2017), pp. 341–353. ISSN: 0267-3649. DOI: 10.1016/j.clsr.2017.03.011.

[396] Shaileshh Bojja Venkatakrishnan, Giulia C. Fanti, and Pramod Viswanath. "Dandelion: Redesigning the Bitcoin Network for Anonymity". In: *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, Urbana-Champaign, IL, USA, June 05 - 09, 2017*. Ed. by Bruce E. Hajek et al. ACM, 2017, p. 57. DOI: 10.1145/3078505.3078528.

[397] Friedhelm Victor and Bianca Katharina Lüders. "Measuring Ethereum-Based ERC20 Token Networks". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 113–129. DOI: 10.1007/978-3-030-32101-7_8.

[398] Fabian Vogelsteller et al. *Merkle Patricia Trie Specification*. 2017. URL: https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[399] *Vyper*. 2020. URL: https://vyper.readthedocs.io/en/latest/.

[400] Walletexplorer. *Bitcoin block explorer with address grouping and wallet labeling*. URL: https://www.walletexplorer.com/.

[401] Channing Walton. *Scala Contracts Project*. 2012. URL: https://github.com/channingwalton/scala-contracts/wiki.

[402] Haiyang Wang et al. "Enhancing Traffic Locality in BitTorrent via Shared Trackers". In: *NETWORKING 2012 - 11th International IFIP TC 6 Networking Conference, Prague, Czech Republic, May 21-25, 2012, Proceedings, Part II*. Ed. by Robert Bestak et al. Vol. 7290. Lecture Notes in Computer Science. Springer, 2012, pp. 59–70. DOI: 10.1007/978-3-642-30054-7_5.

[403] Liang Wang and Ivan Pustogarov. "Towards Better Understanding of Bitcoin Unreachable Peers". In: *CoRR* abs/1709.06837 (2017). URL: https://arxiv.org/abs/1709.06837.

[404] Lawrence H. White. “The Troubling Suppression of Competition from Alternative Monies: The Cases of the Liberty Dollar and E-Gold”. In: *GMU Working Paper in Economics* (14-06 2014). ISSN: 1556-5068. DOI: 10.2139/ssrn.2406983.

[405] JR Willett et al. *Omni Protocol Specification (formerly Mastercoin)*. 2016. URL: https://github.com/OmniLayer/spec.

[406] Wolfgang Wögerer. *A survey of static program analysis techniques*. Tech. rep. Technische Universität Wien, 2005. URL: https://www.ics.uci.edu/~lopes/teaching/inf212W12/readings/Woegerer-progr-analysis.pdf.

[407] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger*. 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[408] WorldCoinIndex. *WorldCoinIndex*. URL: https://www.worldcoinindex.com/coin/ethereum.

[409] Pieter Wuille. *Replace global trickle node with random delays*. Nov. 2015. URL: https://github.com/bitcoin/bitcoin/pull/7125.

[410] Pieter Wuille, Jonas Nick, and Anthony Towns. *BIP-341. Taproot: SegWit version 1 spending rules*. 2020. URL: https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki.

[411] Pieter Wuille, Andrew Poelstra, and Sanket Kanjalkar. *Miniscript*. 2019. URL: http://bitcoin.sipa.be/miniscript/.

[412] Xazax310. *Time to discuss the elephant in the room. Nicehash 51% Attacks.* 2019. URL: https://www.reddit.com/r/gpumining/comments/ael09k/time_to_discuss_the_elephant_in_the_room_nicehash/.

[413] ydtm. *The bug which the DAO hacker exploited was not merely in the DAO itself.* 2016. URL: https://redd.it/4opjov.

[414] Masahiro Yoshida and Akihiro Nakao. “BPEX: Localizing BitTorrent Traffic via Biased Peer Exchange”. In: *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2012, Victoria, BC, Canada, November 12-14, 2012*. Ed. by Fatos Xhafa, Leonard Barolli, and Kin Fun Li. IEEE, 2012, pp. 41–48. DOI: 10.1109/3PGCIC.2012.15.

[415] Zcash. *Zcash official website.* URL: https://z.cash/.

[416] Roman Zeyde. *Bitcoin Full Node on AWS Free Tier*. 2018. URL: https://gist.github.com/romanz/17ff716f13a34df49ff4.

[417] Fan Zhang et al. “Town Crier: An Authenticated Data Feed for Smart Contracts”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 270–282. DOI: 10.1145/2976749.2978326.

[418] ZmnSCPxj. *Improving Lightning Network Pathfinding Latency by Path Splicing and Other Real-Time Strategy Game Techniques*. 2019. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-August/002095.html.

[419] ZmnSCPxj. *Outsourcing route computation with trampoline payments*. 2019. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-April/001950.html.

[420] ZmnSCPxj. *Proposal: routetricks plugin*. 2019. URL: https://github.com/ElementsProject/lightning/issues/3001 (visited on 2019-10-31).

List of Publications

1. Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. “Findel: Secure Derivative Contracts for Ethereum”. In: *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*. Ed. by Michael Brenner et al. Vol. 10323. Lecture Notes in Computer Science. Springer, 2017, pp. 453–467. DOI: 10.1007/978-3-319-70278-0_28. URL: https://hdl.handle.net/10993/30975 ([38])

2. Sergei Tikhomirov. “Ethereum: State of Knowledge and Research Perspectives”. In: *Foundations and Practice of Security - 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers*. Ed. by Abdessamad Imine et al. Vol. 10723. Lecture Notes in Computer Science. Springer, 2017, pp. 206–221. DOI: 10.1007/978-3-319-75650-9_14. URL: https://hdl.handle.net/10993/32468 ([374])

3. Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. “Privacy-preserving KYC on Ethereum”. In: *Proceedings of 1st ERCIM Blockchain Workshop 2018*. European Society for Socially Embedded Technologies (EUSSET), 2018. DOI: 10.18420/blockchain2018_09. URL: https://hdl.handle.net/10993/35915 ([39])

4. Sergei Tikhomirov et al. “SmartCheck: Static Analysis of Ethereum Smart Contracts”. In: *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018, Gothenburg, Sweden, May 27 - June 3, 2018*. ACM, 2018, pp. 9–16. URL: https://hdl.handle.net/10993/35862 ([377])

5. Alex Biryukov and Sergei Tikhomirov. “Transaction Clustering Using Network Traffic Analysis for Bitcoin and Derived Blockchains”. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 2019, pp. 204–209. DOI: 10.1109/INFCOMW.2019.8845213. URL: https://hdl.handle.net/10993/39728 ([43])

6. Alex Biryukov and Sergei Tikhomirov. “Deanonymization and Linkability of Cryptocurrency Transactions Based on Network Analysis”. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 172–184. DOI: 10.1109/EuroSP.2019.00022. URL: https://hdl.handle.net/10993/39724 ([41])

7. Alex Biryukov and Sergei Tikhomirov. “Security and privacy of mobile wallet users in Bitcoin, Dash, Monero, and Zcash”. In: *Pervasive Mob. Comput.* 59 (2019). DOI: 10.1016/j.pmcj.2019.101030. URL: https://hdl.handle.net/10993/39729 ([42])

8. Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. "A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network". In: *2020 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, September 7-11, 2020*. IEEE, 2020. URL: `https://eprint.iacr.org/2020/303` ([375])

9. Sergei Tikhomirov et al. "Probing Channel Balances in the Lightning Network". In: *CoRR* abs/2004.00333 (2020). URL: `https://arxiv.org/abs/2004.00333` ([376])