Diss. ETH No. 24057

# On the Security, Performance and Privacy of Proof of Work Blockchains

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Arthur Gervais

Diplôme d'Ingénieur, INSA de Lyon
Master of Science, KTH Stockholm
Master of Science, Aalto University
born 29.03.1987
citizen of France and Germany

accepted on the recommendation of

Prof. Dr. Srdjan Čapkun, examiner
Prof. Dr. Bryan Ford, coexaminer
Prof. Dr. Sarah Meiklejohn, coexaminer
Prof. Dr. Roger Wattenhofer, coexaminer

2016

# Abstract

In this thesis, we examine the security, performance, and privacy of Proof of Work-based (PoW) blockchains and digital currencies such as Bitcoin. The decentralized characteristics of blockchains have the benefit of removing trusted third parties; however, they create new challenges for security, performance, scalability, and privacy, which we investigate. The blockchain's security, for example, affects the ability of participants to exchange monetary value or participate in the network communication and the consensus process.

In our first contribution, we observe the decentralized nature of Bitcoin and show that few individuals typically control vital operations in the Bitcoin ecosystem. Moreover, we show that a third party can unilaterally affect the fungibility of individual Bitcoins. Our second contribution provides a quantitative framework to objectively compare the security and performance characteristics of Proof of Work-based blockchains under adversaries with optimal strategies. Our work allows us to increase Bitcoin's transaction throughput by a factor of ten, given only one parameter change and without deteriorating the security of the underlying blockchain. In our third contribution, we highlight previously unconsidered impacts of the PoW blockchain's scalability on its security and propose design modifications that are now implemented in the primary Bitcoin client. In our fourth contribution, we investigate the privacy of lightweight Bitcoin clients, those that are the most critical to Bitcoin's mainstream adoption. Similarly, we propose appropriate design modifications that are being implemented to protect the user's privacy. Orthogonally, in our fifth contribution, we analyze the location privacy implications of public transaction prices. Surprisingly, we show that, given only a few prices from a consumer, we can accurately position the purchase location.

# Zusammenfassung

In dieser Arbeit untersuchen wir die Sicherheit, Leistungsfähigkeit und Privatsphäre von Proof of Work-basierten (PoW) Blockchains und digitalen Währungen wie Bitcoin. Die dezentralen Merkmale von Blockchains haben den Vorteil, trusted third parties zu entfernen; jedoch schaffen sie neue Herausforderungen für Sicherheit, Leistungsfähigkeit, Skalierbarkeit und Privatsphäre, die wir untersuchen. Die Sicherheit der Blockchain beeinflusst beispielsweise die Fähigkeit der Teilnehmer, Geldwerte auszutauschen oder an der Kommunikation im Netzwerk und dem Konsensus Prozess teilzunehmen.

In unserem ersten Beitrag beobachten wir die dezentrale Natur von Bitcoin und zeigen, dass wenige Individuen in der Regel lebenswichtige Operationen im Bitcoin-Ökosystem kontrollieren. Darüber hinaus zeigen wir, dass ein Dritter die Fungibilität einzelner Bitcoins einseitig beeinflussen kann. Unser zweiter Beitrag liefert eine quantitative Methode, um die Sicherheits- und Leistungsmerkmale von Proof of Work-basierten Blockchains unter Gegnern mit optimalen Strategien zu vergleichen. Unsere Arbeit ermöglicht es uns, den Transaktionsdurchsatz von Bitcoin um den Faktor zehn zu erhöhen, mithilfe nur einer Parameteränderung und ohne die Sicherheit der zugrundeliegenden Blockchain zu verschlechtern. In unserem dritten Beitrag heben wir die bisher unberücksichtigten Auswirkungen der Skalierbarkeit der PoW-Blockchain auf ihre Sicherheit hervor und schlagen Konstruktionsänderungen vor, die nun im primären Bitcoin Klienten implementiert sind. In unserem vierten Beitrag untersuchen wir die Privatsphäre von ressourcenschonenden Bitcoin Implementationen, die für die Mainstream-Adoption von Bitcoin am wichtigsten sind. Ebenso schlagen wir geeignete Designänderungen vor, die implementiert werden, um die Privatsphäre der Benutzer zu schützen. Parallel hierzu, analysieren wir in unserem fünften Beitrag, die Implikationen von öffentlichen Transaktionspreise

auf die Privatsphäre der Käufer. Überraschenderweise zeigen wir, dass wir mit nur wenigen Kaufpreisen eines Konsumenten, die Position des Einkaufsortes genau positionieren können.

# Resumé

Dans cette thèse, nous examinons la sécurité, performance et vie privée de blockchains basé sur le Proof of Work (PoW) comme par exemple la monnaie digitale Bitcoin. Les caractéristiques décentralisées de blockchains ont l'avantage d'enlever des tiers de confiance ; ils créent néanmoins de nouvelles défis concernant la sécurité, performance, évolutivité, et vie privée que nous étudions. La sécurité de la blockchain par exemple influence la possibilité des participants à échanger de la valeur monétaire ou généralement à participer à la communication réseau et au processus d'atteindre un consensus.

Dans notre première contribution, nous observons la nature décentralisée de Bitcoin et montrons que quelques individus contrôlent généralement les opérations vitales dans l'écosystème Bitcoin. De plus, nous montrons qu'une partie tierce peut unilatéralement affecter la fongibilité de Bitcoins individuels. Notre deuxième contribution fournit une méthodologie quantitative pour comparer objectivement les caractéristiques de sécurité et de performance de blockchains basé sur le Proof of Work en tenant compte d'adversaires avec des stratégies optimales. Notre travail nous permet d'augmenter le débit de transaction de Bitcoin par un facteur de dix, avec qu'un seul changement de paramètre et sans détérioration de la sécurité de la blockchain sous-jacent. Dans notre troisième contribution, nous mettons en évidence les impacts précédemment inconsidérés de l'évolutivité de PoW blockchain sur sa sécurité et nous proposons des modifications de conception qui sont maintenant mises en œuvre dans le client Bitcoin primaire. Dans notre quatrième contribution, nous étudions la vie privée des clients Bitcoin légers, ceux qui sont les plus critiques à l'adoption générale de Bitcoin. De même, nous proposons des modifications de conception appropriées qui sont mises en œuvre pour protéger la vie privée des l'utilisateurs. En parallèle, dans notre cinquième contribution, nous analysons les impli-

cations des prix de transaction publics concernant la confidentialité de la localisation. Étonnamment, nous montrons que, étant donné que quelques prix d'un consommateur, nous pouvons positionner avec précision l'emplacement d'achat.

# Acknowledgments

Primarily, I wish to give my deepest gratitude to my supervisor Srdjan Čapkun for his education, guidance, and support in conceiving this work. The years of work, thinking, and communication have taught me to organize my ideas with more perspective and separate dead-end research from promising research avenues. I wish to extend my gratitude further to my Master advisors Tuomas Aura, Peter Sjödin and Youakim Badr that helped me beginning my academic career.

I wish to express my thanks and appreciation to my external committee members — Bryan Ford, Sarah Meiklejohn, and Roger Wattenhofer — for accepting to read through this thesis, taking their time to provide constructive criticism, helpful comments and allowing me to defend my work.

The results of this thesis owe a large debt of gratitude to my collaborators who supported me heavily in its creation; besides my doctoral advisor, foremost my colleague Hubert Ritzdorf, my external advisors Ghassan Karame, Vincent Lenders, Vedran Čapkun, Elli Androulaki, Mario Lucic as well as my students Karl Wüst, Vasileios Glykantzis and Damian Gruber.

I wish to thank all those who commented on ideas, presentations, papers, drafts that this thesis is built from. The multi-cultural and open work attitude at ETH Zurich provides a very positive environment for research. In alphabetical order the helpful individuals are Aritra Dhar, Marco Guarnieri, Nikolaos Karapanos, Kari Kostiainen, Claudio Marforio, Ognjen Maric, Sinisa Matetic, Aanjhan Ranganathan, Joel Reardon, Hubert Ritzdorf, David Sommer, Claudio Soriente, Petar Tsankov, and Der-Yeuan Yu.

I moreover wish to acknowledge my funder Armasuisse and Vincent Lenders whose financial support and regular discussions facilitated this work. The generous financial and critical thinking support is greatly

appreciated.

Our research institute's group secretary Barbara Pfändner, deserves particular praise, because she easily, immediately and perfectly handled the numerous bureaucratic issues that arose during our work.

Finally, I wish to extend my deep personal heartfelt gratitude to my parents Ute Richter-Gervais and Lionel Gervais, my sister Lukretia Gervais and to Esther González Martínez, whose love, and support in particular in challenging moments was of great importance for me. Without the loving care of my parents and support of my sister, I would not have made it this far. In particularly a significant amount of the creative work of this thesis would not have been possible without the adventures, cultural exposures and late evening flights to Madrid, Spain.

# Contents

# List of Figures

# List of Tables

# Part I

# Introduction and Background

# Chapter 1

# Introduction

The scope of this thesis is Proof of Work (PoW) blockchain security, privacy, performance and observations about its (de-)centralized ecosystem. Because the scope of this field is substantially wider and complex than a thesis might capture, our work focuses in particular on open and decentralized blockchains (e.g., Bitcoin, Ethereum), it's security, lightweight client privacy and the resulting location privacy leakages of transaction prices.

The concept of PoW blockchains is known in the scientific literature since the inception of Bitcoin by Satoshi Nakamoto in the year 2009 [125]. Bitcoin, its blockchain, and smart contracts have fueled innovation and allowed many novel ideas to emerge. The blockchain's decentralized nature enables mutually distrusting peers to exchange information, trade and enforce contracts without the need for a trusted third party. With over 12B$ in market capitalization and more than 1.4B$ in venture capital investment, the security properties of blockchain have clearly forged new business sectors while challenging long-established traditional institutions.

The nearly 1000 Bitcoin forks [36], Ethereum [62], Stellar [107], Ripple [30] or Litecoin [41] are only a few examples of alternative blockchains that have been proposed recently. Whether explicitly stated or implicitly assumed, and however named, a blockchain is a distributed database, or ledger [30], that periodically accumulates transactions within a block that are written to its data storage. Blocks are cryptographically linked, such that each block refers to its parent block, forming a blockchain. The blockchain, therefore, provides a *tamper-proof*,

*integrity* protected distributed database where the participants agree in consensus about its data content. More importantly, the blockchain solves the *double-spending* problem in a decentralized setting, i. e., a monetary value (e.g., a token, coin or bill) cannot be spent more than once.

In the physical world, the importance of the double-spending protection is well accepted: once a cash payment performed, the payer is not able to spend the transferred money again without physically stealing the money from the payee. Similarly, the importance of tamper-proof, robust and integer archive or data storage is considered significant: companies, for instance, are supposed to store their incoming, and outgoing cash flows over several years if the tax bureau wishes to verify the claimed profits. Physical money, i. e., bills, and coins moreover typically do not suffer from the so-called double-spending problem — i. e., a coin cannot be spent twice without physically stealing the coin from a merchant after purchase.

In the digital world, the double-spending problem is typically solved with the use of digital signatures and a centralized entity, e. g., a bank. Although these systems make use of a central body, electronic cash systems can be constructed in a privacy-preserving manner: ECash [59,63, 65] allows user anonymity and transaction unlinkability, such that payments of the same user cannot be linked unless the user misbehaves. Funds can still, however, be frozen by the bank, and these systems, therefore, do not solve the double-spending problem in a decentralized and censorship-resistant manner. Similarly, the importance of integer data logs is also well recognized in the digital world. Corporate or legislative requirements, in particular, classified or privileged communication may require the integer storage of corporate activities over decades [38].

Freeing electronic cash from a central entity, providing a decentralized mechanism for the exchange of monetary value and dispute mediation, while still preserving elements of anonymity is the goal of Bitcoin like blockchains. Surprisingly, the underlying innovation of Bitcoin and PoW blockchains does not lie in the creation or usage of new cryptographic techniques. It is rather their subtly engineered combination, which allowed Bitcoin, cryptocurrencies and PoW blockchains to grow to a billion dollar industry [36]. Due to its recent introduction, decentralized blockchains pose, in particular for research, previously unknown opportunities and challenges from a security and privacy perspective that we study in this thesis.

## 1.1 Scope

In the following we outline the scope of this work, as well as which related lines of research are not covered in this thesis.

The *security* of a blockchain is tightly coupled with the integrity of the ledger. Transactions are typically considered to be *securely* written to a blockchain, if the transactions are unlikely to be reversed in the future, i.e., the blockchain's data history is unlikely to change. As such, a blockchain acts as a *tamper-proof*, *append-only* transaction log that maintains the history of all transactions. Coupled with the resilience against double-spending, i. e., only one out of a set of $n$ conflicting transactions is ultimately accepted to the ledger, blockchains are naturally fitted to provide the backend of decentralized electronic currencies [62, 82, 125]. In this thesis, we investigate the security provisions of PoW blockchains.

The *performance* of a blockchain is typically equivalent to the number of transactions per second that the blockchain can clear, i.e., write persistently to its data storage. Performance in PoW blockchains can be increased by for instance lowering the interval at which content is written to the blockchain — the block time interval. Performance enhancements, however, have implications for the security of the system that we understand, analyze and quantify within this thesis.

A blockchain transaction is supposed to be irreversibly stored to the ledger. Due to this persistent and open storage of transaction history, decentralized public blockchains face a critical *privacy* issue: all transaction amounts, senders, receivers and the time of payment are publicly known. Although senders and receivers are obfuscated to some extend through pseudonyms — commonly referred to as addresses, their behavior is linkable and once the pseudonymity is deanonymized, all their transactions can be identified. This thesis presents heuristics for clustering Bitcoin addresses.

We identify two main blockchain client types: (i) full and (ii) lightweight blockchain clients. Full blockchain clients typically consume significant resources regarding CPU processing, network bandwidth, storage space and require technical knowledge to setup and maintain such a client. The average mainstream consumer, however, is typically not willing to invest many resources into the operation of a blockchain or payment client. To overcome the challenge of mass-adoption, decentralized blockchains (e.g. Bitcoin) provide so-called lightweight clients

(LWC) that are supposed to provide similar security guarantees as full blockchain clients, but in general do not contribute to the growth and stability of the underlying network. To operate as efficiently as possible, lightweight clients sacrifice the user's privacy as we show in this thesis.

The purpose of Proof of Work blockchains is to be *decentralized*, such that no single entity controls which transactions are included in the blockchain. As opposed to traditional consensus mechanisms, such as Practical Byzantine Fault Tolerant protocols (PBFT) [64, 96, 107, 137], participants vote in Proof of Work with their computing power — hence the term Proof of Work. At the time of writing, the PoW consensus mechanism accounts for over 90% of the market capitalization of crypto currencies [36], therefore effectively being the most widely used consensus mechanism for open and decentralized digital currencies.

Before highlighting the topics covered by this thesis, we wish to list related works that are not covered within this thesis. Some works are orthogonal to our contributions, while others are higher-level concepts that build on the provisions and capabilities given by PoW blockchains.

- *Centralized blockchains*

  Hierarchical [30] or permission led blockchains [39,73,96] are built such that one or several entities have superior rights compared to the other blockchain participants. These supernodes decide for instance on which nodes are allowed to join the blockchain system, or which nodes are eligible to send transactions. We do not consider centralized blockchains and focus on open PoW based blockchains.

- *Smart Contract Applications, Security and Privacy*

  A smart contract is typically referred to as a program that is executed on top of the blockchain [62]. Ethereum, for instance, supports a Turing-complete programming language, such that the blockchain's functionality can be extended beyond the transfer of funds between participants. Bitcoin supports a stack-based programming language called Script, which similarly allows the implementation of advanced features such as a peer-to-peer crowdfunding application [40].

  Although smart contracts provide a seemingly innovative technique for interactions among participants without trust requirements, they open up a few new challenges. The code and data

of a smart contract are publicly visible in the blockchain; smart contract privacy is, therefore, to be considered [103]. More critical however is the question whether the intended execution of the smart contract matches the desired specifications. In the year 2016, Ethereum manually reverted the history of the blockchain (i. e., performed a *hard-fork*) due to a smart contract that didn't execute as intended and which was exploited by a malicious adversary; at the time this contract contained over 150 million USD [42, 45]. Smart contracts are built on top of the blockchain and can be considered higher-level than our work.

- *Building Privacy Preserving PoW Blockchains*

  Bitcoin, its forks, and Ethereum use addresses, a cryptographic hash of a public key, as an identifier for accounts that hold monetary value. The accounts can transfer funds via transactions that are inherently linkable, because the blockchain stores the time, value, senders, and recipients of the transaction. Such information clearly threatens the privacy of the blockchain participants.

  Related work has shown efforts to remove the linkability and value information of transactions by taking advantage of modern cryptographic primitives such as zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) [53, 109]. In our work, we do not attempt to construct a blockchain that hides transaction correspondents or amounts.

- *Legal and Regulatory Aspects of Cryptocurrencies*

  A decentralized currency, where no government or legal entity should technically be able to freeze or collect funds challenges the current legislative institutions. Participants are moreover not only pseudo-anonymous, such that their identity is not revealed immediately but also, any individual can transact at will with the decentralized cryptocurrency. Due to these properties, Bitcoin has been used by marketplaces for illegal merchandise (e.g., drugs, weapons [44]) as well as a payment method for ransomware [37]. We do not consider the impact of decentralized electronic currencies on legislative and regulatory processes.

- *Economic Aspects of Cryptocurrencies*

  Bitcoin and the emergence of many alternative Bitcoin forks, commonly referred to as altcoins [36], is partially motivated by the

monetary reward that one can gain for first creating and mining an altcoin. Gambling services moreover rank among the most popular applications of Bitcoin [43]. Whether gambling on the blockchain or through the speculative investments in cryptocurrencies, the open nature of Bitcoin clearly supports economy related research. Within this thesis, we do not consider the economic aspects of PoW blockchains.

- *Proof of Work alternatives*

  The best-known solution to solving the PoW puzzle is brute force search, which is computationally expensive and many PoW alternatives have therefore been proposed. In *Proof of Stake* (PoS) [2], the voting power of peers is based on the amount of "stake" they own in the respective blockchain system. *Proof of Burn* (PoB) is a proposal to replace PoW by burning transaction outputs, such that they can no longer be spent. Existing PoB-based blockchains, however, rely on PoW to create blocks and therefore ultimately rely on PoW for coin creation. *Proof of Capacity* (PoC), aims to use the available hard-disk space to replace PoW. The literature features some additional proposals [83, 96, 102, 107, 137] that rely on traditional Byzantine fault tolerant consensus protocols in the hope to increase the consensus efficiency and achieve high transactional throughput. Recent studies propose to combine the use of PoW with BFT protocols to realize highly-performant, open consensus protocols (Byzcoin [102]). We consider developing a PoW alternative as an orthogonal topic and do not attempt its exploration.

## 1.2 Contributions and Publications

The contributions of this thesis are the following:

- We analyze the (de-)centralized nature of Bitcoin and show that — contrary to widespread belief — Bitcoin is not a truly decentralized system as it is deployed and implemented today.
- We analyze the impact of changing the block size and/or the block interval on selfish mining and double-spending. We, therefore, quantify the double-spending resilience of PoW blockchains when subject to rational adversaries and objectively compare the

security of different PoW blockchains with respect to the required number of transaction confirmations. Similarly, we quantify the selfish mining resilience of PoW blockchains. Besides, we show that the higher the block reward of a blockchain (in $) the more resilient it is against double-spending.

- We provide a scalable and parameterizable open source Bitcoin simulator. The simulator allows to measure the network propagation impact of reparametrizations of Bitcoin (e.g., the block interval times and the block sizes). We simulated a broad range of possible parameter combinations and concluded that Bitcoin could scale its transactions per second limit by a factor of ten, with only one parameter change and without sacrificing its security provisions (confirmed by our security model).

- We analytically and experimentally confirm that a resource constrained adversary can abuse existing scalability measures adopted in Bitcoin clients to deny information about newly generated blocks and transactions to Bitcoin nodes for at least 20 minutes. We then extend this basic attack and show how an adversary can continuously deny the delivery of such information. We validate our analysis through implementation using a handful of hosts located around the globe. Our results demonstrate the feasibility and easy realization of our attacks in current Bitcoin client implementations. We show that our results allow the adversary to increase its mining advantage in the network considerably, double-spend transactions despite the current countermeasures adopted by Bitcoin, and easily mount Denial-of-Service attacks. We propose several mitigations for hardening the security of the network against such a misbehavior without deteriorating the scalability of Bitcoin. Namely, we propose a modification of the block request management system in Bitcoin to detect any misconduct in the delivery of blocks. Additionally, we leverage our findings to estimate the minimum amount of waiting time required to ensure, with significant probability, the security of fast payments in Bitcoin.

- We show that lightweight client implementations (SPV) reveal relevant information about the client's Bitcoin addresses. Bitcoin addresses of users who possess a modest number of addresses (e.g., $< 20$) are leaked by a single Bloom filter in existing lightweight

(SPV) clients. We additionally show that an adversary can easily link different Bloom filters which embed the same elements—irrespective of the target false positive rate. This also enables the adversary to link, with high confidence, different Bloom filters which pertain to the same originator. We show that a considerable number of the addresses of users are leaked if the adversary can collect at least two Bloom filters issued by the same SPV client—irrespective of the target false positive rate and the number of user addresses. Finally, we propose a lightweight and efficient countermeasures to enhance the privacy offered by SPV clients. Our countermeasures are being integrated within existing SPV client implementations.

- We propose a generic quantitative framework for evaluating attacks against the location privacy of consumer purchases. We validate our framework on three independent price datasets of real-world consumer prices and show that location information can be extracted reliably. We introduce three privacy metrics to capture the performance of the adversary in the attack as well as the extent to which location privacy of consumers is reduced when the adversary has access to a specific dataset of purchases.

Most of the work presented in this thesis is based on the following co-authored publications:

1. Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, Srdjan Capkun, "On the Security and Performance of Proof of Work Blockchains", *In Proceedings of the 23nd ACM Conference on Computer and Communication Security (CCS)*, 2016

2. Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, Srdjan Capkun, "Tampering with the Delivery of Blocks and Transactions in Bitcoin", *In Proceedings of the 22nd ACM Conference on Computer and Communication Security (CCS)*, 2015

3. Arthur Gervais, Hubert Ritzdorf, Mario Lucic, Vincent Lenders, Srdjan Capkun, "Quantifying Location Privacy Leakage from Transaction Prices", *In Proceedings of the 21th European Symposium on Research in Computer Security (ESORICS)*, 2016

4. Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, Srdjan Capkun, "Misbehavior in Bitcoin: A Study of Double-Spending and Accountability", *In ACM Transactions on Information and System Security (TISSEC)*, 2014

5. Arthur Gervais, Ghassan Karame, Damian Gruber, Srdjan Capkun, "On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients", *In Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC)*, 2014

6. Arthur Gervais, Ghassan O. Karame, Srdjan Capkun, Vedran Capkun, "Is Bitcoin a Decentralized Currency?", *In IEEE Security and Privacy Magazine*, 2014

## 1.3 Organization and Structure

This thesis is organized into five parts and nine chapters.

Chapter 2 introduces the blockchain, Bitcoin, Proof of Work and the required cryptographic concepts necessary for its understanding. We cover Bitcoin's different transaction types and moreover dive into the specific ways of how Bitcoin transactions are evaluated, i.e., explain how the Script language is executed. The blockchain is built through communication on the network layer where we highlight how peers exchange blockchain relevant information.

Chapter 3 surveys and organizes related work in different categories, such as security, privacy, network based and PoW alternatives. We highlight and challenge the various adversarial assumptions that related work has adopted. Also, we contrast related work to our contributions to explicitly state our contributions.

Chapter 4 observes that the necessary operations of the Bitcoin blockchain are in fact not decentralized but typically in control of a few individuals. These individuals are moreover not bound to any specific regulations nor are required to provide a methodological transparency on their income or decision process.

Chapter 5 provides to our knowledge the first realistic framework that allows quantifying the security provisions of different PoW blockchain instantiations (e.g., Bitcoin, Litecoin, Dogecoin or Ethereum). In addition to the framework, we provide a blockchain simulator that allows simulating different blockchain instances. Given the framework, we can

judge about their security provisions — which ultimately makes it possible to create a more performant PoW blockchain without deteriorating its security.

Chapter 6 presents a powerful blockchain network partitioning attack that allows an adversary to delay the propagation of blocks and transaction for at least 20 minutes up to an indefinite amount of time. This is made possible due to specific scalability measures in place for the Bitcoin network to scale. We moreover show how an adversary can take advantage of the attacks to perform double-spending, aggravated selfish mining or affordable network-wide denial of service attacks. Our proposed design changes to mitigate the vulnerabilities have been implemented in the Bitcoin code base.

Chapter 7 is the first work that analytically and experimentally evaluates the privacy provisions of bloom filters in lightweight Bitcoin clients. We show how an adversary can in most cases already identify with one bloom filter all Bitcoin addresses belonging to the Bitcoin wallet of a lightweight client. Our proposed mitigations are being implemented in alternative Bitcoin client implementations.

Chapter 8 shows that given only a transaction price of a bought product, location information of the purchase is leaked. Our location privacy evaluation is performed given three datasets that capture worldwide, country-wide and metropolitan-wide location granularity. Our work motivates the need to hide transaction amounts for location privacy appropriately.

Chapter 9 concludes the thesis by reviewing our contributions and outlining avenues for future research. Finally, we draw conclusions and summarize our work.

# Chapter 2

# Bitcoin internals

## 2.1 Introduction

With the publication of the Bitcoin whitepaper in 2009 and the subsequent delivery of the first working implementation of Bitcoin 2 months later, the individual or group behind the pseudonym Satoshi Nakamoto, has started to prove that a new class of decentralized currency is practical. Although its founder has officially vanished from the day-to-day development only 1.5 years after its release, the Bitcoin community persisted and grew from zero to an over 12 billion industry at the time of writing.

Bitcoin is an electronic payment system that allows two or more parties to exchange monetary values among themselves without passing through intermediaries (such as banks or payment processors). Due to a missing centralized entity, all nodes in the Bitcoin network have to reach consensus about the transactions that are considered valid. Specifically, nodes reach consensus given two mechanisms: (i) all nodes in the Bitcoin network receive all transactions through a broadcast protocol, and (ii) the consensus algorithm allows all participants to synchronize about their perceived reality in regular time-intervals. Nodes differ in their view of reality if different or conflicting transactions are considered valid. The broadcast mechanism is realized on top of TCP/IP, such that any node connected to the Internet is technically capable of receiving and sending transactions through the Bitcoin network. This design has the advantage, that it enables any individual, regardless of the re-

spective geographical location, to participate in monetary transactions, might, however, suffer from scalability challenges that are inherent to broadcast communication. Bitcoin's consensus algorithm consists of an expensive computational challenge that needs to be solved periodically by its participants. The node that solves the challenge broadcasts the solution in the network and therefore timestamps the current consensus among the nodes.

To send a payment from one participant $\mathcal{A}$ to another participant $\mathcal{B}$, the participant $\mathcal{A}$ is required to create a transaction paying from his *address* $\mathcal{X}$ to the address $\mathcal{Y}$ of participant $\mathcal{B}$. $\mathcal{A}$ then broadcasts the transaction such that all peers in the Bitcoin network can receive and validate the transaction. A participant can generate multiple addresses, and an address corresponds to the cryptographic hash of a public key, while the related private key is required to spend funds associated to the given address. Besides one-to-one payments, transactions can as well be paid from one-to-many or many-to-many addresses. The freedom to express different transaction types is realized through a Bitcoin-specific stack-based transaction programming language called *Script*. Before forwarding, every node is supposed to *validate* a received transaction, validation includes basic sanity checks, the correct formulation of the associated script and signature verification. Transactions are signed by the spender, such that the redeemer (and any other participant) can verify the authenticity of the payment. Once a transaction has propagated in the network, and a node solves the consensus challenge, the challenge typically *confirms* the currently unconfirmed transactions in the network. At this point, the participants of the Bitcoin network reached consensus and repeat the same process periodically.

The following sections serve as support for the assimilation and understanding of this thesis. This chapter was used in the development of the book [86].

## 2.2 Data Types

Before diving into particularities of Bitcoin's functionality and to aid its assimilation, the following paragraphs are devoted to the required background knowledge.

### 2.2.1 Cryptographic Tools

The Bitcoin protocol limits its use of cryptographic tools to cryptographic hash functions such as SHA256 [117] and RIPEMD160 [105], Merkle trees [108] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [116].

**Cryptographic hash functions** Hash functions, map an arbitrarily long input byte sequence to a fixed size output, commonly referred to as digest, effectively fingerprinting the input sequence. Cryptographic hashes are designed such that it should be nearly infeasible to both recreate the input sequence, as well as to find two input sequences that map to the same output. Also, ideal cryptographic hash functions are computationally inexpensive to be used, and the output hash always changes when the input sequence is altered.

Cryptographic hash functions are widely used in Bitcoin, e.g., the *id* of a transaction corresponds to the cryptographic hash of the transaction. Hash functions are as well a base component of different types of data structures used in Bitcoin, e.g. Merkle trees.

**Merkle trees** Merkle trees allow to combine multiple cryptographic hash input sequences in a hash tree converging into the topmost Merkle root hash. This data structure allows the compact representation of a set of transactions, e.g., when the tree is built up from the transaction hashes (cf. Figure 2.1).

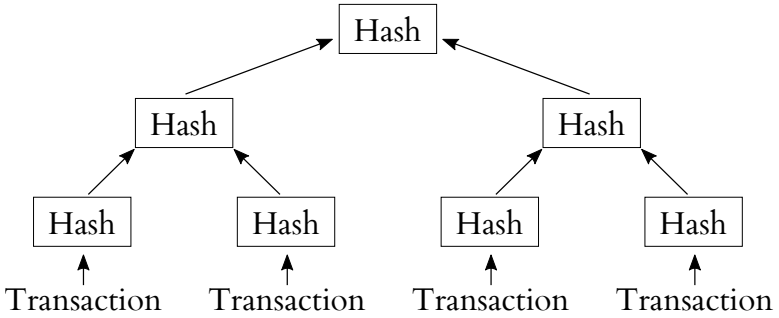



**Figure 2.1:** Merkle tree of transactions. The topmost hash is referred to as merkle root and consists of a compact representation of the involved transactions.

**ECDSA Signature Scheme** Inorder to sign and verify transaction signatures, Bitcoin currently relies on the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve [61]. The required secp256k1 private keys have a length of 256 bits and can be transformed deterministically into a secp256k1 public key. Recall, that nothing in Bitcoin is actually encrypted, ECDSA is rather used by a transaction creator to sign the respective transaction.

ECDSA curves are typically defined by the employed finite field $F_p$, the respective equations used, the elliptic curve base point $G$ and an integer of order $n$ such that $n \times G = O$. Note that we denote by $\times$ the elliptic curve point multiplication.

The respective finite field $F_p$ used by secp256k1 is defined by $p$ as follows (cf. Equation 2.1).

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \tag{2.1}$$

The corresponding curve $y^2 = x^3 + ax + b$ over $F_p$ is defined by $a = 0$ and $b = 7$, resulting in $y^2 = x^3 + 7$ [61]. [1]

In the following, we detail how two entities $U$ and $V$ can sign a message and verify its ECDSA signature.

**ECDSA message signing** In the setup procedure, $U$ and $V$ should agree on the hash functions they want to use [60] (e.g., Bitcoin uses currently sha256). $U$ is furthermore required to generate a private/public ECDSA key pair. The private key is an integer $d_U$ selected at random from $d_U \in [1, n-1]$. From the private key, the public key is derived with the help of the elliptic curve base point $Q_A = d_U \times G$.

The signature over message $m$ is subsequently performed by following the following operations [139]:

1. Calculate $e = hash(m)$, where $hash$ is the cryptographic hash function that has been agreed upon in the setup phase.

2. Denote by $L_n$ the bit length of the group order $n$. Let $z$ be the $L_n$ leftmost bits of $e$.

3. Select a random integer $k \in [1, n-1]$.

[1] For reference, the base point $G$ in compressed form is $G$ = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798, while the $n$ is $n$ = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141 [61]

4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \ mod \ n$. Go to step 3 if $r = 0$.
6. Calculate $s = k^{-1}(z + rd_U) \ mod \ n$. Go to step 3 if $s = 0$.

The resulting signature pair for message $m$ is $(r, s)$.

**ECDSA signature verification** For entity $V$ to verify the authenticity of the signature of $U$ on the message $m$, $V$ is required to have a copy of $V's$ public key. Given the public key, $U$ follows these steps to verify the signature [139]:

1. If $r$ and $s$ are not integers within $[1, n-1]$, the signature is invalid.
2. Calculate $e = hash(m)$, where $hash$ is the cryptographic hash function that has been agreed upon in the setup phase.
3. Let $z$ be the $L_n$ leftmost bits of $e$.
4. Calculate $w = s^{-1} \ mod \ n$.
5. Calculate $u_1 = zw \ mod \ n$ and $u_2 = rw \ mod \ n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_U$.

Only if $r = x_1 \ mod \ n$, the signature $(r, s)$ over $m$ is valid.

### 2.2.2 Bitcoin specific data types

In this section, we are introducing the main Bitcoin-specific data types and concepts necessary for its understanding.

**Script** Bitcoin introduced a custom stack-based scripting language Script in an attempt to allow different types of transactions. Script's flexibility allows extending the functionality of transactions beyond the simple transfer of funds. Script is stack-based, supports many functions (commonly referred to as opcodes) and either evaluates to true or false. The language supports dozens of different opcodes, ranging from simple comparison opcodes to cryptographic hash functions and signature verification. Because Script is supposed to be executed on any validating Bitcoin node, execution time is critical regarding Denial of Service attacks and therefore, many opcodes have been temporarily

disabled. Consequently, Script is kept on purpose simple, and therefore does not support the same complexity as general purpose programming languages.

An example Script program contains two constants (denoted by <...>) and one opcode (execution goes from the left to the right): <signature> <publicKey> OP_CHECKSIG. Constants are pushed by default on the stack, and upon execution, the stack would, therefore, contain <signature> <publicKey>. Then, OP_CHECKSIG is executed which verifies the <signature> under the provided <publicKey>. If the signature matches the provided public key, OP_CHECKSIG returns true, and in return, the script returns true.

**Addresses** An address is a unique identifier that can be both, the origin and the destination of a transfer of Bitcoin through a transaction. Technically, an address corresponds to the Base58 encoded cryptographic hash of a public key [76]. An address's associated Bitcoin balance can be zero or positive, ranging from the smallest unit of 1 satoshi ($10^{-8}$ Bitcoin) to 21 million Bitcoin, the maximum amount of Bitcoins that can be created according to the current consensus. Typically, Bitcoin addresses start with the decimal 1 (for P2PKH transactions), or 3 (for P2SH, e.g., multi-signature addresses) and typically range from 27 to 34 characters.

### 2.2.3 Transactions

Transactions allow participants to exchange funds by specifying three key elements: (i) the funding source(s) and proof that the funds can be spent, (ii) the recipient(s) (iii) the conditions for redemption.

A transaction is a data structure with inputs and outputs (cf. Figure 2.2). An input redeems the Bitcoins that are referenced in a previous transaction output. Transactions therefore effectively form a chain of transactions, and Bitcoins are technically only kept in transaction outputs, not within addresses.

A transaction output specifies, how many Bitcoins it contains as well as under which conditions a subsequent transaction can redeem the output. The subsequent transaction redeems the output of a previous transaction, by encoding the necessary spending information in a transaction input. These conditions under which an output can be spent are encoded with the help of Script, and only the participants that are capable of providing input to Script, such that it evaluates

to true upon execution, are allowed to spend Bitcoins of a particular Bitcoin transaction.

In Figure 2.2, a simplified transaction with one input and two outputs is visualized. This transaction spends Bitcoin from address $X$ to both, address $Y$ and $Z$.

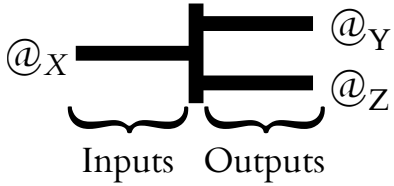



**Figure 2.2:** A transaction has in- and outputs, here one input from address $X$ to adress $Y$ and $Z$.

More precisely, the input from $X$, corresponds to a *former* output that has spent Bitcoins to $X$ (cf. Figure 2.3). Transactions therefore create a chain of transactions. The outputs that have not yet been spent (in this example the two outputs of transaction 2), are commonly referred to as *unspent transaction outputs (UTXO)*.

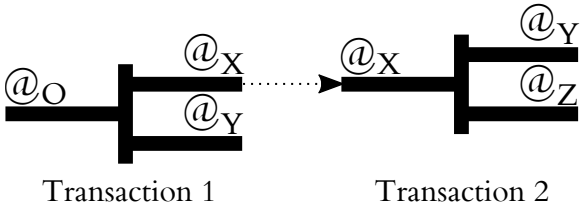



**Figure 2.3:** The input of transaction 2 refers to the output of transaction 1.

In the following, we will discuss the different transaction types that are available in Bitcoin.

**Transaction format** Table 2.1 summarises the general format of a Bitcoin transactions, while Table 2.2 and Table 2.3 respectively describe the transaction input and output format.

**Standard transaction types** Bitcoin supports by default several *standard* transaction types. Only standard transaction types are broadcasted and validated within the network. Transactions that do not

| *Field* | *Description* | *Size* |
|---|---|---|
| Version number | Version, currently 1 | 4 bytes |
| Input counter | positive integer | 1 - 9 bytes |
| List of inputs | cf. Table regarding transaction inputs | Variable |
| Output counter | positive integer | 1 - 9 bytes |
| List of outputs | cf. Table regarding transaction inputs | Variable |
| locktime | Block height or time when transaction is valid | 4 bytes |

**Table 2.1:** Transaction format inside a Bitcoin block.

| *Field* | *Description* | *Size* |
|---|---|---|
| Previous transaction hash | Dependency | 32 bytes |
| Previous transaction output index | Dependency index | 4 bytes |
| Script length | | 1 - 9 bytes |
| ScriptSig | Input script | Variable |
| Sequence number | generally 0xFFFFFFFF | 4 bytes |

**Table 2.2:** Transaction input format inside a Bitcoin block.

| *Field* | *Description* | *Size* |
|---|---|---|
| value | positive integer of Satoshis to be transferred | 4 bytes |
| Script length | | 1 - 9 bytes |
| ScriptSig | Output script | Variable |

**Table 2.3:** Transaction output format inside a Bitcoin block.

match the standard transaction type are generally discarded. Please note, that because transactions can have multiple outputs, the different output types can be combined within one transaction.

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

  `OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG`

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

  `<Sig> <PubKey>`

- **Pay To Script Hash (P2SH):** A P2SH transaction output can only be redeemed by an input that provides a script, that hashes

to the hash of the corresponding output. A P2SH output for example contains the following transaction output:

`OP_HASH160 <Hash160(redeemScript)> OP_EQUALVERIFY`

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

`<sig> <redeemScript>`

P2SH allows to create a transaction where the responsibility for providing the redeem conditions of a transaction is pushed from the sender to the redeemer of the funds. Consequently the sender is not required to pay an excess in transaction fees, if the redeem script happens to be complex and thus big in terms of bytes. In practice P2SH outputs are heavily used for multi-signature (multisig) transactions, but multi-signatures can both be accomplished with M-of-N output scripts as well as P2SH.

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

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

  while the redeeming input follows this structure:

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

**Script execution** Given the background knowledge of how transactions are constructed, we now turn to the process of script execution. To validate a new transaction, the input (signature script) and the output of the previous transaction (pubkey script) are concatenated. Once concatenated, the script is executed according to the Script language. Constants, denoted by <..> are pushed on the stack, and opcodes execute their respective actions by taking into account the topmost stack value.

In Figure 2.4, we visualize the validation of transaction 2, which spends a previous output of transaction 1. The output and input script

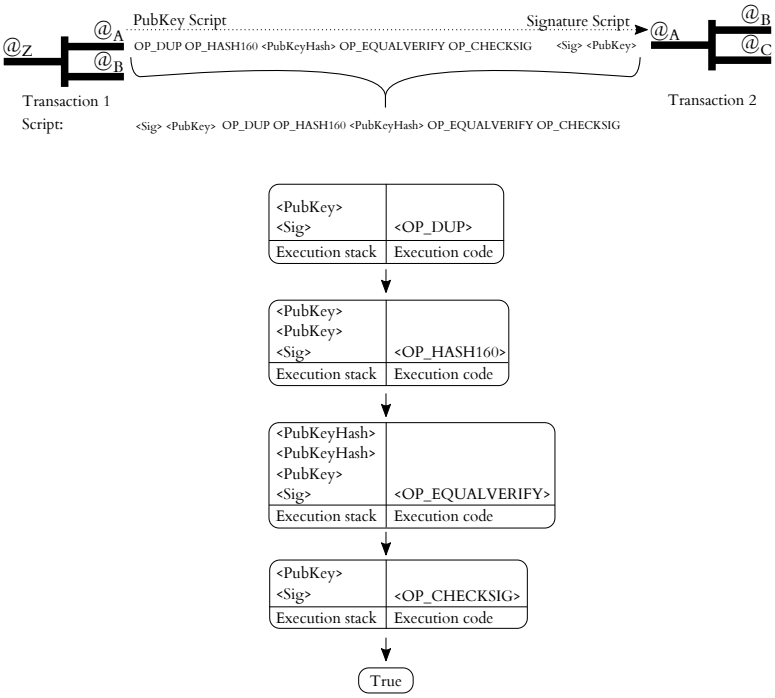



**Figure 2.4:** Script execution for a P2PKH transaction.

are concatenated (Signature script first and then the PubKey script). In a first step, the two constants <PubKey> <Sig> are pushed onto the stack. Then OP_DUP duplicates the top-most stack value, <PubKey> in this case. The next opcode OP_HASH160 hashes the <PubKey> and saves it as <PubKeyHash> on the stack. The constant <PubKeyHash> is pushed onto the stack and OP_EQUALVERIFY verifies if the two top-most stack elements are equal. If they are equal, they are removed from the stack, and the last opcode OP_CHECKSIG verifies if the public key on the stack (<PubKey>) matches the signature (<Sig>). If the signature is valid, the script returns true, meaning that the input of transaction 2 is allowed to spent the output 1 of transaction 1.

**Transaction change and fees** A transaction output requires being spent in its entirety. In practice, however, the output amount is unlikely

to satisfy the exact transaction amount. A transaction output can, therefore, be split into $n$ parts. An output that solely serves the purpose to refund the change to its originator is commonly referred to as *change output*.

As a sanity check, the sum of Bitcoins of the transaction outputs cannot exceed the sum of Bitcoins of the transactions inputs. The sum of Bitcoins of the transaction outputs, however, can be smaller than the sum of Bitcoins of the transaction inputs. This difference is paid as a fee to the Bitcoin miner that includes the transaction within a block.

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

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

### 2.2.4 Blocks

A Bitcoin block is a data structure containing in a simplified view a header with a list of transactions. Each block header has a specific set of fields that we are listed in Table 2.4. In particular, a block header contains a pointer to the previous block, effectively creating a blockchain.

### 2.2.5 Blockchain

As mentioned above, each block contains a pointer to the previous block, the chain of blocks, therefore, constructs the blockchain (cf. Figure 2.5). The blockchain starts with a genesis block that has been generated by the creator of Bitcoin.

The blockchain is extended by appending blocks to the last block that has been added to the blockchain. The process of creating blocks

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

**Table 2.4:** Bitcoin block header format.

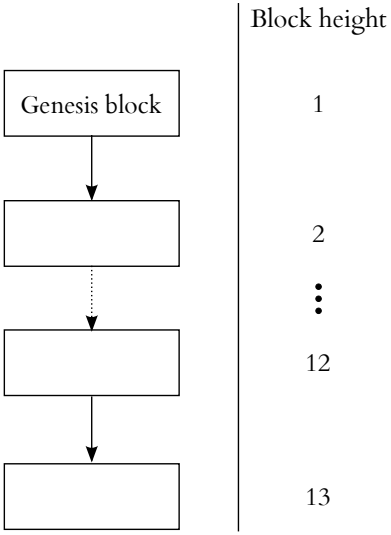



**Figure 2.5:** Chain of blocks, starting with the genesis block forms the blockchain.

is commonly referred to as mining, and it requires to provide a Proof of Work that we detail in the next section.

Because of different miners perform mining, it can happen that competing blocks are created at the same block height. This event is commonly referred to as forking and visualized in Figure 4.2. Eventually, only one blockchain, the longest, can prevail. Blocks that are therefore discarded are referred to as *stale* or orphan blocks.

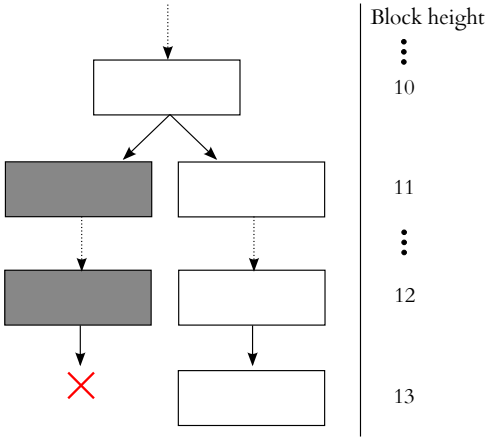



**Figure 2.6:** The blockchain forked and the grey blocks became orphan.

## 2.3 Proof of Work

To achieve consensus among peers in the Bitcoin network, Bitcoin relies on a cryptographic hash based Proof of Work mechanism. A consensus vote is carried out by finding a block, more precisely by finding a block header that hashes to a smaller value than the difficulty of the network requires. The current difficulty is a network wide parameter, which is adjusted dynamically every 2016 blocks to meet an average block generation time of 10 minutes.

More precisely, given a set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin miners need to find a nonce such that the double SHA256 hash is smaller than the target:

$$\text{SHA256}\{\text{SHA256}\{\text{Block}|\text{MerkleRoot}(tx_1,\ldots,tx_N)|\text{Nonce}\}\} \leq \text{target} \tag{2.2}$$

Block denotes the last generated block, $\text{MerkleRoot}(tx_1,\ldots,tx_N)$ denotes the root of the Merkle tree of the transactions that have been included in the block by the miner and, Nonce is the 4-byte nonce, and target is a 32-byte value. To date, the best strategy to solve the PoW mechanism is the brute-force search. The smaller the target becomes, the more difficult is the process of finding a suitable nonce.

Peers that perform the Proof of Work are commonly referred to as

miners. The more hashes a miner can perform, the more likely the miner will find a block, thus the ability to find blocks is proportional to the miner's hashing power. Bitcoin's particular Proof of Work mechanism is to require the double SHA256-hash of the block header content.

## 2.4 Architecture

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

### 2.4.1 Node Types

Due to the variety of different entities that can participate in the Bitcoin ecosystem, multiple nodes types emerged.

**Miner** Miners perform the Proof of Work to find and broadcast blocks in the Bitcoin network. Their operations consist mainly in quickly retrieving information about the newest blocks, validating transactions that are included in new blocks, operate dedicated mining hardware to perform as many hash operations as possible, and to efficiently spread found blocks to the whole network.

Every discovered Bitcoin block provides a monetary reward, for example, 12.5 bitcoins to the respective miner. Because a block, however, is found on average only every 10 minutes, the time until a miner receives a payout can be significant. Miners therefore typically organize themselves into groups of miners, commonly referred to as mining pools. Because of its higher overall hashing power, a mining pool has a higher chance to find a block, and mining pool members can consequently receive payouts more periodically than an independent miner.

**Full node** We define a full Bitcoin node as a node that (i) maintains the full copy of the blockchain, (ii) validates incoming transactions and blocks, and (iii) forwards transactions and blocks to its peers. In addition to providing validation services to the Bitcoin network, a full node might provide an open TCP port (Bitcoin uses the TCP port 8333), where other Bitcoin peers can connect to.

**Lightweight Clients** Lightweight clients are clients that do not maintain the full Bitcoin blockchain, but rather follow the Simple Payment Verification (SPV) scheme. This scheme allows the lightweight client to verify that a transaction has been included in the blockchain, by only receiving the block headers. Besides, lightweight clients do not receive transactions that are irrelevant to their operation, do not need to perform transaction or block validation and consequently require significantly fewer resources to operate than full nodes or miners.

### 2.4.2 Peer-to-Peer Overlay Network

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

**Peer discovery** Upon start, a Bitcoin node typically is not aware of other full node's IP addresses. The client, therefore, queries so-called DNS seeds which provide, for bootstrap purposes, a list of IP addresses of full Bitcoin peers. Different DNS seeds are available, while the Bitcoin core developers actively maintain some.

Once a peer has received a list of full Bitcoin node IP addresses, the peer performs up to 8 outgoing connection attempts. To diversify the number of peers that a node is aware of, a node can request from its peers their known IP addresses of other peers with the *addr* P2P network message. A Bitcoin node that accepts incoming TCP connections by default can operate up to 125 TCP connections.

**Peer connection** Given a known IP address of a Bitcoin node, a peer attempts to connect via a standard TCP/IP connection on TCP port 8333. Once the TCP connection is established, following the basic three-way TCP handshake, the peers exchange *version* messages that contain (i) the client version number, (ii) current block height and (iii) the current time. Both nodes acknowledge the *version* messages with a *verack* message to confirm that the connection is established.

**Block synchronization** The *genesis block* is the first block starting the blockchain and is hard coded in each Bitcoin node. Subsequent blocks typically synchronize during operation. If a node has been offline during a substantial amount of time (typically 24 hours), the block synchronization allows a node to catch-up to the current tip of the best blockchain.

The current Bitcoin implementation (version 0.10 upwards) features a *headers-first* block synchronization. This process allows first to retrieve all the headers of the unknown blocks, determine the best chain based on the length and the difficulty of the block headers, and to download then in parallel the actual block content.

Please note that only peers that synchronize with the current blockchain can perform transaction and block validation.

**Block and transaction propagation** Once a peer has synchronized with the current blockchain, the ongoing synchronization continues. Each time a miner discovers a block or a peer creates a new transaction, this information propagates within the P2P network such that peers (i) receive, (ii) validate and (iii) forward this information.

The standard propagation mechanism for peers is visualized in Figure 2.7. Upon arrival of a hash of a new transaction or block, encoded within an *inv* message, a peer can request to retrieve the transaction or block with a (*getheaders* and a) *getdata* message.

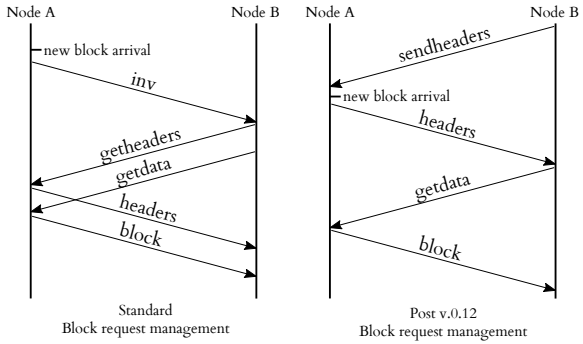



**Figure 2.7:** Propagation mechanism for blocks and transactions.

**Protection from malicious peers** Because there's no trust in any participating P2P node, peers naturally might behave maliciously. A peer, for example, could propagate transactions with wrong signatures, that would require a substantial amount of time to validate on the receiving peers.

To avoid that the P2P network suffers from malicious peers, a node maintains for each peer a score of malicious behavior. Once this score reaches a hard coded threshold, the particular peer is disconnected from, and its corresponding IP address is banned for 24 hours.

**Alternative relay network** The miner's priority is to receive new blocks as fast as possible and to broadcast found blocks as efficiently as possible to the majority of the other miners. The time of block reception is a competitive advantage because mining on an old block could result in a fork, and eventually, one miner will lose in profits.

The Bitcoin P2P overlay network can be considered a general-purpose broadcast mechanism for full bitcoin nodes, SPV clients as well for miners. Due to these constraints, it is not optimized for the sole propagation of blocks, critical to miners. Therefore, alternative relay networks [68] specialized for miners and private peering among miners have emerged.

## 2.5 Conclusion

Despite the lack of new cryptographic primitives, Bitcoin's protocol and smart use of existing cryptographic concepts have its inceptor allowed to create a new and truly revolutionary form of digital, decentralized currency. Besides the ability to serve as a digital currency, Bitcoin moreover has the ability to provide more sophisticated functionalities, such as distributed markets, smart and transferable property or assurance contracts [3].

Because Bitcoin and Proof of Work blockchains in general, however, can be considered a relatively new and unexplored technology, their security, privacy, and performance characteristics are not well understood. The following chapters aim to shed light on the possible problems, as well as appropriate solutions about security and privacy of blockchain participants.

# Chapter 3

# Related work

This chapter surveys related work that is relevant to the scope of this thesis, namely the area of PoW blockchains with a particular focus on security, privacy and network aspects. Bitcoin [125] is without a doubt the most well studied and understood PoW based decentralized blockchain. Bitcoin has, at the time of writing, the biggest market capitalization of over 10 Billion USD, followed by Ethereum with nearly 1 Billion USD. Alternative PoW blockchains, as well as PoW alternatives, however, have been proposed that we cover in the following. The chapter moreover familiarizes the reader with the standard language of adversarial attacks introduced in the previous background section and contrasts related work with our contributions.

Bitcoin [125] was proposed by a pseudonym named Satoshi Nakamoto in the year 2009 and sparked a considerable effort, both from academia and industry to the development and deployment of blockchain technology. While Bitcoin mostly focuses on the ability to provide a decentralized cryptocurrency, alternative PoW blockchains such as Ethereum [62] emerged which allow executing smart contracts described in a Turing-complete programming language run on top of the blockchain. For a comprehensive introduction and overview that exposes Bitcoin and the second generation crypto-currencies, we refer to Bonneau et al. [58].

Due to their financially sensitive nature, the security and privacy of PoW blockchains have received considerable attention from the research community.

## 3.1 Security

In the following, we summarize related work relevant to the security of PoW blockchains. PoW blockchain's security relies on two main pillars, (i) the protection against double-spending; that is the possibility to spend a monetary amount or coin more than once and (ii) selfish mining attacks. In selfish mining, a miner does not follow the honest mining protocol and attempts to increase its relative mining share.

**Double-Spending** Several contributions analyze double-spending attacks in Bitcoin [4, 123] given a chosen adversarial strategy. Finney [4] describe a double-spending attack in Bitcoin where the attacker includes in her generated blocks transactions that transfer some coins between her addresses; these blocks are only released in the network after the attacker double-spends the same coins using fast payments and acquires a given service. Welten et al. [50] compile countermeasures to detect double spending attacks. In [98], Karame et al. investigate double-spending attacks of fast payments (i. e., payments that have not been confirmed in the blockchain) in Bitcoin and show that double-spending of fast payments can be performed in spite of the measures recommended by Bitcoin developers. Barber et al. [51] analyze possible ways to enhance the resilience of Bitcoin against security threats.

**Selfish Mining** In a seminal contribution, Eyal and Sirer [84] show that a selfish miner can increase its relative mining revenue by not directly publishing his blocks. Similarly, Courtois and Bahack [69] study subversive mining strategies. In [115], Nayak et al. combine selfish mining and eclipse attacks, without considering optimal adversarial strategies. This thesis shares similarities with Sapirshtein et al. [124]. Here, the authors devise optimal adversarial strategies for selfish mining in Bitcoin. Unlike [124], our work (cf. Chapter 5) however captures optimal adversarial selfish mining strategies for PoW-powered blockchain and takes into account network delays and eclipse attacks. We additionally obtain optimal double-spending strategies—where we also take into account the mining costs of the adversary, the number of required block confirmations, and the double-spending value to properly account for expenses of the attack.

Croman *et al.* [71] discuss the scalability limitations of Bitcoin, but

do not quantify the security implications of smaller block intervals or bigger blocks on the security of the system. Several works [71,85] analyse the security of Bitcoin's protocol but only consider the synchronous network model. Our analysis and findings do not rely on any assumption on synchrony in the network and capture realistic network delays witnessed in existing deployments.

**Alternative Blockchain Structures** Bitcoin's blockchain [125] forms as a chain, i. e., each block (except the genesis block) has a parent block, actually creating a chain. The longest chain with the highest difficulty is considered the main chain. GHOST [129] is an alternative to the longest chain rule for establishing consensus in PoW based blockchains and aims to alleviate adverse impacts of stale blocks.

### 3.1.1 Network

Most security-related studies assume that nodes directly receive the information disseminated in the Bitcoin network. In [75], Wattenhofer et al. connect to a subset of the Bitcoin network and measure the propagation delay of blocks. Miller et al. [110] try to discover Bitcoin's topology by exploiting, for instance, a 2-minute request timeout for transactions.

**Eclipse Attacks** Open decentralized PoW blockchain's security relies on the assumption that most participating P2P nodes receive all transaction and block information nearly simultaneously - i. e., there is a tight synchronization.

Eclipse attacks aim to partition the P2P network into 2 or more clusters, such that the synchronization is no longer possible. Heilman's et al. were the first to show eclipse attacks on Bitcoin [92]. Here, the authors showed that by monopolizing the connections of nodes in the system (an expensive strategy), an adversary could perform selfish mining and abuse Bitcoin's consensus protocol. Our work shares similarities with Heilman's et al. [92], we, however, show that a resource-constrained attacker can achieve eclipse attacks and considerable damage in the network using only a handful of connections. Because we lower the costs of eclipse attacks substantially, our results suggest that the double-spending and selfish mining attacks outlined in [69,84,92,98] can be even more aggravated.

## 3.2 Privacy

A number of contributions focus on the privacy aspects of PoW blockchains. Although peers transfer funds among pseudonyms (i. e., addresses), payments are linkable such that the origin of a payment is traceable at any time. Transaction prices and time of payment are also published and stored persistently in the blockchain.

Miers et al. introduced in [109] ZeroCoin, a cryptographic extension to Bitcoin that augments the protocol to prevent the tracing of coin expenditure. In [46], Androulaki and Karame proposed an extension of ZeroCoin to hide the transaction values and address balances in the system. In [14], Elias investigates the legal aspects of privacy in Bitcoin. Reid and Harrigan [121] analyze the flow of Bitcoin transactions in a small part of Bitcoin log. In [1], Androulaki et al. evaluate user privacy in Bitcoin and show that Bitcoin leaks considerable information about the profiles of users. More specifically, the authors show that in a typical university setting, even when users adopt privacy measures, profiles of almost 40% of the users are recoverable. In [122], Ron and Shamir analyze the behavior of Bitcoin users. In [119], Ober et al. studied the time-evolution properties of Bitcoin by analyzing its transaction graph. In [78], Meiklejohn et al. identify big players in the Bitcoin system by leveraging Heuristics I and II adapted from [1, 99]. The authors perform transactions with big vendors such as Mt. Gox. and use our heuristics to identify clusters of addresses of such merchants. In [104], the authors investigated the possibility of linking addresses of the same user together by utilizing the Bitcoin peers network address information (IPs).

**Privacy in Bloom Filters:** Bloom filters have been proposed as a privacy preserving way for lightweight Bitcoin clients to retrieve transactions relevant to their wallets. In chapter 7, we show that the current Bloom filter implementations considerably leak information about the addresses a lightweight client possesses.

As far as we are aware, Mullin et al. [112] were the first to propose an estimate of the false positive rate of Bloom filters. In [66], Christensen et al. propose a novel technique for computing the false positive rate, which results in tighter estimates when compared to [112].

In [54], Bianchi et al. quantify the privacy properties of Bloom filters; their analysis, however, does not address the privacy provisions when the adversary has access to multiple Bloom filters originating from

the same entity. In [118], Nojima et al., propose a cryptographically secure privacy-preserving Bloom-filtering protocol based on blind signatures; this proposal, however, incurs additional computational load on SPV nodes. Private Information Retrievals (e.g., [101]) (PIRs) can also be used as an alternative to Bloom filters; PIR schemes, however, result in the non-negligible computational overhead on the nodes. For example, Bloom filters are used, e.g., to enhance the privacy of document search [52], design privacy-preserving record linkage [126], or obfuscate the schemes of database tables from curious administrators [138].

**Location Privacy** Blumberg [57] et al. provide a non-technical discussion of location privacy, its issues, and implications. Gruteser and Grunwald [90] initiate major research in the area of the anonymization approaches to location privacy. Further, Narayanan et al. [114] investigate location privacy from a theoretical standpoint and present a variety of cryptographic protocols motivated by and optimized for practical constraints while focusing on proximity testing. Shokri et al. [127] propose a formal framework for quantifying location privacy in the case where users expose their position sporadically. They model various location-privacy-preserving mechanisms, such as location obfuscation and fake location injections. This work is orthogonal to ours because, in our setting, the consumers are not willingly revealing their locations. Voulodimos et al. [136] address the issue of privacy protection in context-aware services through the use of entropy as a means of measuring the capability of locating a user's whereabouts and identifying personal selections. Narayanan [113] and Shmatikov propose statistical de-anonymization attacks against high-dimensional micro-data. We do not rely on their methods because we are not aiming to de-anonymize the consumers. De Montjoye et al. [141] show that consumers can be uniquely identified within credit card records with only a few spatiotemporal triples containing location, time and price value. Contrary to their work, we focus on the price values, and we localize instead of identifying consumers.

**Payment systems** The privacy implications of public transaction prices have been widely ignored. One prominent example is Bitcoin [58, 125], where transactions are exchanged between peers by means of pseudonyms. The actual transaction prices are archived and publicly available. The literature features many different methods for analyzing the privacy implications of Bitcoin, e.g., using appropriate heuris-

tics [1], tainting [87], or other techniques [78, 122]. Reid and Harrigan [121] analyze the flow of Bitcoin transactions in a small part of the Bitcoin blockchain, and show that external information like publicly-announced addresses, can be used to link identities and organizations to some transactions. In [109] the authors propose Zerocoin, a cryptographic extension to Bitcoin that augments the protocol to allow for fully anonymous currency transactions using a distributed ECash scheme. To the best of our knowledge only two contributions [46, 53] have aimed to hide the transaction prices in Bitcoin.

**Price rigidity** Herrmann and Moeser [93] perform a quantitative analysis of price variability and conclude that prices are often rigid for several weeks. Pricing strategies for same brands, however, vary significantly among retailers. Their observations match the studies of the Big Mac index [19] (the Economist), the Starbucks coffee index [28] (the Wall Street Journal) and the Ikea Billy Bookshelf index [12] (Bloomberg). The former studies show that prices of identical products from a single brand vary across locations. Dutta et al. [77] find that retail prices respond promptly to direct cost changes as well as upstream manufacturers' costs. Hosken and Reiffen [94] find that each product has a price mode—a price that the product stays at most of the time. Note that Hosken's non-public dataset contains nearly as many price observations as our Numbeo dataset.

**Further Related Work** In [49], Babaioff et al. address the lack of incentives for Bitcoin users to include recently announced transactions in a block. Furthermore, in [13], Syed et al. propose a user-friendly technique for managing Bitcoin wallets. In [111], Moore and Christin study the economic risks that investors face due to Bitcoin exchanges. Clark et al. [67] propose the use of the Bitcoin PoW to construct verifiable commitment schemes.

## 3.3 Proof-of-Work Alternatives

Many PoW alternatives have been proposed. In *Proof of Stake* (PoS) [2], the voting power of peers is based on the amount of "stake" they own in the respective blockchain system. *Proof of Burn* (PoB) is a proposal to replace PoW by burning transaction outputs, such that they can no longer be spent. Existing PoB-based blockchains, however, rely on PoW

to create blocks and therefore ultimately rely on PoW for coin creation. *Proof of Capacity* (PoC), aims to use the available hard-disk space to replace PoW. Bitcoin-NG [83] performs leader election of PoW—allowing the leader to sign micro-blocks until a new leader is elected. The literature features many additional proposals [96,102,107,137] that rely on traditional Byzantine fault tolerant consensus protocols in the hope to increase the consensus efficiency and achieve high transactional throughput. Recent studies propose to combine the use of PoW with BFT protocols to realize highly-performant open consensus protocols (Byzcoin [102]).

# Part II

# On Bitcoin's decentralization

# Chapter 4

# Decentralisation

## 4.1 Introduction

Bitcoin has already witnessed a wider adoption and attention than any other digital currency proposed to date. One of the main reasons for such a broad adoption of Bitcoin has been a promise of a low-cost, anonymous, and decentralized currency that is inherently independent of governments and of any centralized authority [5].

In this chapter, we analyze the (de-)centralized nature of Bitcoin and show that—contrary to widespread belief—Bitcoin is not a truly decentralized system as it is deployed and implemented today.

Namely, in Bitcoin, the users "vote" with their computing power to prevent double-spending (i.e., by *power-voting*) which effectively limits the power of individual users and makes Sybil attacks difficult. Given the huge computing power that is harnessed in the Bitcoin system (currently around 1'600'000 Tera hashes per second), users believe that it is unlikely for any entity to acquire such power alone, and control the network. However, even a quick look at the distribution of computing power in Bitcoin reveals that the power of dedicated "miners" far exceeds the power that individual users dedicate to mining, allowing few parties to effectively control the currency; currently the top-three (centrally managed) mining pools control more than 50% of the computing power in Bitcoin. Indeed, while mining and block generation in Bitcoin was originally designed to be decentralized, these processes are currently largely centralized.

On the other hand, other Bitcoin operations, like protocol updates and incident resolution are not designed to be decentralized, and are controlled by a small number of administrators whose influence does not depend on the computing power that they control but is rather derived from their function within the system. Bitcoin users do not have any direct influence over the appointment of the administrators—which is somewhat ironic since some of the Bitcoin users opt for Bitcoin in the hope of avoiding centralized control that is typically exercised over national currencies.

Furthermore, we note that Bitcoin introduces a level of transparency in terms of coin mining and spending since the transaction logs in Bitcoin are public and available for inspection by any interested party. However, it is not clear how any potential disputes would be resolved in Bitcoin since this would then require appropriate regulatory frameworks—a move which clearly goes against the very nature of Bitcoin.

We further observe that the existence of public logs in Bitcoin can have some negative effects on this currency which extend beyond known privacy concerns [1]. Bitcoin users can e.g., decide not to accept coins that appear to have originated from a particular address (i.e., that were mined by the owner of that address); since the use of any coin (or its fraction) can be traced back to its origin, this decision by the users will practically deflate the value of these coins, since other users become reluctant on accepting these coins as payments. We call this effect *coin tainting* and postulate that it can have a negative effect on the use of Bitcoin as a currency.

Finally, we explore possible solutions and recommendations to enhance the decentralization in Bitcoin. We hope that our findings solicit further research in this area.

The remainder of this chapter is organized as follows. In Section 4.2, we discuss the limits of decentralization in Bitcoin. In Section 4.3, we explore possible ways to enhance the decentralization of Bitcoin. We conclude in Section 4.4.

## 4.2 Centralized Processes in Bitcoin

The original design of Bitcoin (cf. Chapter 2) aims at fully decentralizing the transaction generation and confirmation processes. In fact, decisions in Bitcoin have to be approved by the majority of the computing power in the network (which is assumed to be honest) in order

to be effective in practice.

In what follows, we demonstrate that critical processes in the Bitcoin ecosystem are controlled by a small set of Bitcoin entities, whose influence does not depend on the computing power that they control but is rather derived from their function within the system.

### 4.2.1 Bitcoin Services

Bitcoin has lead to the emergence of several centralized services that monopolize a considerable share of the Bitcoin market.

**Mining Pools:** Bitcoin resists double-spending attacks (i.e., signing-over the same coin to two different users) through the reliance on a distributed PoW-based service [125]. The underlying intuition is that as long as the majority of the computing power in the network is "honest", then the security of Bitcoin transactions can be guaranteed. This implicitly assumes that the computing power is shared among all users in the network, and as such the security of Bitcoin can be ensured as long as the majority of Bitcoin users are honest.

Since mining in Bitcoin is rewarded with BTC generation, this process has become very competitive; an "arms race" has emerged using the various hashing technologies that have been specifically designed to mine BTCs [5]: high-end Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs) and then Application-Specific Integrated Circuits (ASICs).

To guarantee regular payouts of miners, their computing power is often "pooled" into a central "mining pool" that coordinates the mining activities of the participants. Here, the mining pool administrator outsources the search inputs for the PoW problem (e.g., the version, difficulty, last block hash) and asks them to find a solution for the specific outsourced problem. In these systems, all participating miners receive a number of BTCs every time a PoW is found; this payment is proportional to the computing power invested in finding that PoW.

This business model has lead to the emergence of a number of such mining pools. Figure 4.1 depicts the distribution of computing power in the Bitcoin network from April to July 2013. Our findings show that more than 75% computing power in Bitcoin is controlled by 6 major centralized mining pools. In June 2016 the situation remains similar, 5 major mining pools control more than 75% of the computing power. *If these pools were to collude in order to acquire more than 50% of*

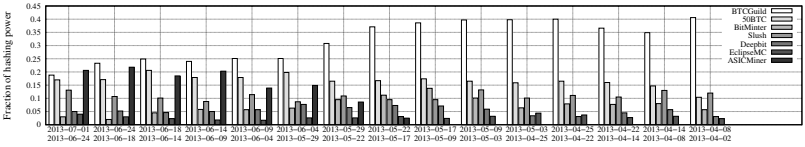



**Figure 4.1:** Distribution of computing power in Bitcoin between April, 2nd 2013 and the 1st of July 2013. More than 55% of the computing power in the network was controlled by BTCGuild and 50BTC, until ASICMiner achieved over 20% of hashing power.

*computing power share in the network, they can effectively control the confirmation of all transactions occurring in the system.* This includes preventing transactions from being executed, approving a specific set of transactions and double-spending transactions [5].

**Bitcoin Web Wallets:** A Bitcoin client installation takes more than 70 GB of disk space and requires several hours in order to download and index the current blockchain. Therefore, users started relying on centralised services that host the main Bitcoin functionalities, called web wallets. A web wallet is an online wallet, hosted on a remote server and accessible through a website. It is instantly functional, does not occupy hard disk space, is accessible anywhere and is consequently more convenient than a local Bitcoin client.

Web wallets empower their operators with full unilateral powers of the BTCs of users. For instance, a malicious web wallet operator could potentially *(i)* steal or forward stolen Bitcoins, *(ii)* trade with non-used funds, and *(iii)* profile users. For example, in April 2013, a theft of 923 BTCs occurred in the mining pool OzCoin. A subset of the stolen BTCs were transferred to a web wallet hosted by StrongCoin. Although StrongCoin claims that it supports user privacy, and does not have access to the user funds, StrongCoin intercepted the allegedly stolen BTCs and transferred them back to OzCoin. *This decision was taken by few entities in the network without acquiring consensus from the majority of users in the network.*

**SPV Mode:** Simplified payment verification (SPV) consists of a modified Bitcoin node which does not verify transactions or blocks in the Bitcoin network. SPV relies on a trusted node which forwards transactions and blocks to all connecting nodes. SPV clients have been

introduced because the default Bitcoin client consumes a significant amount of power, which most smartphones for example cannot afford. *This clearly comes at costs of decentralization, since a critical security component of the network is outsourced to a single node.*

### 4.2.2 Protocol Maintenance and Modifications

The Bitcoin core developers have the authority to make all the necessary modifications to the Bitcoin protocol; according to the Bitcoin Github repository, all "radical" decisions require consensus amongst all the developers. For example, in the Bitcoin client version 0.8.2 the developers unilaterally introduced a fee policy change and decided to lower the default fee for low-priority transactions from 0.0005 BTC to 0.0001 BTC. Clearly, this empowers the Bitcoin developers to regulate and control the entire Bitcoin economy. In what follows, we illustrate this by means of further recent examples in Bitcoin.

**Blockchain Forks:** During the normal Bitcoin operation, miners work on extending the longest blockchain in the network. If miners do not share the same view in the network (e.g., due to network partitioning), they might work on different blockchains, thus resulting in "forks" in the blockchain.

Blockchain forks are detrimental for the operation of the Bitcoin system. Since one blockchain will eventually prevail (the longest), all transactions that were included in all other chains will be invalidated by the miners in the system. Note that Bitcoin does not embed any mechanism to alleviate this problem; instead, if the forks persist for a considerable period of time, Bitcoin developers have to take a decision in favoring one chain on the expense of another (e.g., by sending alert messages and hard-coding the preferred chain in the client code).

As an example, we proceed to describe a chain fork in March 2013, that solicited intervention from the Bitcoin developers. Recall that the Bitcoin client version 0.7 stored the blockchain in the BerkleyDB database, while client version 0.8 switched to the more efficient LevelDB database. Version 0.7 sets the threshold for the maximum number of "locks" per BerkleyDB update to 10,000; this limit, on the other hand, is set to 40,000 in version 0.8. This discrepancy caused a serious fork in the blockchain starting from block 225,430 on 11th of March 2013. This block contained around 1,700 transactions, affected more than 5,000 block index entries, and therefore exceeded the required number

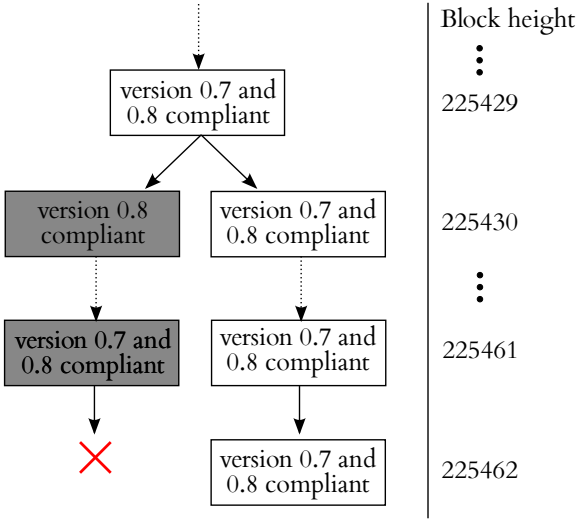



**Figure 4.2:** Sketch of the blockchain fork that occurred in Bitcoin on 11.03.2013.

of locks for version 0.7 (each block index entry requires around 2 locks in BerkleyDB). As a consequence, this resulted in a blockchain fork; all version 0.7 miners rejected block 225,430 and continued working on a blockchain that does not include it, while miners with version 0.8 accepted that block and added it to their blockchain. This process is depicted in Figure 4.2. The chain adopted by version 0.8 clients was supported by the majority of the computing power in the network (it exceeded the chain adopted by 0.7 clients by 13 blocks at block 225,451). Nevertheless, the Bitcoin developers decided, 90 minutes after the fork occurred, to force the smallest chain to be the "genuine" one, and convinced the owner of the biggest mining pool, Eleuthria from BTC Guild, to support this decision.

*This decision comes at odds with the claim that Bitcoin is a decentralized system and that the majority of the computing power regulates Bitcoin. Less than 10 entities [6] took a decision to out-vote the majority of the computing power in the network; this decision has affected the transactions of thousands of users.* We also point out that such influential entities also have the power to make more "radical" decisions (e.g., accepting/rejecting transactions in the system, etc.).

**Alert Mechanism:** Alert messages have been introduced in the Bitcoin client after version 0.3.10. These messages serve to alert the Bitcoin users in case of critical incidents. For example, if a severe vulnerability is found in the Bitcoin client, Bitcoin developers can issue an alert message that will be displayed to the user. In Section 4.2.3, we show how such alert messages can be used to deflate the value of coins in the system.

Since alert messages are cryptographically signed, they can only be sent by people that possess the appropriate cryptographic key. Currently, this key is shared among the Bitcoin developers. *This gives these entities privileged powers to reach out to users and urge them to adopt a given Bitcoin release.* We point out that the current alert payload format supports a RESERVED string which is not currently being used. It is straightforward to see that this field can be abused in the future to send additional control commands (i.e., Botnet-like commands) to be executed by Bitcoin users [7].

**Bitcoin Improvement Proposals:** In order to affect the Bitcoin development process, Bitcoin users are requested to file a Bitcoin Improvement Proposal (BIP) [5] that is assessed by the Bitcoin developers. The developers then unilaterally make a decision whether such a proposal will be supported by the future Bitcoin releases.

*This limits the impact that users have, irrespective of their computing power, to affect the development of the official Bitcoin client. Recent events reveal that contributing within the Bitcoin community is not a trivial process [8].*

### 4.2.3 Coin Tainting

Given that Bitcoin transactions basically consist of a chain of digital signatures, the expenditure of individual coins can be publicly tracked. This enables any entity to "taint" coins that belong to a specific (set of) addresses and monitor their expenditure across the network. The literature features a number of proposals that cluster Bitcoin addresses [1], and gather behavioral information about these addresses [119, 122].

Coin tainting is currently used to achieve a degree of accountability in the Bitcoin network; if an address misbehaves, then Bitcoin users can decide to stop interacting with the address (i.e., not accepting its coins), thus deflating the value of all the coins pertaining to that address. For instance, following a theft of 43,000 BTCs from the Bitcoin trading

platform Bitcoinica, the Bitcoin service MtGox traced the stolen BTC and locked accounts that were receiving the tainted coins [7].

These incidents show that powerful entities in Bitcoin can—rightfully or not—deflate the value of BTCs owned by specific addresses. If these entities were to cooperate with the handful of developers that have privileged rights in the system, then all Bitcoin users can be warned not to accept BTCs that pertain from a given address (e.g., using alert messages). Even worse, developers can hard-code a list of banned Bitcoin addresses within the official Bitcoin client releases, thus blocking all interactions with a given Bitcoin address without the consent of users.

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

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

We conducted two experiments to analyze the impact of coin tainting on the Bitcoin network. In the first experiment, we measured the number of unspent transaction outputs (UTXO) that are affected when tainting a coinbase. Recall that a coinbase is the first transaction in a block, and attributes the block mining reward to a particular address. We randomly sampled 100 coinbases from the last 20,000 blocks of the blockchain which by the time of the experiment had the highest block-height at 247,054. Our results show that a single coinbase tainting affects a large number of transaction outputs; on average, tainting a single coinbase affects 857,239 UTXO (with a standard deviation of 767,528), accounting for 12.9% of all possible UTXO. The resulting distribution of the number of affected UTXO within our sampled coinbases is depicted in Figure 4.3.

In a second experiment, we analyzed the effect of tainting addresses belonging to a single entity in Bitcoin. Given the absence of data to identify these addresses, we relied on two heuristics adapted from [1] in order to cluster addresses across Bitcoin entities.

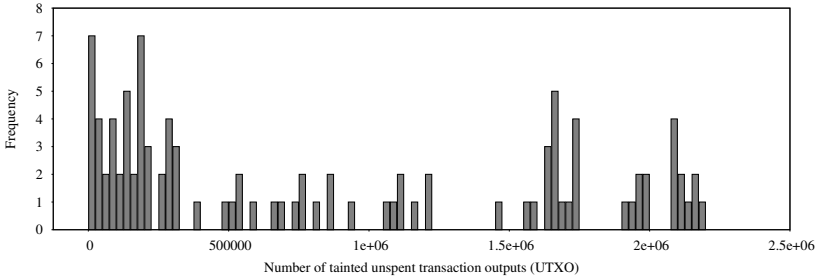



**Figure 4.3:** Tainting 100 random coinbases between block-height 227,054 and 247,054 and counting the number of affected UTXO.

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

**Heuristic II—"Shadow" Addresses:** Bitcoin generates a new address, the "shadow" address [5], onto which each sender can collect back the "change". This mechanism suggests a distinguisher for shadow addresses. In the case when a Bitcoin transaction has $n$ output addresses, such that only one address is a new address (i.e., an address that has never appeared in the transactions log before), and all other addresses correspond to an old address (an address that has appeared previously in the transactions log), we can safely assume that the newly appearing address constitutes a shadow address for the input address. Note that the official Bitcoin client started to support transactions with multiple recipients since December 16, 2010.

We point out that these heuristics combined can only give a lower estimate on the number of addresses per entity. Given the aforementioned heuristics, most entities that we discovered (85%) were equipped with only one address, while there are 0.122% (i.e., almost 9845) of these

entities with more than 40 addresses. On average, our results show that tainting the addresses of a single entity in Bitcoin would result in devaluating an average of 1.42 BTCs (with a standard deviation of 127.58 BTC). The actual distribution of Bitcoin addresses per Bitcoin entity can be seen in Figure 4.4.

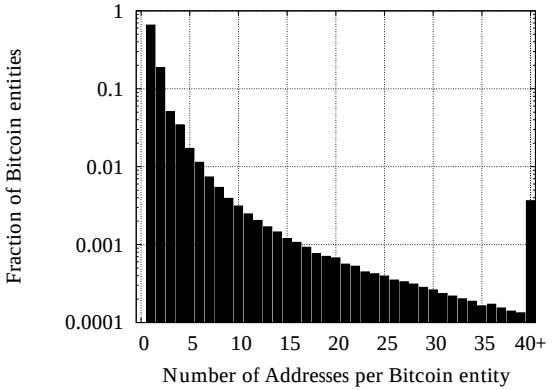



**Figure 4.4:** Number of addresses per Bitcoin entity derived from Heuristics I and II [1].

As an exemplary application of our analysis, we identified two Bitcoin addresses belonging to Torservers.net (using information available from `blockchain.info`). Given the knowledge of these two addresses, we were able to identify a total of 47 addresses, belonging to the operator of Torservers, with a total balance of 498.20 BTC. If an external entity, e.g. a governmental institution, would like to stop the Torservers from receiving Bitcoin donations, it could taint all UTXO of the affected Bitcoin addresses. An exemplary application of the impact coin tainting was demonstrated by the shutdown of Silk Road—one of the most well known underground online black market—by the FBI in October 2013. Note that Silk Road was only accessible through the Tor network; the FBI seized over 27,000 BTCs stored within one or more Bitcoin addresses.

## 4.3 Enhancing the Decentralization in Bitcoin

In what follows, we explore possible avenues that could enhance the decentralization in Bitcoin.

**Mining Pools:** While most existing mining pool protocols assume the existence of a logically centralized operator that orchestrates the block generation process, a number of fully decentralized mining pools, such as P2Pool, are emerging. Such pools share the benefits of centralized pools since all the participating users get regular payouts that reflect their contribution towards generating a block. However, these pools do not require the existence of any centralized coordinator and operate in a completely decentralized fashion. Currently, P2Pool only holds a marginal share of the computing power in the network; Bitcoin users can only hope that such decentralized pools can be transformed into profitable businesses in the near future in order to attract most miners.

**One "Vote" per Client:** Although few mining pools clearly control the computing power in the network (cf. Figure 4.1), we argue that there is still hope for reducing the impact of mining pools on the Bitcoin system. Given that Bitcoin relies on the notion of "controlled supply" which effectively limits the total number of generated BTCs (i.e., the amount of BTC that are generated for each block is halved every four years), the ultimate dominance of mining pools is expected to decrease with time, since their profits would depend less and less on self-awarded BTCs, and more on transaction fees. This, in turn, also increases the contribution of individual users to the Bitcoin economy. Indeed, mining pools operators would then have less incentives to accept client versions that are adopted by the minority of the clients (cf. Section 4.2.2). Users would then contribute more to the decision making process in Bitcoin by deciding to adopt a client version that suits their preferences. Recall that since the Bitcoin source code is open-source, there are already a considerable number of different Bitcoin implementations [5].

**Transparent Decision Making:** In some settings, it is inevitable that various client versions/implementations require constant maintenance and development by a group of leading developers. Here, problems arise in those situations where the developers have to take action

in order to resolve possible conflicts that may have raised. Indeed, this process needs to be completely transparent and should be tightly regulated in order not to abuse the trust of users, and to minimize such unilateral interventions in the system. For example, in order to prevent the (ab-)use of alerts in Bitcoin, these alerts should be accompanied with provable and undeniable justifications. Based on these proofs, users can then decide whether to accept or not such warnings. For instance, double-spending alerts can include the double-spending transactions [98]; this provides irrefutable proof that a given address is double-spending.

Finally, careful planning and testing of version releases is required so as to ensure backward compatibility with previous versions.

**Marketplaces, SPV Clients and Web-Wallets:** Few centralized services, such as marketplaces and web-wallets, have emerged in order to facilitate the use of the decentralized Bitcoin protocol. Indeed, this shows the lack of foresight when deploying Bitcoin since several of the decentralized aspects of Bitcoin are not user-friendly; users find it cumbersome to maintain local wallets and do not wish to download the entire blockchain upon client installation. On the one hand, a truly decentralized Bitcoin is unlikely to be adopted by a vast majority of users since decentralization often comes at odds with user-friendly performance and service. On the other hand, decentralization has been one of the main attractions for users that decided to adopt Bitcoin.

Currently, there is only a handful of services that populate Bitcoin and enable a user-friendly trade and storage of BTCs. For instance, three Bitcoin marketplaces, MtGox, Bitstamp and BTC China handle more than 80% of the USD to Bitcoin trading. We can only hope that more similar services will form to reduce the market share of the current oligopoly.

## 4.4 Conclusion

While the original design of Bitcoin aims at a fully decentralized Bitcoin, recent events in Bitcoin are revealing the true limits of decentralisation in this system. A large number of centralized services currently host Bitcoin and control a considerable share in the Bitcoin market. Even worse, Bitcoin developers retain privileged rights in conflict resolution and maintenance of the official client version. These entities

altogether can decide the fate of the entire Bitcoin system, thus bypassing the will, rights, and computing power of the multitude of users that populate the network.

Currently, almost every financial system is controlled by governments and banks; Bitcoin substitutes these powerful entities with other entities such as IT developers and owners of mining pools. While current systems are governed by means of transparent and thoroughly investigated legislations, vital decisions in Bitcoin are taken through the exchange of opinions among developers and mining pool owners on mailing lists. In this sense, Bitcoin finds itself now in unfamiliar territory: on the one hand, the Bitcoin ecosystem is far from being decentralized; on the other hand, the increasing centralization of the system does not abide by any transparent regulations/legislations. This could, in turn, lead to severe consequences on the fate and reputation of the system, e.g., similar to the Libor scandal.

## 4.5 Appendix: Linked addresses from Torservers.net

| Bitcoin addresses of torservers.net |
|---|
| 18cJBa96SVZLUDvVVpnZZTSwrBDtqgGDa7 |
| 194vsFKoTYP27TZ7PCpeGuKwwmkYPyr6Xg |
| 1DDtChhbKQoknSravfXKgYz51BumYumoVD |
| 1CsBZRX2CpCg2Bk1tDyCCMYoS8WgeSijCA |
| 16r7gfMecotoNbstCc5a2363K1TGqv3ftb |
| 1DGVoJo5Pkoju3VpvNkhRHQnHWksWUR2Sp |
| 1CnX3fCSvcHXgGxW3L56B62dZAdk6nUc1V |
| 1Pyq1VHtehcntRCwwGLZo2yJtKKkZgQ25b |
| 187x5i3ZEL2JeoZ3K2H2VLqdRs1wxqLyqz |
| 1DcBFQFHau8dRe7ixgTnpFh3gg1jun5hWx |
| 15JSnneuvWnmkbqYbLMJfUfrTreQ2eqhLW |
| 152vFYgrdwD8GmuicL1CFerJCrDC2TZWAn |
| 169wQB5S4cdRnCEtE37tX6z4Zj48kRAF71 |
| 1Bgmq5HtFh1oxT2gNRC6Nnevzzxfc3KDCW |
| 1L9QXkyRDx6ujHKr2jzg9FRCAL41RpSJj7 |
| 1ME6f9qhd43XWmzfRkyjQDePpPrs2m4Rkk |
| 1LjCv7jhT3ZWoZnJkQQFxkEqbi7XRixzmV |
| 19hJ1Xkp7bvw2R8j6m8WWgFVSU8Dn1waSH |
| 128HG1QrxWQF7pTkWRqdq5nxRjJRqHJTk3 |
| 1Dh8bQPRTQHyQA5eVKrTeVHbt3VkdzV3Lh |
| 18C31HZGHKCgfDbtzWMXiGK9Adm8N6Uyb2 |
| 1F5MW8CLmkndWnKxf46y4Vp4rupLfvcmC8 |
| 1DwzQHCo9ZZqwyrrzfq8cRB1wGv5YtuGJo |
| 1JMTprwECUtNpqqiB6BwVdpXtkAGyyWNi |
| 1NWKBnDdNb8q14HcyMrmFRt2yfsfaK2MTu |
| 1FtFvuFQzVRNQFmxUeuVbUYvHwyG5Xjwfw |
| 1MCS25nK9mzCmvBYU8ec3sgTbVjAHurKsW |
| 1FQzPR4VYEkV7rLD6Mxipwkm3Sw8Go5v7J |
| 1BruqPe9dkwe2aUNt8WCGMrjxBqAsWSsKJ |
| 19y5GzcpMyfYLpYCtKQ9Po9UjYJGoGG28c |
| 16wDBv7WS7yUJrkDNJMZvPp8KP9VaqdFQ4 |
| 1HSj5qVovCidtzy8Xjt7yvSjjVC1xJRptS |
| 1DiLgDg5nvrrVTfY3CSCfUpNrnfWdt3QeK |
| 1LUZep9TkrJWkNy9gdL8tjmgg3nRUpP3Hx |
| 12XZpkQvrnFQaErhQKBhpdSV9E7hu9pYru |
| 1Krbyg3UANFwr2V6Bnv5Kkzx7SaShFN9kr |
| 1CEpZRCeexVzEKcfQB2jY3qHEJnUQg1zqF |
| 115swHzEr1Tf58hkmg7aP2WSzErg54Tq3K |
| 1CvTorfEs9bwS2wWj5soScCWBf2AAxMLCJ |
| 1GBWxKRtEk7ShQMNsEnFwVyXHbDcLcbfMd |
| 1FTJxLssVmBjDeDtStQi3LqcdQ53wYjE29 |
| 1FXDfiyxfn7Miz2927j5mjmoRcfNuGkT2Y |
| 17cjiuEHntiWt9fLn37ZUWnAqVCumM6ZEU |
| 12Y6AuS1ZQVuS5eMpWxHnhW32em5M9g9JM |
| 17boStnVsuhRno3TdBUNk6JWP3X964qL4H |
| **1KLhvgi9exzvea7fr1CNdHroDC28SkkvEL** |
| **1N5sJeuxCCtG8mXxhgHD3H7W5FWTHK3H21** |

**Table 4.1:** Linked addresses from Torservers.net. The two addresses in bold at the bottom are publicly advertised addresses from Torservers.net.

# Part III

# PoW Blockchain Security and Performance

# Chapter 5

# Security, Performance and Scalability of PoW Blockchain

## 5.1 Introduction

Since its inception in 2009, Bitcoin's blockchain has fueled innovation and a number of novel applications, such as smart contracts, have been designed to take advantage of the blockchain. Bitcoin has been forked a number of times in order to fine-tune the consensus (i.e., the block generation time and the hash function), and the network parameters (e.g., the size of blocks and the information propagation protocol) and to increase the blockchain's efficiency. For instance, Litecoin and Dogecoin—Bitcoin's most prominent forks—reduce the block generation time from 10 to 2.5 and 1 minute. In parallel to these efforts, alternative decentralized blockchain-based networks (such as Ethereum) emerged with the ambition to optimize the consensus and network parameters and to ease the deployment of decentralized applications on top of the blockchain.

Although a number of consensus protocols (PBFT [64], Proof of Stake [2], Proof of Elapsed Time [97]) have been proposed, most existing blockchains leverage the computationally expensive Proof of Work (PoW) consensus mechanism—which currently accounts for more than

90% of the total market capitalization of existing digital currencies. While the security provisions of Bitcoin have been thoroughly analyzed [84,98,123,129], the security guarantees of variant PoW blockchains have not received much attention in the literature. Recent studies hint that the performance of PoW based blockchains cannot be enhanced without impacting their security. However, the relationship between performance and security provisions of PoW blockchains has so far not been studied in much detail.

In this chapter, we address this problem and provide a novel quantitative framework to analyze the security and performance implications of various consensus and network parameters of PoW blockchains. Leveraging our framework, we capture the security implications of existing PoW instantiations (e.g., Bitcoin, Ethereum, Litecoin, and Dogecoin) as well as other possible instantiations subject to different consensus and network parameters.

Our framework (cf. Figure 5.1) consists of two key elements: *(i)* a blockchain instance and *(ii)* a blockchain security model. The blockchain instance refers to a PoW blockchain (e.g., Litecoin, Ethereum) that is instantiated with a given set of consensus/network parameters, such as network delays, block generation times, block sizes, information propagation mechanisms, etc. To realistically capture any blockchain instance, we design an open source simulator [48] that mimics the blockchain consensus and network layer by implementing advertisement-based information propagation, unsolicited block pushes, the relay network, the *sendheader* propagation mechanism, among others. The main output of the blockchain instance/simulator is the stale (orphan) block

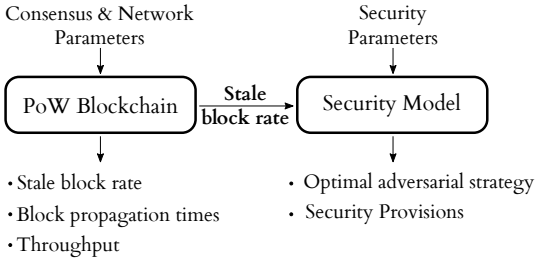



**Figure 5.1:** Sketch of our quantitative framework to analyze the security and performance implications of various consensus and network parameters of PoW blockchains.

rate, which is fed as input into our security model. On the other hand, our security model is based on Markov Decision Processes (MDP) for double-spending and selfish mining and allows us to reason about optimal adversarial strategies while taking into account the adversarial mining power, the impact of eclipse attacks, block rewards, and real world network and consensus parameters—effectively captured by the stale block rate.

Given the current discussions in the Bitcoin community about a suitable maximum block size that ensures the scalability and growth in the system [35], our work provides a way to holistically compare the security and performance of PoW blockchains when subject to different parameters—-including the block size. For instance, we find that increasing the block size from the current Bitcoin transaction load (average 0.5MB) to up to 4 MB, does not significantly affect the selfish mining and double-spending resilience of the blockchain—provided that the block propagation mechanism ensures a low stale block rate. We summarize our findings as follows.

**Summary of findings**

- We show that selfish mining is not always a rational strategy. To this end, we quantify the double-spending resilience of PoW blockchains when subject to rational adversaries and objectively compare the security of different PoW blockchains with respect to the required number of transaction confirmations. By doing so, we provide merchants with the knowledge to decide on the required number of confirmations for a given transaction value to ensure security against double-spending.

- Our results show that, due to the smaller block rewards and the higher stale block rate of Ethereum[1] compared to Bitcoin (from 0.41% to 6.8% due to the faster confirmation time), Ethereum (block interval between 10 and 20 seconds) needs at least 37 confirmations to match Bitcoin's security (block interval of 10 minutes on average) with 6 block confirmations against an adversary with 30% of the total mining power. Similarly, Litecoin would require 28, and Dogecoin 47 block confirmations respectively to match the security of Bitcoin.

[1] We show that, contrary to common beliefs, Ethereum does not apply GHOST's principle to include the contributions of "uncles" in the main chain and therefore currently resembles Bitcoin.

- We show that the higher the block reward of a blockchain (in $) the more resilient it is against double-spending.
- Finally, we analyze the impact of changing the block size and/or the block interval on selfish mining and double-spending. Our results surprisingly show that setting the block size to an average 1 MB, and decreasing the block interval time to 1 minute do not considerably penalize security. Our results therefore suggest that PoW blockchains can attain an effective throughput above 60 transactions per second (tps) (implying that the 7 tps limit of Bitcoin can be substantially increased) without compromising the security of the system.

The remainder of the chapter is organized as follows. In Section 5.2, we overview the basic concepts behind PoW blockchains. In Section 5.3, we introduce our MDP model to quantitatively analyze the security of PoW blockchains. In Section 5.4, we present our simulator and evaluate the security and performance of a number of variant PoW-based blockchain instances. We conclude the chapter in Section 5.5.

## 5.2 Background

In this section, we briefly recap the operations of the consensus layer and the network layer of existing PoW blockchains.

### 5.2.1 Consensus Layer

The Proof of Work (PoW) consensus mechanism is the widest deployed consensus mechanism in existing blockchains. PoW was introduced by Bitcoin [125] and assumes that each peer votes with his "computing power" by solving Proof of Work instances and constructing the appropriate blocks. Bitcoin, for example, employs a hash-based PoW which entails finding a nonce value, such that when hashed with additional block parameters (e.g., a Merkle hash, the previous block hash), the value of the hash has to be smaller than the current target value. When such a nonce is found, the miner creates the block and forwards it on the network layer (cf. Section 5.2.2) to its peers. Other peers in the network can verify the PoW by computing the hash of the block and checking whether it satisfies the condition to be smaller than the current target value.

**Block interval:** The block interval defines the latency at which content is written to the blockchain. The smaller the block interval is, the faster a transaction is confirmed and the higher is the probability of stale blocks. The block interval adjustment directly relates to the difficulty change of the underlying PoW mechanism. A lower difficulty results in a larger number of blocks in the network, while a higher difficulty results in less blocks within the same timeframe.

It is therefore crucial to analyze whether changing the difficulty affects the adversarial capabilities in attacking the longest chain—which is the main pillar of security of most PoW-based blockchains. This also implies the adjustment of the required number of confirmations that a merchant should wait in order to safely accept transactions (and avoid double-spending attacks) (cf. Section 5.3).

### PoW security

PoW's security relies on the principle that no entity should gather more than 50% of the processing power because such an entity can effectively control the system by sustaining the longest chain. We now briefly outline known attacks on existing PoW-based blockchains.

First, an adversary can attempt to double-spend by using the same coin(s) to issue two (or more) transactions—thus effectively spending more coins than he possesses. Recent studies have shown that accepting transactions without requiring blockchain confirmations is insecure [98]. The more confirmations a transaction obtains, the less likely this transaction will be reversed in the future.

Second, miners might attempt to perform *selfish mining* [84] attacks in order to increase their relative mining share in the blockchain, by selectively withholding mined blocks and only gradually publishing them [84, 124]. Recent studies show that, as a result of these attacks, a selfish miner equipped with originally 33% mining power can effectively earn 50% of the mining power.

Double-spending attacks and selfish mining can be alleviated if all nodes in the blockchain system are tightly synchronised. Note that, in addition to network latency, synchronization delays can be aggravated due to *eclipse attacks* [89,92] (cf. Chapter 6) where an adversary creates a logical partition in the network, i.e., provides contradicting block and transaction information to different blockchain network nodes.

### 5.2.2 Network Layer

On the network layer, we identify two main parameters that are of particular importance for PoW-based blockchains, namely: the *block size*, and the *information propagation mechanism*.

#### Block size

The maximum block size indirectly defines the maximum number of transactions carried within a block. This size therefore controls the throughput attained by the system. Large blocks incur slower propagation speeds, which in turn increases the stale block rate (and weaken the security of the blockchain as stated earlier).

#### Information propagation mechanism

The block request management system dictates how information is delivered to peers in the network. Eventually, since all peers are expected to receive all blocks, a broadcast protocol is required. The choice of the underlying broadcast protocol clearly impacts the robustness and scalability of the network (cf. Section 5.4). In what follows, we briefly describe well-known network layer implementations of existing PoW-based blockchains.

**Advertisement-based information dissemination:** Most PoW blockchains propagate messages with the help of an advertisement-based request management system. If node $\mathcal{A}$ receives information about a new object (e.g., a transaction or a block) from another node, $\mathcal{A}$ will advertise this object to its other connections (e.g. node $\mathcal{B}$) by sending them an *inv* message (the hash and type of the advertised object). Only if node $\mathcal{B}$ has not previously received the advertised object, $\mathcal{B}$ will request the object from $\mathcal{A}$ with a *getdata* request. Node $\mathcal{A}$ will subsequently respond with a Bitcoin object, e.g., the contents of a transaction or a block.

**Send headers:** Peers can alternatively issue a *sendheaders* message in order to directly receive block *headers* in the future from their peers—skipping the use of *inv* messages. This reduces the latency and bandwidth overhead of block message propagation and is adopted by Bitcoin since version 0.12. We proposed the *sendheaders* mechanism in order to counter eclipse attacks (cf. Chapter 6).

**Unsolicited block push:** This mechanism enables miners to broadcast their generated blocks without advertisement (i.e., since they mined the block). Note that this push system is recommended[2], but not implemented in Bitcoin.

**Relay networks:** Relay networks [68] primarily enhance synchronization of miners that share a common pool of transactions. Transactions are typically only referenced in relayed blocks with a transaction ID (2 bytes per transaction instead of an average of 250 bytes per transaction). As a consequence, the resulting block size is smaller than the regular block (cf. Bitcoin Relay Network [68]).

**Hybrid Push/Advertisement Systems:** A number of systems, such as Ethereum, combine the use of push and advertisement dissemination. Here, a block is directly pushed to a threshold number of peers (e.g., Ethereum directly pushes blocks to $\sqrt{n}$ peers, where $n$ is the total number of neighbors connected to the peer). Concurrently, the sender advertises the block hash to all of its neighbors.

### 5.2.3 Stale blocks

Stale blocks refer to blocks that are not included in the longest chain, e.g., due to concurrency, conflicts. Stale blocks are detrimental to the blockchain's security and performance because they trigger chain forks—an inconsistent state which slows down the growth of the main chain and results in significant performance and security implications. On the one hand, stale blocks increase the advantage of the adversary in the network (e.g., double-spending). On the other hand, stale blocks result in additional bandwidth overhead and are typically not awarded mining rewards (except in Ethereum).

In an experiment that we conducted, we measure the stale block rate in the Bitcoin (block generation time = 10 minutes, average block size = 534.8KB), Litecoin (block generation time = 2.5 minutes, average block size = 6.11KB) and Dogecoin (block generation time = 1 minute, average block size = 8KB) network. All three blockchains rely on a PoW-based blockchain (with different generation times) and the same information propagation system (with different block sizes).

We crawled the available nodes in Litecoin and Dogecoin [55] in February 2016 and found about 800 and 600 IP addresses respectively.

[2] `https://bitcoin.org/en/developer-reference#data-messages`

| | Bitcoin | Litecoin | Dogecoin | Ethereum |
|---|---|---|---|---|
| Block interval | 10 min | 2.5 min | 1 min | 10-20 seconds |
| Public nodes | 6000 | 800 | 600 | 4000 [80] |
| Mining pools | 16 | 12 | 12 | 13 |
| $t_{MBP}$ | 8.7 s [71] | 1.02 s | 0.85 s | 0.5 - 0.75 s [81] |
| $r_s$ | 0.41% | 0.273% | 0.619% | 6.8% |
| $s_B$ | 534.8KB | 6.11KB | 8KB | 1.5KB |

**Table 5.1:** Comparison of different Bitcoin forks, Ethereum and the impact of parameter choices on the network propagation times. Stale block rate ($r_s$) and average block size ($s_B$) were measured over the last 10000 blocks. $t_{MBP}$ stands for median block propagation time.

We then measured the block propagation times by registering the times at which we receive the block advertisements from a particular block from all our connections in the respective network [75]. We operated one node for Litecoin and Dogecoin, which we connected to 340 and 200 peers, respectively. Once one of these peers advertises block information in form of either *(i)* a new hash of a block (*inv* message) or *(ii)* a block header (*headers* message), we registered the time this block information appeared. Every subsequent reception of a particular piece of block information then provides information about the propagation of the block.

Our results (cf. Table 5.1) suggest that the stale block rate indeed largely depends on the block interval and the block sizes. For instance, unlike Dogecoin and Litecoin, Bitcoin features larger block sizes due to a higher transaction load (of up to 1MB which results in a higher stale block rate (0.41% vs. 0.273%)—although the block interval of Bitcoin is 4 times longer than that of Litecoin. Moreover, the stale block rate differences between Litecoin and Dogecoin are mainly due to the difference in the block interval (2.5 minutes vs. 1 minute), since their average block sizes are comparable (6.11KB and 8KB). Given a confirmation time reduction of 60%, the stale block rate increased by 127% from Litecoin to Dogecoin.

Notice that in Ethereum, *uncle blocks* correspond to stale blocks that are referenced in the main chain. The uncle block rate in Ethereum is 6.8%, compared to a stale block rate of 0.41% in Bitcoin. In Section 5.3, we study the impact of the stale block rate on the security of PoW blockchains.

## 5.3 PoW Security Model

In this section, we introduce our blockchain security model that we leverage to quantify the optimal adversarial strategies for double-spending and selfish mining. We then use these strategies as a basis to compare the security provisions of PoW-based blockchains when instantiated with different parameters.

### 5.3.1 Security Model

Our model extends the Markov Decision Process (MDP) of [124] to determine optimal adversarial strategies, and captures:

**Stale block rate** The stale block rate $r_s$ allows us to account for different block sizes, block intervals, network delays, information propagation mechanisms and network configuration (e.g., number of nodes).

**Mining power** $\alpha$ is the fraction of the total mining power of the adversary (the rest is controlled by the honest network).

**Mining costs** The adversarial mining costs $c_m \in [0, \alpha]$ correspond to the expected mining costs of the adversary (i.e., total mining costs such as hardware, electricity, man-power) and are expressed in terms of block rewards. For example, if $c_m = \alpha$, the mining costs of the adversary are equivalent to its mining power times the block reward, i.e., the mining costs are covered exactly by the earned block revenue in honest mining.

**The number of block confirmations** $k$ This corresponds to the number of blocks that need to confirm a transaction, such that a merchant accepts the transaction.

**Propagation ability** The propagation parameter $\gamma$ captures the connectivity of the adversary within the network (i.e., captures the fraction of the network that receives the adversary's blocks in the case when the adversary and the honest miner release their blocks simultaneously in the network).

**The impact of eclipse attacks** Our model accounts for eclipse attacks. Here, we assume that the miners of the honest network are affected by the stale block rate, while the adversary and the

colluding eclipsed victims do not mine stale blocks. This is due to the fact that the adversary can use any mined blocks for an attack and effectively only has a small chance of mining a stale block after adopting the honest chain. Therefore, in practice, the adversary exhibits a significantly lower real stale block rate than the honest network. The honest network features propagation and validation delays—hence it will witness a higher stale block rate. Note that the blocks found by the eclipsed victim can also count towards the private chain of the adversary.

*We contrast this to existing models, such as Sapirshtein et al.'s [124], which only focus on selfish mining and cannot capture different blockchain instances (with various stale block rates and confirmations) and real-world parameters such as network delays.*

To analyze optimal double-spending strategies, we define the double-spending amount $v_d$ that corresponds to the minimum transaction value that makes double-spending more profitable than honest mining. We argue that $v_d$ emerges as a robust metric to quantify security under double-spending attacks. Namely, if the reward of honest mining is larger than that of dishonest behavior, merchants can safely accept a payment transaction of value $v_d$ (since such a value is considered secure, e.g., based on a given confirmation number). If however, adversarial behavior is financially more rewarding, a merchant should be aware of the associated double-spending risks and of the related incentives of miners.

We capture the blockchain model using a single-player decision problem $M := \langle S, A, P, R \rangle$ where all other participants follow the standard protocol, and $S$ corresponds to the state space, $A$ to the action space, $P$ to the stochastic transition matrix, and $R$ to the reward matrix. We instantiate $M$ as a Markov Decision Process (MDP) as outlined in Section 5.3.2 and 5.3.3.

In our model, the following actions are available to the adversary:

**Adopt** The adversary accepts the chain of the honest network, which effectively corresponds to a restart of the attack. This action is appropriate if the adversary deems that the likelihood to win over the honest chain is small.

**Override** The adversary publishes one block more than the honest chain has and consequently overrides conflicting blocks. This

happens when the adversary's secret chain is longer than the currently known public chain (i.e. $l_a > l_h$) and it is optimal for the adversary to publish $l_h + 1$ of his blocks to replace the honest network's chain with his own. If the adversary exploits the mining power of the victim, the adversary might use $b_e$ blocks from the victim for an *override* action.

**Match** The adversary publishes as many blocks as the honest chain has, and triggers an adoption race between the two chains instead of *overriding* the honest chain.

**Wait** The adversary continues mining on its hidden chain until a block is found.

**Exit** This action is only relevant when studying double-spending as it corresponds to a successful double-spending with $k$ confirmations and is only feasible if $l_a > l_h$ and $l_a > k$.

The state space $S$ is defined as a four-tuple of the form $(l_a, l_h, b_e, \text{fork})$, where $l_a$ and $l_h$ represent the length of the adversarial and honest chain respectively, $b_e$ the blocks mined by the eclipsed victim, and fork can take three values, irrelevant, relevant and active:

**relevant** The label relevant signifies that *(i)* the last block has been found by the honest network, and *(ii)* if $l_a \geq l_h$ the *match* action is applicable. A state of the form $(l_a, l_h - 1, b_e, \cdot)$ for instance results in $(l_a, l_h, b_e, \text{relevant})$.

**irrelevant** When the adversary found the last block, the previous block has likely already reached the majority of the nodes in the network. The adversary is therefore not able to perform a *match* action. A state of the form $(l_a - 1, l_h, b_e, \cdot)$ for instance results in $(l_a, l_h, b_e, \text{irrelevant})$.

**active** The state is described with the label *active*, if the adversary performed a *match* action, i.e., the network is currently split and in process of determining the longest chain.

In our model, every state transition (except *exit*) corresponds to the creation of a block. Consequently, a state transition implies a reward for the honest network, the adversary, or the eclipsed victim.

Given the adversarial mining power $\alpha$, the initial state $(0, 0, 0,$ irrelevant) transitions to $(1, 0, 0, \text{irrelevant})$ with probability $\alpha$, i.e., the

adversary found one block. If the honest network finds a non-stale block, the resulting state is $(0, 1, 0, \text{relevant})$. On the other hand, if the honest network's block results in a stale block, the state remains $(0, 0, 0, \text{irrelevant})$ since a stale block does not count towards the longest chain. The last case accounts for the eclipsed victim which finds a block with probability $\omega$, resulting in state $(1, 0, 1, \text{irrelevant})$.

**Selfish Mining vs. Double-spending** In this work, we consider double-spending and selfish mining independently, since selfish mining is not always a rational strategy: the objective of selfish mining is to increase the relative share of the adversarial blocks committed to the main chain, while in double-spending the adversary aims to maximize his absolute revenue.

Namely, as long as the difficulty of a PoW blockchain does not change (e.g. Bitcoin's difficulty changes only once every two weeks), selfish mining yields fewer block rewards than honest mining. In honest mining, the adversary is rewarded for every mined block, while he will lose any previously mined blocks when adopting the main chain in selfish mining. Since the adversary has less mining power than the honest network, he has a high probability of falling behind the main chain, causing him to adopt the main chain when he has no significant chance of catching up—which in turn leads to lost block rewards. For instance, following our optimal selfish mining strategy (cf. Section 5.3.2), an adversary with 30% of the mining power earns 209 block rewards on average in a duration where 1000 blocks are mined by the whole network (as opposed to 300 for honest mining). Similarly, Eyal and Sirer's [84] strategy yields on average 205.80 blocks rewards.

**Eclipse attacks** In an eclipse attack, a fraction $\omega$ of the overall mining power is eclipsed [92, 115] from receiving information from the honest network. Here, a number of eclipse attack variants arise:

**No eclipse attack** This case is captured in our model if $\omega = 0$.

**Isolate the victim** This is captured implicitly in our model. Namely, this corresponds to a decrease of the total mining power and thus an increase of the attacker mining power to $\alpha' = \frac{\alpha}{1-\omega}$.

**Exploit the eclipsed victim** Here, the adversary exploits the victim's mining power $\omega$ and uses it to advance his private chain.

This is the most likely choice of a rational adversary when performing double-spending attacks. In this case, we assume that the victim is fully eclipsed from the network and does not receive/send blocks unless permitted by the adversary [92, 115].

### 5.3.2 Selfish Mining MDP

Our goal is to find the optimal adversarial strategy for selfish mining. Recall that the objective of the adversary in selfish mining is not to optimise the absolute reward, but to increase the share of blocks that are included in the chain accepted by the network. We capture this by optimising the relative revenue $r_{rel}$ defined in Equation 5.1, where $r_{a_i}$ and $r_{h_i}$ are the rewards in step $i$ for the adversary and the honest network, respectively:

$$r_{rel} = \mathbb{E}\left[\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r_{a_i}}{\sum_{i=1}^{n}(r_{a_i} + r_{h_i})}\right] \tag{5.1}$$

Since an adversary aims to increase his relative reward $r_{rel}$ (Equation 5.1) in selfish mining, as opposed to the absolute reward, the single-player decision problem cannot be modelled directly as an MDP, because the reward function is non-linear. In order to transform the problem into a family of MDPs, we apply the technique of Sapirshtein et al.'s [124], which we describe below.

We assume that the value of the objective function (i.e., the optimal relative reward) is $rho$ and define for any $\rho \in [0, 1]$ the transformation function $w_\rho : \mathbb{N}^2 \to \mathbb{R}$ with the adversarial reward $r_a$ and the reward of the honest network $r_h$ in Equation 5.2.

$$w_\rho(r_a, r_h) = (1 - \rho) \cdot r_a - \rho \cdot r_h \tag{5.2}$$

This results in an infinite state MDP $M_\rho = \langle S, A, P, w_\rho(R)\rangle$ for each $\rho$ that has the same action and state space as the original decision problem and the same transition matrix but the reward matrix is transformed using $w_{rho}$. The expected value of such an MDP under policy $\pi$ is then defined by $v_\rho^\pi$ in Equation 5.3, where $r_i(\pi)$ is the reward tuple in step $i$ under policy $\pi$.

$$v_\rho^\pi = \mathbb{E}\left[\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} w_\rho(r_i(\pi))\right] \tag{5.3}$$

The expected value under the optimal policy follows:

$$v_\rho^* = \max_{\pi \in A} \left\{ v_\rho^\pi \right\} \tag{5.4}$$

To optimise $r_{rel}$ we adopt the following propositions [124]:

1. If $v_\rho^* = 0$ for some $\rho \in [0, 1]$, then an optimal policy $\pi^*$ in the transformed MDP $M_\rho$ also maximises $r_{rel}$ and $r_{rel} = \rho$.
2. $v_\rho^*$ is monotonically decreasing in $\rho$.

Since standard MDP solvers are not able to solve infinite state MDPs, we restrict the state space of our family of MDPs by only allowing either chain to be of length at most $c$, resulting in a finite state MDP $M_\rho^c$. If either chain reaches length $c$, the adversary is only allowed to perform the *override* or *adopt* action. This gives a lower bound for the optimal value of the infinite state MDP.

Intuitively, we can reason about the correctness of the first proposition as follows for the bounded single-player decision problem. In a recurring finite state MDP, the initial state will be visited again in expectation after some finite number of steps $S$. During that time, the adversary gains an expected reward of $R_a = \mathbb{E}\left[\sum_{s=1}^{S} r_{a_i}\right]$ and the honest network gains a reward of $R_h = \mathbb{E}\left[\sum_{s=1}^{S} r_{h_i}\right]$ in the original (bounded) decision problem. It follows that the expected reward per step in the Markov Chain is $\overline{r_a} = \mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} r_{a_i}\right]$ and $\overline{r_h} = \mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} r_{h_i}\right]$ for the adversary and the honest network, respectively. We can thus simplify the expected relative revenue $r_{rel}$ to:

$$r_{rel} = \mathbb{E}\left[\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r_{a_i}}{\sum_{i=1}^{n}(r_{a_i} + r_{h_i})}\right] \tag{5.5}$$

$$= \mathbb{E}\left[\lim_{n\to\infty} \frac{n \cdot \overline{r_a}}{n \cdot (\overline{r_a} + \overline{r_h})}\right] \tag{5.6}$$

$$= \mathbb{E}\left[\lim_{n\to\infty} \frac{\overline{r_a}}{(\overline{r_a} + \overline{r_h})}\right] \tag{5.7}$$

$$= \mathbb{E}\left[\frac{\overline{r_a}}{\overline{r_a} + \overline{r_h}}\right] \tag{5.8}$$

$$= \frac{\overline{r_a}}{\overline{r_a} + \overline{r_h}} \tag{5.9}$$

$$\tag{5.10}$$

Additionally, we develop $v_\rho^*$ as follows:

$$
\begin{aligned}
v_\rho^* &= v_\rho^{\pi^*} && (5.11)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} w_\rho(r_i(\pi^*))\right] && (5.12)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}((1-\rho)\cdot r_{a_i} - \rho\cdot r_{h_i})\right] && (5.13)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\left((1-\rho)\cdot\sum_{i=1}^{n} r_{a_i} - \rho\cdot\sum_{i=1}^{n} r_{h_i}\right)\right] && (5.14)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \frac{1}{n}\left(n\cdot(1-\rho)\cdot\overline{r_a} - n\cdot\rho\cdot\overline{r_h}\right)\right] && (5.15)\\
&= \mathbb{E}\left[\varliminf_{n\to\infty} \left((1-\rho)\cdot\overline{r_a} - \rho\cdot\overline{r_h}\right)\right] && (5.16)\\
&= \mathbb{E}\left[(1-\rho)\cdot\overline{r_a} - \rho\cdot\overline{r_h}\right] && (5.17)\\
&= (1-\rho)\cdot\overline{r_a} - \rho\cdot\overline{r_h} && (5.18)
\end{aligned}
$$

And thus, for the case where $\rho = r_{rel} = \frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}}$:

$$
\begin{aligned}
v_\rho^* &= (1-\rho)\cdot\overline{r_a} - \rho\cdot\overline{r_h} && (5.19)\\
&= (1-\frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}})\cdot\overline{r_a} - \frac{\overline{r_a}}{\overline{r_a}+\overline{r_h}}\cdot\overline{r_h} && (5.20)\\
&= \frac{\overline{r_a}\cdot(\overline{r_a}+\overline{r_h}) - \overline{r_a}^2}{\overline{r_a}+\overline{r_h}} - \frac{\overline{r_a}\cdot\overline{r_h}}{\overline{r_a}+\overline{r_h}} && (5.21)\\
&= \frac{\overline{r_a}^2 + \overline{r_a}\cdot\overline{r_h} - \overline{r_a}^2 - \overline{r_a}\cdot\overline{r_h}}{\overline{r_a}+\overline{r_h}} && (5.22)\\
&= 0 && (5.23)
\end{aligned}
$$

The reasoning for the second proposition is straightforward. For any given policy $\pi$, it holds for $\rho > \rho'$ that $w_\rho(r_a, r_h) \leq w_{\rho'}(r_a, r_h)$ for every transition with rewards $r_a$ and $r_h$ for the adversary and the honest network, respectively. It follows directly that $v_\rho^\pi \leq v_{\rho'}^\pi$ for every policy $\pi$ and thus $v_\rho^* \leq v_{\rho'}^*$, i.e., $v_\rho^*$ is monotonically decreasing in $\rho$.

We use binary search on our restricted family of MDPs for $\rho \in [0, 1]$ in order to find the $\rho$ for which the expected value in the instantiated

```
function OPTIMAL STRATEGY(c, ε)
    low ← 0
    high ← 1
    repeat
        ρ ← (low + high)/2
        (π, v*_ρ) ← MDP_SOLVER(M^c_ρ)
        if v*_ρ > 0 then
            low ← ρ
        else
            high ← ρ
        end if
    until high − low < ε
    return π, ρ
end function
```

**Figure 5.2:** Binary search algorithm for the family of MDPs.

MDP is zero and which therefore maximizes the reward in the original single-player decision problem [124]. Since $v^*_\rho$ is monotonically decreasing, this can be done efficiently as described in Figure 5.2.

As far as we are aware, this is the first selfish mining model that *(i)* captures various parameters such as block propagation times, block size, block generation interval, and *(ii)* known network vulnerabilities such as eclipse attacks. Note that we do not consider mining costs in the selfish mining MDP since the objective here is to increase the relative mining share (and not the monetary reward).

### Optimal Strategies for Selfish Mining

In order to solve the MDP's, we apply an MDP solver for finite state space MDPs [95], and use a cutoff value of 30 blocks.

We first analyze the impact of the stale block rate on selfish mining. In Figure 5.3, we compare selfish mining under a stale block rate of 1% and 10%, and we observe that the higher the adversarial mining power, the bigger the relative revenue of a selfish miner grows (up to a maximum difference of 0.074). For comparison purposes, we plot an upper bound $\frac{\alpha}{1-\alpha}$ of the adversarial relative revenue from selfish mining which corresponds to the case where the adversary's advantage is maximized by utilizing one block to override one block generated by the honest network (as reported by Sapirshtein et al. [124]). As we

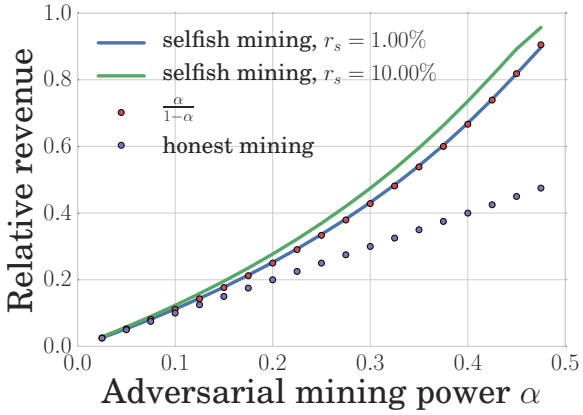



**Figure 5.3:** Selfish mining for $r_s$ of 1%, 10%.

observe, this upper bound is exceeded when taking into account network delays and parameters that we capture via the stale block rate.

For an adversary with a mining power of $\alpha = 0.1$ and $\alpha = 0.3$ respectively, we observe in Figure 5.4 that there is a non-linear relationship between the stale block rate and the relative mining revenue of selfish mining.

We moreover study the impact of eclipse attacks on selfish mining in Figure 5.5. Here, we only consider the case where the adversary *(i)* exploits the victims mining power $\omega$, and *(ii)* uses all the victim's blocks to advance his private chain. We therefore only determine the optimal adversarial choices given these restraints. We observe that the higher $\omega$, the stronger his selfish mining capabilities become. We note that for some values of $\omega$ (e.g., $\omega = 0.3$, $\alpha = 0.38$), it is more rewarding for the adversary not to include some of the victims block in his private chain. This is because the victim's block rewards count towards the reward of the honest network, and therefore reduce the relative block share of the adversary.

### 5.3.3 Double-Spending MDP

Unlike selfish mining where the optimal strategy is not always financially rewarding compared to honest mining (cf. Section 5.3), we proceed in what follows to study optimal double-spending strategies, where

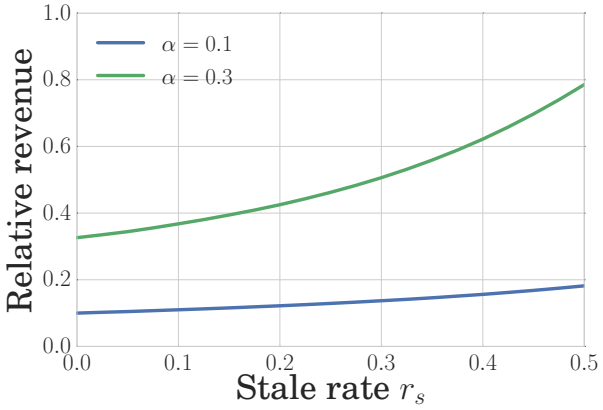



**Figure 5.4:** Selfish mining for $\alpha = 0.1$ and 0.3.

we assume a *rational* adversary that is interested in maximizing his benefits (measured in financial gains) in the network.

We implicitly require that each time the adversary starts a double-spending attack (e.g. after an *adopt* action), he publishes a transaction $T_l$ in the network, and mines on including a conflicting transaction $T_d$ in his private chain. We assume that the operational costs of "losing" a double-spending attempt are small, since the adversary effectively receives a good or service in exchange for transaction $T_l$.

In addition to the states described for the selfish mining, the double-spending MDP features the *exit* state (cf. Table 5.2). This state can only be reached provided that the adversarial chain is at least one block ahead the honest chain ($l_a > l_h$), after $k$ confirmations ($l_a > k$), given an honest network with mining power $1 - \alpha$. Before reaching the *exit* state, the adversary adopts an optimal strategy to maximize its reward, given the state and action space described in Section 5.3. After reaching the exit state, transitions back to the exit state model rewards of honest mining. Note that since we assume that the adversary is rational, an optimal strategy might advise against performing double-spending attacks (i.e. the adversary will never reach the *exit* state)—depending on the value of the attempted attack. In the *exit* state, the adversary earns a block reward of $l_a - b_e + v_d$, $\lfloor (l_h + 1)\frac{l_a - b_e}{l_a} \rfloor - c_m$ block rewards after an override with eclipse attack (because the adversary's reward needs to discount the $b_e$ victim's blocks and $\lfloor (l_h)\frac{l_a - b_e}{l_a} \rfloor - c_m$ block

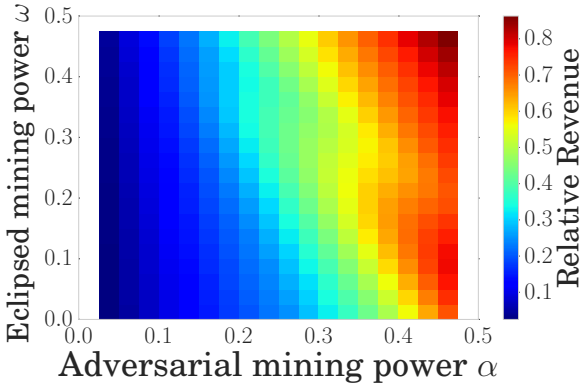



**Figure 5.5:** Selfish mining with eclipse attacks.

rewards if the adversary's chain wins the race after a *match* action. For every state transition we discount the mining costs $-c_m$.

The adversary either abides by the optimal double-spending strategy $\pi$ or performs honest mining, depending on the expected reward. We are therefore interested in the minimal double-spending value $v_d$, such that $v_d$ is strictly larger than the honest mining reward (cf. Equation 5.25).

$$P = (\alpha, \gamma, r_s, k, \omega, c_m) \tag{5.24}$$

$$v_d = \min\{v_d | \exists \pi \in A : R(\pi, P, v_d) > R(\text{honest mining}, P)\} \tag{5.25}$$

The double-spending value $v_d$ can serve as a generic metric to compare the security of various blockchain instantiations. Namely, if $v_d$ of a blockchain instance $A$ is bigger than for blockchain $B$ for given $\alpha$, $\gamma$ and $\omega$, then blockchain $A$ can be considered more resistant against double-spending attacks.

### Optimal Strategies for Double-Spending

In what follows, we analyze the solutions of our aforementioned double-spending MDP given various parameters. To solve for the optimal strategy in our MDP, we rely on the *pymdptoolbox* library[3] and apply

[3] `https://github.com/sawcordwell/pymdptoolbox`

| State × Action | Resulting State | Probability | Reward (in Block reward) |
|---|---|---|---|
| $(l_a, l_h, b_e, \cdot)$, adopt | $(1,0,0,i)$ | $\alpha$ | $(-c_m, l_h)$ |
| | $(1,0,1,i)$ | $\omega$ | $(-c_m, l_h)$ |
| | $(0,1,0,r)$ | $(1-\alpha-\omega)\cdot(1-r_s)$ | $(-c_m, l_h)$ |
| | $(0,0,0,i)$ | $(1-\alpha-\omega)\cdot r_s$ | $(-c_m, l_h)$ |
| $(l_a, l_h, b_e, \cdot)$, override | $\left(l_a - l_h, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, i\right)$ | $\alpha$ | $\left(\lfloor (l_h+1)\frac{l_a - b_e}{l_a} \rfloor - c_m, 0\right)$ |
| | $\left(l_a - l_h, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil + 1, i\right)$ | $\omega$ | $\left(\lfloor (l_h+1)\frac{l_a - b_e}{l_a} \rfloor - c_m, 0\right)$ |
| | $\left(l_a - l_h - 1, 1, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, r\right)$ | $(1-\alpha-\omega)\cdot(1-r_s)$ | $\left(\lfloor (l_h+1)\frac{l_a - b_e}{l_a} \rfloor - c_m, 0\right)$ |
| | $\left(l_a - l_h - 1, 0, b_e - \lceil (l_h+1)\frac{b_e}{l_a} \rceil, i\right)$ | $(1-\alpha-\omega)\cdot r_s$ | $\left(\lfloor (l_h+1)\frac{l_a - b_e}{l_a} \rfloor - c_m, 0\right)$ |
| $(l_a, l_h, b_e, i)$, wait | $(l_a+1, l_h, b_e, i)$ | $\alpha$ | $(-c_m, 0)$ |
| $(l_a, l_h, b_e, r)$, wait | $(l_a+1, l_h, b_e+1, i)$ | $\omega$ | $(-c_m, 0)$ |
| | $(l_a, l_h+1, b_e, r)$ | $(1-\alpha-\omega)\cdot(1-r_s)$ | $(-c_m, 0)$ |
| | $(l_a, l_h, b_e, i)$ | $(1-\alpha-\omega)\cdot r_s$ | $(-c_m, 0)$ |
| $(l_a, l_h, b_e, a)$, wait | $(l_a+1, l_h, b_e, a)$ | $\alpha$ | $(-c_m, 0)$ |
| $(l_a, l_h, b_e, r)$, match | $(l_a+1, l_h, b_e+1, a)$ | $\omega$ | $(-c_m, 0)$ |
| | $(l_a - l_h, 1, b_e - \lceil (l_h)\frac{b_e}{l_a} \rceil, r)$ | $\gamma\cdot(1-\alpha-\omega)\cdot(1-r_s)$ | $\left(\lfloor (l_h)\frac{l_a - b_e}{l_a} \rfloor - c_m, 0\right)$ |
| | $(l_a, l_h+1, b_e, r)$ | $(1-\gamma)\cdot(1-\alpha-\omega)\cdot(1-r_s)$ | $(-c_m, 0)$ |
| | $(l_a, l_h, b_e, a)$ | $(1-\alpha-\omega)\cdot r_s$ | $(-c_m, 0)$ |
| $(l_a, l_h, b_e, \cdot)$, exit | exit | 1 | $(l_a - b_e + v_d, 0)$ |

**Table 5.2:** State transition and reward matrices for optimal selfish mining and double-spending strategies in PoW blockchains. $\alpha$ is the mining power of the attacker, $\omega$ is the mining power of the eclipsed node, $b_e$ is the number of blocks in the attacker chain that were mined by the eclipsed node, $\gamma$ is the fraction of nodes that an attacker can reach faster than the honest network, $r_s$ is the stale block rate and $v_d$ is the value of the double-spend. The actions override and match are feasible only when $l_a > l_h$ or $l_a \geq l_h$, respectively. We discount the mining costs $c_m \in [0, \alpha]$ in the state transition reward only for double-spending. The fork label (last element of the state) is denoted by $i$, $r$ and $a$ for *irrelevant*, *relevant* and *active* respectively. For a reward tuple $(a, b)$, $a$ corresponds to the adversary's costs, while $b$ represents the reward for the honest network for selfish mining.

the *PolicyIteration* algorithm [95] with a discount value of 0.999. This methodology allows us to assess whether the number of transaction confirmations $k$ are sufficient to ensure security in the presence of a rational adversary, with respect to the considered transaction value. That is, if the adversary has a higher expected financial gain in double-spending than honest mining, then the transaction cannot be considered safe given $k$ confirmations, and the merchant should wait additional confirmations.

In order to decide whether the adversary should choose to follow the optimal double-spending policy or honest mining (cf. Equation 5.25), and to determine the minimum $v_d$, we instantiate the double-spending MDP with a high double-spending value ($> 10^9$ block rewards), such

| $l_a$ \ $l_h$ | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| **0** | w** | *a* | *** | *** | *** | *** | *** | *** | *** |
| **1** | w** | ww* | ww* | *a* | *** | *** | *** | *** | *** |
| **2** | w** | ww* | ww* | ww* | ww* | *a* | *** | *** | *** |
| **3** | w** | ww* | ww* | ww* | ww* | ww* | *a* | *** | *** |
| **4** | w** | ww* | ww* | ww* | ww* | ww* | ww* | *a* | *** |
| **5** | w** | ww* | ww* | ww* | ww* | ww* | ww* | ww* | *a* |
| **6** | w** | ww* | ww* | ww* | ww* | ww* | ww* | ww* | ww* |
| **7** | e** | e** | e** | e** | e** | e** | e** | w** | ww* |
| **8** | *** | *** | *** | *** | *** | *** | *** | e** | w** |

**Table 5.3:** Optimal double-spending strategy for $\alpha = 0.3, \gamma = 0, r_s = 0.41\%, c_m = \alpha, \omega = 0$ and $v_d = 19.5$. The rows correspond to the length $l_a$ of the adversary's chain and the columns correspond to the length $l_h$ of the honest network's chain. The three values in each table entry correspond to the fork labels *irrelevant*, *relevant* and *active*, where * marks an unreachable state and *w*, *a* and *e* denote the *wait*, *adopt* and *exit* actions, respectively.

that the *exit* state is reachable in the optimal policy. If the policy contains an *exit* state, the expected gain of following the optimal double-spending strategy is higher than honest mining. Otherwise, honest mining is the preferred strategy. We apply binary search to find the lowest double-spending value (in units of block rewards, within an error margin of 0.1), for $\alpha$, $k$, $r_s$, $\gamma$ and $c_m$.

In Table 5.3, we sketch an example of an optimal strategy for the case where $\alpha = 0.3$ (adversarial mining power), $\gamma = 0$ (propagation parameter), $c_m = \alpha$ (maximum mining costs), $\omega = 0$ (no eclipse attack), where we observe only *wait*, *adopt* and *exit* actions. Because we can only solve finite MDPs, we choose a cutoff value of 20 blocks, i.e., neither the chain of the adversary nor the chain of the honest network can be longer than the cutoff value. In the following paragraphs, we discuss in greater details the impact of $\alpha$, $\gamma$, $c_m$, $r_s$, $k$, $v_d$ and $\omega$ on the optimal double-spending strategy and its implications on the security of transaction confirmations.

Recall that the absorbing state [100] of the Markov chain of our double-spending MDP is the *exit* state. By computing the fundamental matrix [100] of the Markov chain, we calculate the expected number of steps in the Markov chain—before being absorbed by the *exit* state.

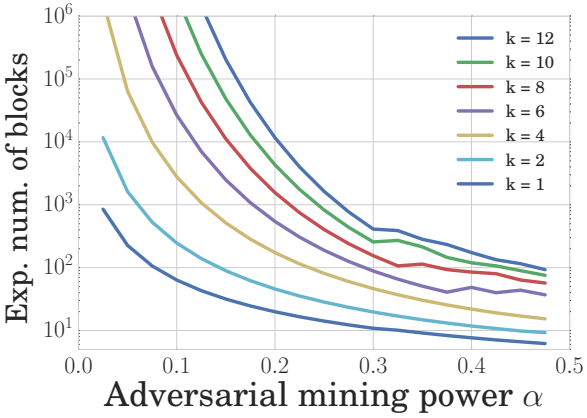



**Figure 5.6:** Expected number of blocks for successful double-spending given $r_s = 0.41\%, \gamma = 0, c_m = \alpha$ and $\omega = 0$.

These steps correspond to the expected number of blocks required for a successful double-spending attack. In Figure 5.6, we evaluate the expected number of blocks with respect to the adversarial mining power and the number of transaction confirmations $k$. We observe that an adversary with a mining power of more than 0.25 is expected to need less than 1000 blocks for a successful double-spending attack (up to $k = 10$ confirmations), which corresponds to a one week attack duration in Bitcoin.

**Impact of the propagation parameter:** Recall that the propagation parameter specifies the connection capability of the adversary. In Figure 5.9, we depict the minimum double-spending transaction value that would result in financial gain when compared to honest mining (cf. Equation 5.25) when $\gamma = 0, 0.5$ and 1 respectively. Recall that a merchant is safe as long as he accepts transactions with a value less than $v_d$ given these parameters.

Clearly, the higher $\gamma$ is, the higher is the transaction value that an adversary is expected to double-spend. For example, if the adversary has $\alpha = 0.3$ of the hashing power in the network, assuming $k = 6$ confirmations, and a mining cost of $c_m = \alpha$, a double-spending strategy is clearly profitable if the double-spending transaction has a value of at least 0.5 block rewards (one block reward was 25 Bitcoin, where one

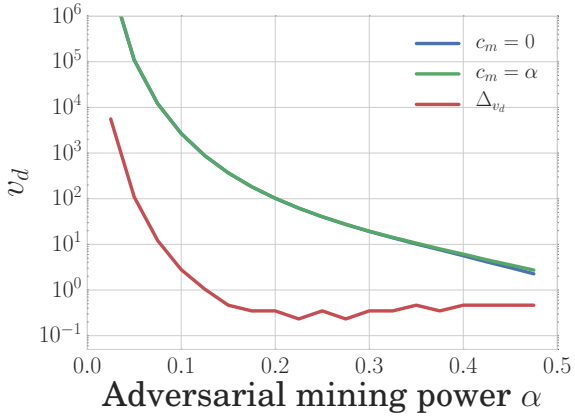



**Figure 5.7:** Impact of the mining cost $c_m$ on the security of double spending ($r_s = 0.41\%$, $\gamma = 0$, $\omega = 0$). $\Delta_{v_d}$ is the difference in costs.

Bitcoin is about 436.7 USD at the time of writing, for $r_s = 0.41\%$) when $\gamma = 1$. When $\gamma = 0.5$, the minimum transaction value increases to 12.9 block rewards.

**Impact of the mining costs:** In Figure 5.7, we analyze the impact of the mining costs on the minimum required double-spending transaction value. Our results show that mining costs have negligible impact on the adversarial strategy.

**Impact of the stale block rate:** We evaluate the impact of the stale block rate for adversaries with a mining power of $\alpha = 0.1$ and $\alpha = 0.3$ in Figure 5.8. We observe that there exists a non-linear relationship between the stale block rate and the double-spending value and that the higher the stale block rate, the worse is the double-spending and selfish mining resistance of a PoW blockchain (cf. Figure 5.8). For instance, for an adversary with mining power $\alpha = 0.3$ and a stale block rate of 10% and 20%, the double-spending value $v_d$ decreases from 9.2 to 6.4 block rewards. Similarly, the relative revenue from selfish mining (cf. Figure 5.4) increases from 0.37 to 0.43.

**Impact of eclipse attacks** We evaluate the impact of eclipse attacks on the adversarial strategy given our MDP. We assume that the ad-

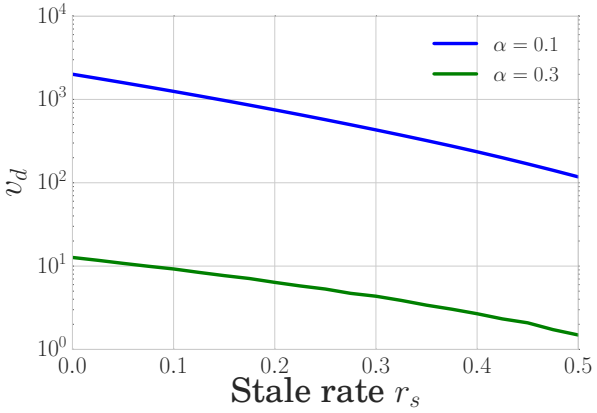


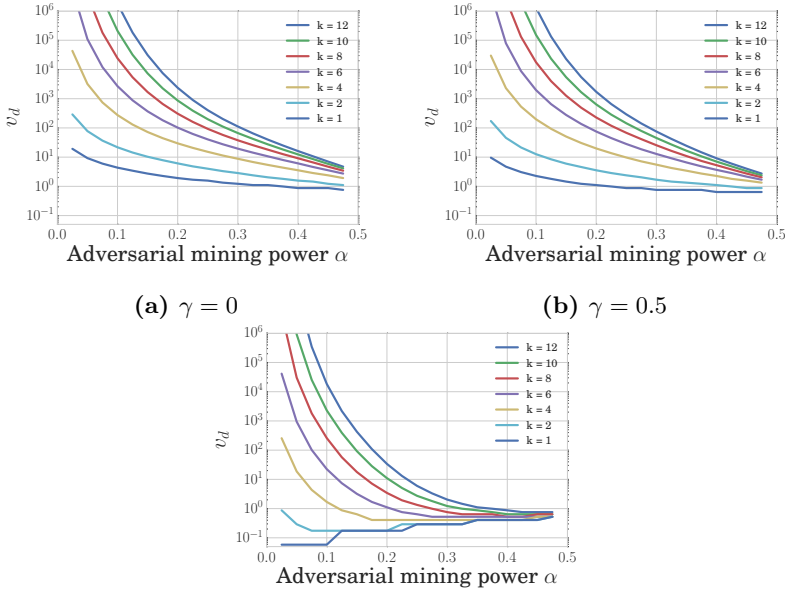

**Figure 5.8:** Impact of stale block rate $r_s$ on the security of double-spending given $\gamma = 0.5$, $\omega = 0$ for $\alpha = 0.1$, $\alpha = 0.3$ and $k = 6$.

versary eclipses a victim with mining power $\omega$ in order to increase its advantage in sustaining his blockchain (cf. Figure 5.10). We observe that an eclipse attack clearly empowers the adversary, since it allows the adversary to effectively increase its overall mining power. For instance, an adversary with $\alpha = 0.1$ can reduce the double-spending value $v_d$ from 880 block rewards to 0.75 block reward if eclipsing a miner with $\omega = 0.025$.

### Bitcoin vs. Ethereum

In order to alleviate the problem that stale blocks decrease PoW's efficiency, a number of proposals, such as Ethereum, suggest to reward miners for stale blocks [62]. Here, although uncle blocks that are included in a block receive a reward, *they do not count towards the total difficulty of a chain*, i.e., Ethereum uses a longest chain rule with added rewards for uncle blocks. This clearly contradicts Ethereum's claim of using a blockchain protocol adapting GHOST [129].

Ethereum has also recently modified its longest chain algorithm to incorporate uniform tie breaking [79]. Notice that such a strategy is meant as a selfish mining countermeasure, but allows a selfish miner to increase its chances of catching up to the honest chain [124]. In Table 5.9, we extend our model to cater for uncle rewards and uniform




**(a)** $\gamma = 0$ **(b)** $\gamma = 0.5$

**(c)** $\gamma = 1$

**Figure 5.9:** Impact of the propagation parameter $\gamma$. We observe that the higher is $\gamma$, the lower is $v_d$ for double-spending to be more profitable than honest mining. $r_s = 0.41\%$ (Bitcoin's stale block rate), $c_m = \alpha$ (maximum mining costs), $\omega = 0$ (no eclipse attack).

tie breaking, and describe the resulting double-spending MDP in order to capture the security of Ethereum against double-spending.

Building on this analysis, we compare in Figure 5.11, the double-spending resilience of Bitcoin ($r_s = 0.41\%$, cf. MDP in Table 5.2) to that of Ethereum ($r_s = 6.8\%$, cf. MDP in Table 5.9), given $\gamma = 0, c_m = 0$ and $\omega = 0$. In order to provide a fair cost comparison, we rely on US dollar based valuation (Bitcoin's block reward is more than 200 times higher than Ethereum's block reward at the time of writing).

We observe that 6 Bitcoin block confirmations are more resilient to double-spending than 6 Ethereum[4] block confirmations. Second, when comparing 12 Ethereum with 6 Bitcoin block confirmations, Ethereum's double-spending resilience is only better than Bitcoin for an adversary

[4]Block generation time between 10 and 20 seconds.

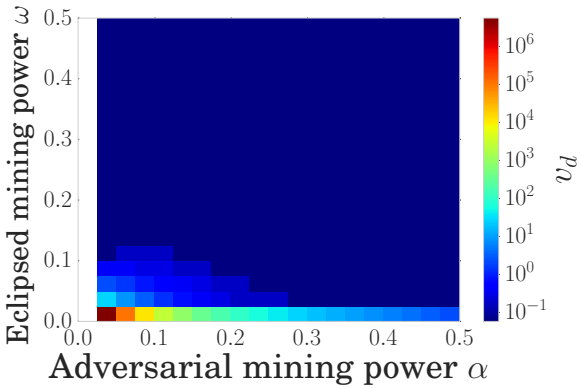



**Figure 5.10:** Full eclipse attack for $r_s = 0.41\%$, $\gamma = 0$ and $c_m = 0$.

with less than 11% of the PoW hashing power. Note that 12 Ethereum blocks are likely to be generated in less than 4 minutes, while 6 Bitcoin blocks last about one hour. Third, we discover that the monetary value of the block reward directly impacts the double-spending security: the higher the block reward of a blockchain (in $) the more resilient it is against double-spending.

In addition to comparing Bitcoin to Ethereum, we compare in Figure 5.12 the two blockchains by setting Bitcoin's stale block rate equal to Ethereum's stale block rate to objectively evaluate their security implications. We observe that, in spite of the reliance on uncle block rewards, and uniform tie breaking, Ethereum's security is weaker than Bitcoin, and conclude that the uniform tie breaking and the uncle reward lower the security of Ethereum's blockchain.

## 5.4 Security vs. Performance of PoW-Blockchains

In this section, we evaluate the performance (and security) of various blockchain instantiations by leveraging our model in Section 5.3.

To this end, we constructed a Bitcoin blockchain simulator in order to evaluate different blockchain instances from a performance perspective. Relying on simulations emerges as the only workable alternative to realistically capture the blockchain performance under different param-

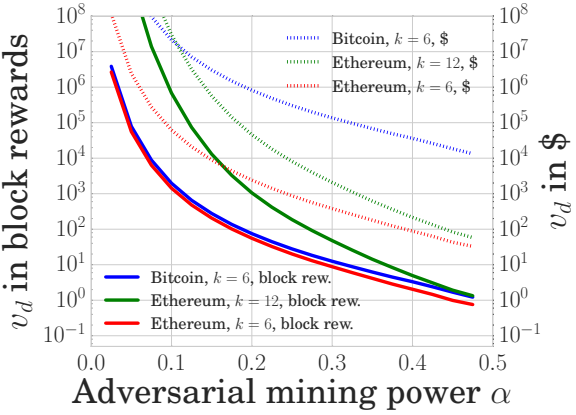



**Figure 5.11:** Double-spending resistance of Ethereum ($k \in \{6, 12\}$) vs. Bitcoin ($k = 6$). USD exchange rate of 2016-04-20.

eters since neither formal modeling, nor the deployment of a thousands of peers (e.g., currently there are 6000 reachable nodes in Bitcoin) would be practical.

By leveraging our simulator, we evaluate different blockchain parameters, such as the block interval, the block size, the propagation mechanisms by measuring the resulting stale block rate, throughput and block propagation times. This also allows us to connect our blockchain simulator to our MDP model in a unified framework. Namely, we feed the stale block rate output by the simulator into our MDP model in order to assess the security (under selfish mining and double-spending) of the resulting blockchain instance.

### 5.4.1 Blockchain Simulator

In Table 5.4, we summarize the parameters captured by our simulator. Here, we simulate the PoW for miners, by attributing a particular mining power to each miner. Based on the block interval distribution (which defines at what time a block is found), a new block is then attributed to a miner. Conforming with the operation of existing PoW-blockchains, a miner mines on the first block he receives, and we assume that forks are inherently resolved by the longest chain rule. Once a fork is resolved, the blocks that do not contribute to the main

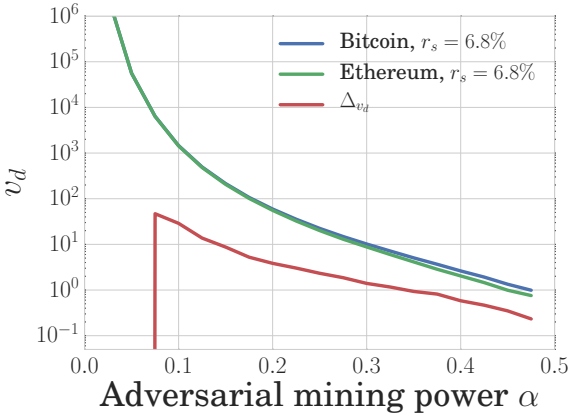



**Figure 5.12:** Direct comparison between Ethereum and Bitcoin with $k = 6$, $r_s = 6.8\%$ and their respective difference $\Delta_{v_d}$.

chain are considered stale blocks. Within our simulations, we do not consider difficulty changes among different blocks; the longest chain is therefore simply defined by the number of its blocks.

When establishing the connections between nodes, we create point-to-point channels between them, which abstracts away any intermediate devices (routes, switches, etc). These channels have two characteristics; the latency and bandwidth. To capture realistic latencies in the network, we adopt the global IP latency statistics from Verizon [135] and assume a Pareto traffic distribution with variance accounting for 20% of the mean latency [47]. On the other hand, to model a realistic bandwidth distribution in the network, we adapted the distribution[5] from `testmy.net` [132].

Our simulator does not model the propagation of transactions, since the focal point of our simulator is to study the impact of the block size, block interval, and the block request management system—all of which can be captured independently of the transaction propagation. Note that transactions are implicitly captured within the block size.

In our simulator, we distinguish between two node types: *(i)* regular nodes, and *(ii)* miners. For regular nodes (up to 6000), we retrieved

[5]Upload bandwidth characteristics: min=0.1Mbps, max=100Mbps, interval=0.1Mbps. Download bandwidth characteristics: min=0.1Mbps, max=500Mbps, interval=0.5Mbps.

| **Consensus parameter** | Description |
|---|---|
| Block interval distribution | Time to find a block |
| Mining power distribution of the miners | PoW power distribution |
| **Network-layer parameter** | Description |
| Block size distribution | Variable transaction load |
| # of reachable network nodes | Open TCP port nodes |
| Geo. distribution of nodes | Worldwide distribution |
| Geo. mining pool distribution | Worldwide distribution |
| # of connections per node | Within network |
| # of connections of the miners | Within network |
| Block request management system | Possible Protocols |
| Standard mechanism (inv/getdata) | Default |
| Unsolicited block push | Miner only push block |
| Relay network | Miner network |
| Sendheaders | Bitcoin v0.12 |

**Table 5.4:** Parameters of the blockchain simulation.

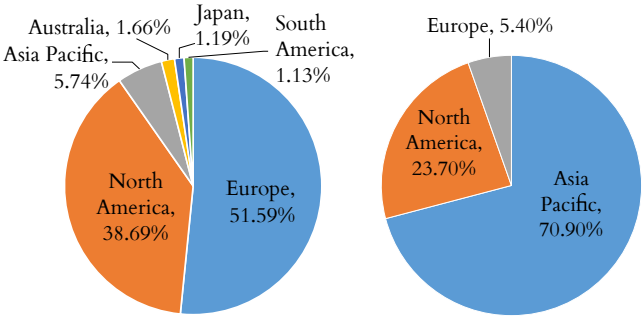
the current geographical node distribution from `bitnodes.21.co` (cf. Figure 5.13a) and adopted this distribution to define the location of our simulated nodes. We also adapted the bandwidth and network latency (according to the geographical location) from Verizon [47, 135] and `testmy.net` [132]. To model miners, we retrieved the mining pool distribution from `blockchain.info`, and accordingly distributed the mining pool's public node to the respective region (cf. Figure 5.13b). Mining pools typically maintain private peering connections among themselves—which we capture in our simulations. Besides direct peering, a number of mining pools nowadays participate in Matt Corallo's relay network [68] that is operated independently of the default Bitcoin P2P overlay network (cf. Section 5.2.2). We also capture the relay network and assume in our simulations that all miners participate in the relay network whenever the relay network option is enabled.

### 5.4.2 Evaluation Results

In what follows, we present the results from our evaluation.

#### Simulator Validation

With the objective to experimentally validate our simulation, we compared Bitcoin, Litecoin, and Dogecoin with their respective simulated




**(a)** Node distribution. **(b)** Miner distribution.

**Figure 5.13:** Geographical distribution of Bitcoin nodes and miners used in our simulator.

counterpart. For each blockchain, we adjusted the parameters of Table 5.4 according to the current parameters featured by existing deployments of the investigated blockchains. For instance, we measured Bitcoin's block size distribution, as well as the block generation rate[6] in the real Bitcoin network between May to November 2015 [98].

In order to measure the stale block rate $r_s$ in the real blockchain networks, we crawled 24,000 Bitcoin blocks[7], 100,000 Litecoin and 240,000 Dogecoin blocks[8]. We moreover adopt the miner mining power distribution for the different blockchains from public block explorers[9]. The number of connection per node in our simulations follows the distribution due to Miller et al. [110].

Our findings (cf. Table 5.5) show that our simulator captures, to a large extent, the performance of existing blockchain deployments. For instance, our results show that the measured and simulated median block propagation times are relatively close. The stale block rates for Litecoin and Dogecoin are particularly close. In the case of Bitcoin, the stale rate falls between the case when all miners use the relay network and unsolicited block push, and the extreme case where the relay network and unsolicited block push is not used by any miner. Note that

[6]The block generation rate distribution follows the shifted geometric distribution with $p = 0.19$ [98].

[7]from `blockchain.info`

[8]from `blockchains.io`

[9]`blockchain.info` and `https://www.litecoinpool.org/pools`

| | **Bitcoin** | **Litecoin** | **Dogecoin** |
|---|---|---|---|
| Block interval | 10 min | 2.5 min | 1 min |
| Measured $t_{MBP}$ | 8.7 s [71] | 1.02 s | 0.98 s |
| Simulated $t_{MBP}$ | 9.42 s | 0.86 s | 0.83 s |
| Measured $r_s$ | 0.41 % | 0.27 % | 0.62 % |
| Simulated $r_s$ | (a)0.14%-(b)1.85% | (b)0.24 % | (b)0.79 % |

**Table 5.5:** Median block propagation time ($t_{MBP}$, in seconds), and $r_s$ in the real networks and the simulation (10000 blocks for each blockchain). (a) assumes that all miners use the relay network and unsolicited block push, while (b) is only given the standard propagation mechanism. We conclude that not all miners in Bitcoin use the relay network and unsolicited block push.

Litecoin and Dogecoin do not have any relay network.

### Impact of the Block Interval

In this section, we study the impact of the block interval on the median block propagation time and the stale block rate in PoW-based blockchains. To this end, we run our simulator for different block interval times ranging from 25 minutes to 0.5 seconds (cf. Table 5.6). Each simulation is run independently for 10000 consecutive blocks, and for each of the four different block request management system combinations: *(Case 1)* the standard block request management, *(Case 2)* the standard block request management enhanced by unsolicited block push from the miners, *(Case 3)* both former components plus the relay network, and *(Case 4)* the send headers mechanism with unsolicited block push and the relay network.

We observe first that for a block interval time of 10 minutes and a standard request management system, our stale block rate is 1.85%, which is comparable to 1.69% as reported by Wattenhofer et al. [75]. Recall that at the time of Wattenhofer's study, the unsolicited block push and relay network were not yet available.

Secondly, we observe that the introduction of the unsolicited block push for miners significantly reduces the stale block rate. This is the case since *(i)* miners are interconnected and profit most from the unsolicited block push, and *(ii)* the propagation method of the first node is crucial to reach the majority of the network rapidly. The addition of the relay network does not seem to affect the stale block rate sig-

nificantly (given the Bitcoin's transaction load) compared to the unsolicited block push, and reduces the propagation time only marginally. For bigger block sizes however (e.g. $>$ 2MB) the relay network indeed provides an advantage over the unsolicited block push (cf. Table 5.7). Moreover, the relay network provides an additional source of block information, in addition to the classical P2P overlay network. Notice that although the impact of the send header mechanisms compared to a fully deployed relay network and unsolicited block push is limited, this mechanism mitigates partial eclipse attacks [89].

To assess the impact of the block interval on the security of PoW blockchains, we feed the resulting stale block rate into our MDP models as shown in Table 5.6. Our results show that, for an adversary equipped with 30% of the total mining power[10], the lower is the consensus time, the higher is the relative revenue from selfish mining and the lower is the double-spending value. We observe that the block propagation mechanism significantly impacts the security of the blockchain, since it directly affects the stale block rate. The standard block propagation mechanism offers less resilience (in terms of double-spending and selfish mining) than the other evaluated block propagation mechanisms. We also note that the double-spending value halves in Table 5.6 for the block propagation mechanism of Case 4 (which results in the lowest stale block rate when compared to the other investigated mechanisms) when reducing the block interval from 25 minutes to 0.5 seconds. Similarly, the relative revenue from selfish mining increases from 0.33 to 0.42.

### Impact of the Block size

We now study the impact of the block size on the performance and security of the blockchain (cf. Table 5.7. To this end, we simulate block sizes ranging from 0.1 MB to up to 8 MB, given a block interval of 10 minutes.

Our results suggest that the block propagation time increases linearly with the block size up to 4 MB; after 8 MB blocks, the block propagation time and stale block rate increases exponentially. Second, we clearly see that a better block propagation mechanism significantly reduces the propagation times and the stale block rate.

[10] Bitcoin's resilience to malicious miners is based on the assumption that the adversary cannot harvest more than 30% of the total mining power [84, 89].

| | Case 1 | | | | Case 2 | | | |
|---|---|---|---|---|---|---|---|---|
| **Block interval** | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ |
| 25 minutes | 35.73 | 1.72 % | 12.47 | 0.34 | 25.66 | 0.16 % | 12.86 | 0.33 |
| 10 minutes | 14.7 | 1.51 % | 12.52 | 0.34 | 10.65 | 0.13 % | 12.88 | 0.33 |
| 2.5 minutes | 4.18 | 1.82 % | 12.45 | 0.34 | 2.91 | 0.16 % | 12.86 | 0.33 |
| 1 minute | 2.08 | 2.15 % | 12.35 | 0.34 | 1.34 | 0.35 % | 12.81 | 0.33 |
| 30 seconds | 1.43 | 2.54 % | 12.06 | 0.34 | 0.84 | 0.45 % | 12.78 | 0.33 |
| 20 seconds | 1.21 | 3.20 % | 11.73 | 0.34 | 0.67 | 0.86% | 12.68 | 0.33 |
| 10 seconds | 1.00 | 4.77 % | 10.73 | 0.35 | 0.35 | 1.73 % | 12.46 | 0.34 |
| 5 seconds | 0.89 | 8.64 % | 10.08 | 0.37 | 0.37 | 2.94 % | 11.85 | 0.34 |
| 2 seconds | 0.84 | 16.65 % | 7.35 | 0.41 | 0.40 | 6.98 % | 10.47 | 0.36 |
| 1 seconds | 0.82 | 26.74 % | 4.37 | 0.53 | 0.53 | 12.44 % | 8.34 | 0.39 |
| 0.5 seconds | 0.82 | 38.15 % | 2.78 | 0.60 | 0.61 | 20.62 % | 6.22 | 0.42 |
| | Case 3 | | | | Case 4 | | | |
| **Block interval** | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ |
| 25 minutes | 22.50 | 0.03 % | 12.89 | 0.33 | 22.44 | 0.02 % | 12.89 | 0.33 |
| 10 minutes | 9.41 | 0.14 % | 12.86 | 0.33 | 9.18 | 0.13 % | 12.87 | 0.33 |
| 2.5 minutes | 2.60 | 0.16 % | 12.86 | 0.33 | 2.59 | 0.15 % | 12.86 | 0.33 |
| 1 minute | 1.30 | 0.25 % | 12.83 | 0.33 | 1.27 | 0.29 % | 12.77 | 0.33 |
| 30 seconds | 0.84 | 0.51 % | 12.77 | 0.33 | 0.84 | 0.52 % | 12.69 | 0.33 |
| 20 seconds | 0.69 | 0.85 % | 12.68 | 0.33 | 0.68 | 0.82 % | 12.68 | 0.33 |
| 10 seconds | 0.33 | 1.41 % | 12.54 | 0.34 | 0.53 | 1.59 % | 12.50 | 0.34 |
| 5 seconds | 0.45 | 2.99 % | 11.80 | 0.34 | 0.44 | 3.05 % | 11.78 | 0.34 |
| 2 seconds | 0.39 | 7.28 % | 10.37 | 0.36 | 0.38 | 7.10 % | 10.42 | 0.36 |
| 1 seconds | 0.38 | 12.59 % | 8.24 | 0.39 | 0.37 | 12.52 % | 8.30 | 0.39 |
| 0.5 seconds | 0.49 | 20.87 % | 6.16 | 0.42 | 0.36 | 21.10 % | 6.02 | 0.42 |

**Table 5.6:** Impact of the block interval on the median block propagation time ($t_{MBP}$) in seconds, and the stale block rate $r_s$, $v_d$ and $r_{rel}$ given the current Bitcoin block size distribution, an adversary with $\alpha = 0.3$ and $k = 6$. Case 1 refers to the standard block propagation mechanism, Case 2 refers to standard mechanism plus unsolicited block push, Case 3 to the combination of Case 2 plus the relay network and Case 4 to the send headers with unsolicited block push and relay network.

This also suggests, conforming with our MDP models, that the bigger the block size, the higher the relative revenue from selfish mining and the lower the double-spending value (cf. Table 5.7). It is however apparent that an efficient block propagation mechanism effectively allows the network to keep nearly the same security provisions against selfish mining and double-spending as we can see in Case 3 (standard propagation mechanism, unsolicited block push, relay network) and Case 4 (send headers propagation mechanism, unsolicited block push, relay network). This confirms that an efficient network propagation mechanism helps to increase the security of the blockchain. Interestingly, given the block propagation mechanism of Case 4, the resilience

| | Case 1 | | | | Case 2 | | | |
|---|---|---|---|---|---|---|---|---|
| **Block Size** | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ |
| 0.1 MB | 3.18 | 0.32 % | 12.80 | 0.33 | 2.12 | 0.03 % | 12.89 | 0.33 |
| 0.25 MB | 7.03 | 0.88 % | 12.67 | 0.33 | 4.93 | 0.11 % | 12.87 | 0.33 |
| 0.5 MB | 13.62 | 1.63 % | 12.48 | 0.34 | 9.84 | 0.13 % | 12.87 | 0.33 |
| 1 MB | 27.67 | 3.17 % | 11.79 | 0.34 | 20.01 | 0.38 % | 12.79 | 0.33 |
| 2 MB | 57.79 | 6.24 % | 10.57 | 0.36 | 44.6 | 1.12 % | 12.61 | 0.34 |
| 4 MB | 133.30 | 11.85 % | 8.20 | 0.38 | 126.57 | 5.46 % | 10.51 | 0.35 |
| 8 MB | 571.50 | 29.97 % | 4.11 | 0.53 | 875.97 | 15.64 % | 7.64 | 0.41 |
| | Case 3 | | | | Case 4 | | | |
| **Block Size** | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ | $t_{MBP}$ | $r_s$ | $v_d$ | $r_{rel}$ |
| 0.1 MB | 2.02 | 0.03 % | 12.89 | 0.33 | 2.02 | 0.2 % | 12.90 | 0.33 |
| 0.25 MB | 4.49 | 0.05 % | 12.88 | 0.33 | 4.46 | 0.17 % | 12.87 | 0.33 |
| 0.5 MB | 8.65 | 0.05 % | 12.88 | 0.33 | 8.64 | 0.06 % | 12.87 | 0.33 |
| 1 MB | 17.24 | 0.07 % | 12.88 | 0.33 | 17.14 | 0.07 % | 12.88 | 0.33 |
| 2 MB | 35.49 | 0.08% | 12.87 | 0.33 | 35.38 | 0.1 % | 12.86 | 0.33 |
| 4 MB | 78.01 | 0.12 % | 12.85 | 0.33 | 78.40 | 0.13 % | 12.66 | 0.33 |
| 8 MB | 555.49 | 0.43 % | 12.65 | 0.33 | 550.25 | 0.4 % | 12.68 | 0.33 |

**Table 5.7:** Impact of the block size on the median block propagation time ($t_{MBP}$) in seconds, the stale block rate $r_s$, $v_d$ and $r_{rel}$, given the current Bitcoin block generation interval and an adversary with $\alpha = 0.3$ and $k = 6$.

(in terms of double-spending value) does not significantly change in Table 5.7 when increasing the block size from 0.1 MB to 8 MB ($v_d$ changes from 12.9 to 12.68 block rewards respectively). Similarly, the relative revenue from selfish mining stays at $r_{rel} = 0.33$, when all miners use the relay network.

Currently, a number of proposals suggest to chunk blocks and download these chunks in parallel (e.g., Blocktorrent [133]). In a separate experiment that we conducted, we implemented a block propagation mechanisms that divides blocks into chunks of a few kilobytes that can be queried from multiple peers. Our results show that such a protocol does not improve the median block propagation time compared to the send headers and relay network protocol, when dealing with modest block sizes (i.e., smaller than 8 MB). This is due to the fact that a chunked block propagates slower than the $10^{th}$ and $25^{th}$ percentile of nodes owing to: *(i)* the communication overhead caused by the chunks, and *(ii)* because a node only forwards block chunks if the respective block has been validated.

| tps | $v_d$ | $r_{rel}$ | Block size | Block interval |
|---|---|---|---|---|
| 33.4 | 12.75 | 0.33 | 0.25MB | 30 seconds |
| 40 | 12.38 | 0.34 | 0.10MB | 10 seconds |
| 50 | 12.45 | 0.34 | 0.25MB | 20 seconds |
| 66.7 | 12.06 | 0.34 | 0.25MB | 15 seconds |
| 66.7 | 12.65 | 0.33 | 0.50MB | 30 seconds |
| 66.7 | 12.71 | 0.33 | 1.00MB | 1 minute |

**Table 5.8:** Throughput in transactions per second (tps) vs. security measured in $v_d$ and $r_{rel}$ for an adversary with 30% mining power, $k = 6$ and given 16 mining pools.

### Throughput

We now evaluate the throughput achieved by various blockchain instantiations. To this end, we vary the block size (from 0.1M B to 8 MB) and the block interval (from 0.5 seconds to 25 minutes) to capture a larger number of blockchain instances with our simulator. Here, we assume that the network relies on an efficient propagation mechanism (send headers with unsolicited block push and relay network for all miners). For each simulated blockchain instance, we compute the resulting throughput in transactions per second (tps), measure the stale block rate and infer $v_d$ and $r_{rel}$ in order to assess the blockchain's security with our MDP model (cf. Section 5.3). We also assume an average transaction size of 250 bytes, $k = 6$ confirmations against double-spending and an adversary with 0.3 mining power and $\gamma = 0.5$.

In Table 5.8, we selectively list candidate blockchain instances which could achieve a transactional throughput beyond 60 tps and achieve similar security provisions to the existing Bitcoin system. Clearly, our results indicate that different parameter configurations can yield the same throughput—though with different security provisions (due to a different stale block rate). In particular, we observe that low consensus intervals offer less security compared to a higher consensus interval given the same overall throughput, since the network requires more round trips in order to commit the same information to the blockchain. Our results show that there is considerable room to enhance the scalability of existing PoW without significantly compromising security.

## 5.5 Concluding Remarks

In this work, we introduced a novel quantitative framework to objectively compare PoW blockchains given real world network impacts and blockchain parameters. Our framework enables us to evaluate the impact of network-layer parameters on the security of PoW-based blockchain. By doing so, we argue that our work lays the foundations to objectively compare the security provisions of different PoW blockchain instances. Namely, our framework allows us to push the boundaries of PoW powered blockchains in terms of throughput in transactions per second, while observing the impact on the security provisions of the blockchain in terms of optimal selfish mining and double spending strategies.

For instance, we find that Ethereum needs at least 37 block confirmations in order to match Bitcoin's security with 6 block confirmations, given an adversary with 30% of the total mining power. Our results indirectly suggest that Bitcoin's blockchain offers more security than Ethereum's blockchain which rewards miners with uncle rewards and performs uniform tie breaking for blockchain fork resolutions. Our results additionally indicate that existing PoW blockchains can achieve a throughput of 60 transactions per second—without significantly affecting the blockchain's security. To the best of our knowledge, this is the first contribution that quantitatively evaluates the impact of the stale block rate on optimal double-spending and selfish mining resistance of a PoW blockchain (cf. Figure 5.8 and Figure 5.4). By doing so, our results quantitatively capture the security of transactions based on their values, and on the block confirmations—effectively quantifying the level of security achieved by the famous required six block confirmations in Bitcoin.

Our insights do not only allow merchants to take into account the security provisions when accepting transactions and to assess their respective risk of double-spending, but also help miners in quantifying any blockchain's resilience against selfish mining.

## 5.6 Appendix

| State × Action | Resulting State | Probability | Reward | Condition |
|---|---|---|---|---|
| $(l_a, l_h, \cdot, nr)$, adopt | $(1, 0, \text{relevant}, nr)$ | $\alpha$ | $-c_m$ | - |
| | $(0, 1, \text{relevant}, nr)$ | $(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(0, 0, \text{relevant}, nr)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \cdot, inc)$, adopt | $(1, 0, \text{relevant}, nr)$ | $\alpha$ | $r_u - c_m$ | - |
| | $(0, 1, \text{relevant}, nr)$ | $(1-\alpha)\cdot(1-r_s)$ | $r_u - c_m$ | - |
| | $(0, 0, \text{relevant}, nr)$ | $(1-\alpha)\cdot r_s$ | $r_u - c_m$ | - |
| $(l_a, l_h, \cdot, rel)$, adopt | $(1, 0, \text{relevant}, rel)$ | $\alpha$ | $-c_m$ | - |
| | $(0, 1, \text{relevant}, inc)$ | $(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(0, 0, \text{relevant}, rel)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \cdot, \cdot)$, override | $(l_a - l_h - 1, 0, \text{relevant}, nr)$ | $\alpha$ | $l_h + 1 - c_m$ | $l_a > l_h$ |
| | $(l_a - l_h - 1, 1, \text{relevant}, nr)$ | $(1-\alpha)\cdot(1-r_s)$ | $l_h + 1 - c_m$ | $l_a > l_h$ |
| | $(l_a - l_h - 1, 0, \text{relevant}, nr)$ | $(1-\alpha)\cdot r_s$ | $l_h + 1 - c_m$ | $l_a > l_h$ |
| $(l_a, l_h, \text{relevant}, nr)$, wait | $(l_a + 1, l_h, \text{relevant}, nr)$ | $\alpha$ | $-c_m$ | - |
| | $(l_a, l_h + 1, \text{relevant}, nr)$ | $(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(l_a, l_h, \text{relevant}, nr)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \text{relevant}, inc)$, wait | $(l_a + 1, l_h, \text{relevant}, inc)$ | $\alpha$ | $-c_m$ | - |
| | $(l_a, l_h + 1, \text{relevant}, inc)$ | $(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(l_a, l_h, \text{relevant}, inc)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \text{relevant}, rel)$, wait | $(l_a + 1, l_h, \text{relevant}, rel)$ | $\alpha$ | $-c_m$ | - |
| | $(l_a, l_h + 1, \text{relevant}, inc)$ | $(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(l_a, l_h, \text{relevant}, rel)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \text{active}, nr)$, wait | $(l_a + 1, l_h, \text{active}, nr)$ | $\alpha$ | $-c_m$ | $l_h > 6$ |
| | $(l_a + 1, l_h, \text{active}, rel)$ | $\alpha$ | $-c_m$ | $l_h \leq 6$ |
| $(l_a, l_h, \text{relevant}, nr)$, match | $(l_a - l_h, 1, \text{relevant}, nr)$ | $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ | $l_h - c_m$ | - |
| | $(l_a, l_h + 1, \text{relevant}, nr)$ | $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | $l_h > 6$ |
| | $(l_a, l_h + 1, \text{relevant}, inc)$ | $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | $l_h \leq 6$ |
| | $(l_a, l_h, \text{active}, nr)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | $l_h > 6$ |
| | $(l_a, l_h, \text{active}, rel)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | $l_h \leq 6$ |
| $(l_a, l_h, \text{active}, inc)$, wait | $(l_a + 1, l_h, \text{active}, inc)$ | $\alpha$ | $-c_m$ | - |
| | $(l_a - l_h, 1, \text{relevant}, nr)$ | $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ | $l_h - c_m$ | - |
| $(l_a, l_h, \text{relevant}, inc)$, match | $(l_a, l_h + 1, \text{relevant}, inc)$ | $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(l_a, l_h, \text{active}, inc)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \text{active}, rel)$, wait | $(l_a + 1, l_h, \text{active}, rel)$ | $\alpha$ | $-c_m$ | - |
| | $(l_a - l_h, 1, \text{relevant}, nr)$ | $\gamma\cdot(1-\alpha)\cdot(1-r_s)$ | $l_h - c_m$ | - |
| $(l_a, l_h, \text{relevant}, rel)$, match | $(l_a, l_h + 1, \text{relevant}, inc)$ | $(1-\gamma)\cdot(1-\alpha)\cdot(1-r_s)$ | $-c_m$ | - |
| | $(l_a, l_h, \text{active}, rel)$ | $(1-\alpha)\cdot r_s$ | $-c_m$ | - |
| $(l_a, l_h, \cdot, nr)$, release | $(l_a, l_h, \cdot, rel)$ | 1 | 0 | $l_h \leq 6 \wedge l_h > 1 \wedge l_a \geq 1$ |
| $(l_a, l_h, \cdot, \cdot)$, exit | exit | 1 | $l_a + v_d$ | $l_a > l_h \wedge l_a > k$ |

**Table 5.9:** State transition and reward matrices for an MDP for optimal double-spending strategies in Ethereum where $r_u$ is the uncle reward (i.e. $\frac{7}{8}$). Every state includes a flag (where $nr$ = not released, $rel$ = released, $inc$ = included) indicating whether an attacker block has been or will be included as an uncle in the honest chain. The release action corresponds to the release of the first block of the attackers fork with the intention to be included as uncle in the honest chain. Therefore, it is only feasible if $1 < l_h \leq 6$ and $l_a \geq 1$, since it is otherwise equivalent to a match or override or the honest chain is too long to include it as uncle. With the release action, no block is mined and a state transitions from not released to released, which transitions to included with the next block mined on the honest chain. In Ethereum, $\gamma$ is fixed at 0.5 and a match is possible even without a prepared block.

# Chapter 6

# Scalability Optimizations and Eclipse Attacks

## 6.1 Introduction

Given the growing investments in Bitcoin, and the increasing adoption by users, the security, scalability, and reliability of Bitcoin has received considerable attention in the literature. Several studies have challenged the security assumptions adopted by Bitcoin [75, 92, 98]. For instance, Eyal and Sirer [84] showed that selfish miners that command more than 33% of the total computing power of the network can acquire a considerable mining advantage in the network.

Given that the Bitcoin overlay network is densely connected, most of these studies assume that all transactions and blocks (and their order of execution) advertised in the system will be almost immediately available to the majority of Bitcoin nodes soon after their release into the network. Recently, a number studies briefly hinted on the possibility of delaying blocks [21] and transactions [20, 110] for a short amount of time (e.g., for 2 minutes).

In this chapter, we extend these observations and we show analytically and experimentally that an adversary can deny the delivery of blocks and transactions to victim Bitcoin nodes for a considerable amount of time. By doing so, this attack enables the adversary to alter the information received by Bitcoin nodes, and to modify their views of the ledger state. This is achieved by exploiting Bitcoin bandwidth opti-

mization techniques and the measures that are in place to tolerate network delays and congestion. The minimal requirement for this attack to succeed in practice is simply that the attacker can establish at least one connection to the victim. An even more powerful attack resulting in almost indefinite delays at the victim node only requires that the attacker can fill the victim's remaining open connection slots—without necessarily causing any network partitioning in the Bitcoin network.

Our results therefore motivate the need for a careful design of the scalability mechanisms adopted in Bitcoin. While existing mechanisms limit the amount of propagated information in the system to the minimum necessary, we show that these techniques come at odds with security and reduce the ability of the network to e.g., detect double-spending attacks, resolve, or prevent blockchain forks. For instance, our findings suggest that the 33% bound outlined by Eyal and Sirer [84] might be even further reduced. Namely, if a considerable fraction of miners does not rely on alternative relay networks to receive network updates [22], then malicious miners can temporarily prevent the propagation of recently mined blocks in the network, in order to further increase their advantage—while commanding less than 33% of the total computing power in the network. In this respect, we propose a modification of the block request process in Bitcoin to deter this misbehavior.

Our findings additionally uncover new vulnerabilities in the Bitcoin network's ability to handle fast payments. Fast payments refer to payments where the time between the exchange of currency and goods is short (in the order of a minute). In light of the vulnerabilities discovered in [98], Bitcoin incorporated a new countermeasure which consists of relaying the first double-spent transaction in order to enhance the security of fast payments [24]. In this respect, we show analytically and experimentally that an adversary can leverage our findings to circumvent this countermeasure and double-spend fast payments without bearing any risk of losing her money. We also show that payments which were only confirmed by few blocks might be reverted by the adversary. Based on our findings, we explore a number of solutions to deter this misbehavior and we estimate a lower-bound on the waiting time required to ensure the security of fast payments.

More specifically, our contributions in this chapter can be summarized as follows:

- We analytically and experimentally confirm that a resource-constrained adversary can abuse existing scalability measures adopted in Bit-

coin clients in order to deny information about newly generated blocks and transactions to Bitcoin nodes for at least 20 minutes. We then extend this basic attack and show how an adversary can continuously deny the delivery of such information.

- We validate our analysis by means of implementation using a handful of hosts located around the globe. Our results demonstrate the feasibility and easy realization of our attacks in current Bitcoin client implementations.

- We show that our results allow the adversary to considerably increase its mining advantage in the network, double-spend transactions in spite of the current countermeasures adopted by Bitcoin, and easily mount Denial-of-Service attacks.

- We propose a number of mitigations for hardening the security of the network against such a misbehavior without deteriorating the scalability of Bitcoin. Namely, we propose a modification of the block request management system in Bitcoin in order to detect any misbehavior in the delivery of blocks. Additionally, we leverage our findings to estimate the minimum amount of waiting time required to ensure, with considerable probability, the security of fast payments in Bitcoin.

The remainder of the chapter is organized as follows. In Section 6.2, we overview Bitcoin and summarize the measures deployed in the system to minimize the amount of propagated information. In Section 6.3, we show how an adversary can effectively delay information propagation in the network. In Section 6.4, we extend our analysis and discuss possible techniques to continuously prevent the delivery of Bitcoin blocks. In Section 6.5, we discuss the impact of our findings on the security of Bitcoin, and we outline possible countermeasures in Section 6.6. We conclude the chapter in Section 6.7.

## 6.2 Scalability Measures in Bitcoin

In this section, we overview the main scalability measures integrated in the Bitcoin system.

**Background** Recall (cf. Chapter 2) that Bitcoin is a loosely-connected P2P network, where nodes can join and leave the network at any moment. Bitcoin nodes are connected to the overlay network over TCP/IP.

A Bitcoin block is mined on average every 10 minutes and currently awards 12.5 BTCs to the generating miner. It was shown in [98] that Bitcoin block generation follows a shifted geometric distribution with parameter 0.19. This also suggests that there is considerable variability in the generation times; for example, one of the longest block generation time so far lasted almost 99 minutes (this corresponds to block 152,218).

During normal operations, miners typically work on extending the longest blockchain in the network. Due to the underlying PoW scheme, however, different miners can potentially find different blocks nearly at the same time—in which case a fork in the blockchain occurs. Forks are inherently resolved by the Bitcoin system; the longest blockchain which is backed by the majority of the computing power in the network will eventually prevail.

**Scalability Measures** Currently, almost 1 transaction per second (tps) [5] is executed in Bitcoin; this results in an average block size of almost 400 KB[1]. In an experiment that we conducted, we measured the amount of traffic observed by a full Bitcoin node[2]; our results show that Bitcoin nodes in- and outbound traffic heavily depends on the number of connections of the node. For instance, a node with about 70 connections witnesses on average 8.5 GB daily traffic[3], while a node with about 25 connections witnesses on average 3 GB traffic within 24 hours[4]. Given the increasing adoption of Bitcoin, the number of transactions, and the block sizes are only expected to increase. For example, if Bitcoin were to handle 1% of the transactional volume of Visa[5], then Bitcoin needs to scale to accommodate almost 500 tps—requiring considerably larger blocks to be broadcasted in the network.

Motivated by these factors, the current Bitcoin protocol implements various bandwidth optimizations, and measures in order to sustain its

[1] The maximum block size is currently limited to 1 MB which corresponds to less than 7 transactions per second.

[2] Our measurements were conducted over a period of 18 days, during which our node had on average 70 active connections.

[3] Measured over 20 days.

[4] Measured over 2 days.

[5] Currently, the Visa network is designed to handle peak volumes of 47,000 tps [32].

scalability and correct operation in spite of ever-increasing use. In what follows, we detail the existing measures taken by Bitcoin developers.

**Measure 1.** *Bitcoin combats the broadcasting of ill-formed blocks and transactions by maintaining an internal reputation management system.*

Whenever a node receives objects (e.g., blocks, transactions), it checks their correctness before forwarding them to other peers in the network. First, objects are validated based on their respective syntax and sizes, e.g., oversized objects are discarded. If this verification passes, the contents of the objects are subsequently validated. For transactions, this includes verifying the signature, the input and output coins used in the transaction; similarly, the PoW included in block headers is verified with respect to the current network difficulty.

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

**Measure 2.** *Bitcoin uses an advertisement-based request management system to minimize the information spread in the network.*

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

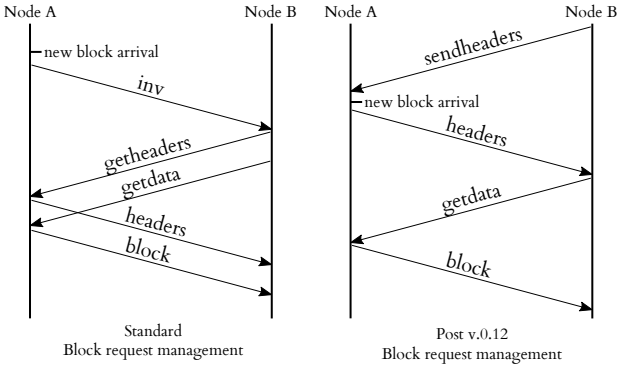



**Figure 6.1:** Summary of the request management system in Bitcoin.

By doing so, inventory messages limit the amount of data broadcast in the network. Notice that in case the object advertised corresponds to a block, neighbor $\mathcal{V}$ first requests the block header before the actual block data. Here, when a block header is advertised via a *headers* message, the receiving node internally stores the highest block known by the sending peer. The receiving node also validates any received header, by verifying its corresponding PoW. Transactions, on the other hand, are requested directly using a transaction *inv* message. This process is summarized in Figure 6.1.

To minimize bandwidth consumption, Bitcoin nodes request a given object only from a single peer, typically the first peer which first advertises the object. In an experiment that we conducted, we measured the traffic (cf. Figure 6.2) witnessed by a default Bitcoin client over a period of 24 hours[6]. We observe that, indeed, the transmission of blocks consumes a significant part of the bandwidth of our client. Requesting the same object from multiple peers entails downloading the same data several times, and therefore can only increase the bandwidth consumption of the system.

**Measure 3.** *Bitcoin relies on static timeouts in order to prevent blocking while tolerating network outages, congestion, and slow connections.*

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

[6]During our measurements, the client was connected to approximately 30 neighbors.

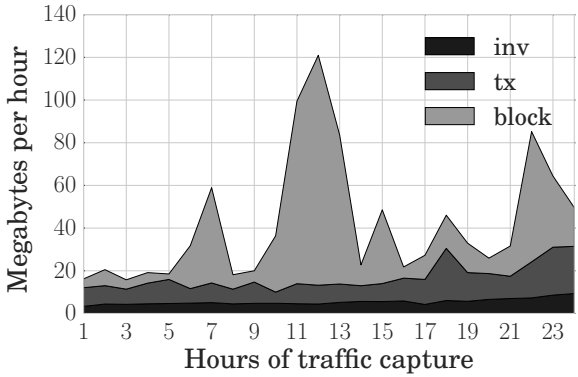



**Figure 6.2:** Hourly traffic distribution of a Bitcoin node, up and downstream, over 24 hours, w.r.t. to different Bitcoin messages. Here, 'tx' denotes transactions.

of the system. Currently, Bitcoin relies on timeouts in order to tolerate network delays—while preventing blocking. Blocking can occur e.g., when a node stops responding during communication.

For example, in Bitcoin version 0.10, the Bitcoin developers introduced a block timeout download of 20 minutes[7].

Similarly, for transactions, the Bitcoin developers introduced a 2-minute timeout. Notice that the choice of the timeout is a non-trivial task and depends on a number of parameters such as bandwidth, object size, latency, processing power, and the Bitcoin version of each node. On the one hand, overly long timeouts might deteriorate the quality of service of the whole network and can be abused to conduct e.g., double-spending attacks [26]. On the other hand, short timeouts might hinder effective communication under varying network conditions, or when communicating with slow peers.

In an experiment that we conducted, we measured the block transmission times in Bitcoin with respect to the nodes' bandwidths. Our results (cf. Figure 6.3) show that block retrieval times significantly vary depending on the connection of the node. For instance, fast nodes with 500 Mbps downstream take an average of 1.55 seconds to download a block, while slow nodes equipped with a 256 Kbps connection take

[7]Available from `https://github.com/bitcoin/bitcoin/pull/5608`

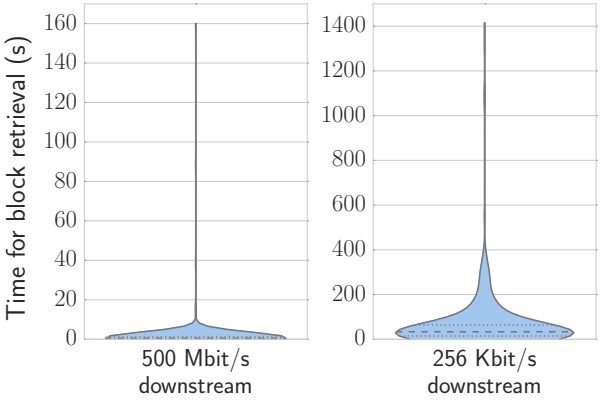



**Figure 6.3:** Block transmission times with respect to the connection speed. Here, we evaluate the time to download 400 consecutive Bitcoin blocks on a 500 Mbps and 256 Kbps connection.

an average of 71.70 seconds to download blocks. It is interesting to note that, during our measurements on the slow node, 0.5% of the generated blocks exceeded the 20 minute timeout adopted by current clients. *This shows that the current timeouts adopted in Bitcoin when sending/receiving blocks are indeed necessary for the correct delivery of blocks in the network.*

**Measure 4.** *Bitcoin clients keep track of the order of the received transaction advertisements. If a request for a given transaction is not delivered, the next peer in the list is queried.*

When a transaction $T$ is advertised via an *inv* message to a given node, the latter keeps track of the order of announcements with a first-in first-out (FIFO) buffer. Each time a peer advertises $T$, the peer is inserted in the buffer. Transaction $T$ is only requested from one peer at a time. For each entry in the buffer, Bitcoin clients maintain a 2-minute timeout, after which, the next entry in the buffer is queried for $T$.

## 6.3 Delaying Information Delivery

In this section, we show how an adversary can exploit the scalability measures of Bitcoin's request management in order to delay the delivery

of message advertisements.

For the purpose of our analysis, we assume that the adversary $\mathcal{A}$ is a full Bitcoin node, and has access to the entire blockchain. Here, the objective of $\mathcal{A}$ is to deny the delivery of an object $O$ for a short period of time from a specific node denoted by $\mathcal{V}$. In Section 6.4, we extend this analysis and show how an adversary can arbitrarily prolong the prevention of message delivery.

### 6.3.1 Necessary Requirements

We start by outlining the necessary conditions which need to be satisfied for $\mathcal{A}$ to successfully prevent the delivery of object information from a given Bitcoin node $\mathcal{V}$.

**Requirement 1:** $\mathcal{A}$ must be the first peer to send a message advertisement of object $O$ to $\mathcal{V}$. If $\mathcal{V}$ requests an object $O$ from a node which is not under the control of $\mathcal{A}$, then little can be done by $\mathcal{A}$ to prevent the node from sending $O$ to $\mathcal{V}$. However, if $\mathcal{A}$ is able to first advertise $O$ via an *inv* message to $\mathcal{V}$, then $\mathcal{V}$ will temporarily abstain from requesting $O$ from any other node in the network. As explained earlier, Bitcoin nodes only request the same data element from a single peer in order to minimize bandwidth consumption (cf. Measure 2). In order to be the first to advertise an object $O$ to $\mathcal{V}$, $\mathcal{A}$ needs to be an immediate neighbor of $\mathcal{V}$ in the Bitcoin overlay network (i.e., there is a direct TCP connection between $\mathcal{A}$ and $\mathcal{V}$).

**Requirement 2:** $\mathcal{V}$ should wait sufficiently long after a *getdata* message before requesting the data from another peer [110]. The longer $\mathcal{V}$ waits for $\mathcal{A}$ to send $O$, the more damaging is the misbehavior of $\mathcal{A}$ (cf. Section 6.5). As mentioned earlier, current Bitcoin implementations inflict a timeout of 20 minutes on $\mathcal{V}$ for blocks, and 2 minutes for transactions; after the timeout, $\mathcal{V}$ can request $O$ from another node (cf. Measure 3).

In what follows, we show how these requirements can be satisfied.

**Satisfying Requirement 1** As mentioned earlier, Bitcoin nodes verify the correctness of every received object before they re-broadcast it in the network. Notice that this process requires considerable time; for example, when receiving transactions, nodes verify the transactions'

signatures, and check that the input coins have not been spent previously. Similarly, for blocks, nodes check the correctness of the PoW in relation with the current difficulty in the network, and verify the correctness of every transaction that is confirmed in the block.

This offers a clear advantage for an adversary in order to satisfy Requirement 1. Indeed, an adversary can simply forward a given object of choice $O$ immediately after it receives it—without verifying its correctness. Since all remaining nodes in the system will have to perform a series of verification steps to verify $O$, the adversary is likely to be the first node to advertise $O$ to its neighbors. As explained in Measure 2, this ensures that $\mathcal{A}$'s neighbors will not request the object from any other node in the network until the timeout expires. Notice, that if $\mathcal{A}$ is interested in denying the delivery of $O$ to a specific node $\mathcal{V}$, then the adversary can advertise $O$ solely to $\mathcal{V}$. Here, in the case where $O$ is created by $\mathcal{A}$ (e.g., $O$ is a transaction), $\mathcal{A}$ can satisfy Requirement 1 by first advertising $O$ to $\mathcal{V}$, before broadcasting it in the network.

We stress here that $\mathcal{A}$ should be directly connected to $\mathcal{V}$. This is a reasonable assumption, since most full Bitcoin nodes do not exhaust their maximum 125 connections; in typical cases, nodes are therefore likely to accept connection request originating from $\mathcal{A}$. Notice that if $\mathcal{V}$ does not accept incoming connections (e.g., is located behind a NAT or firewall), Requirement 1 can also be satisfied if $\mathcal{V}$ connects to $\mathcal{A}$ (i.e., by requesting an outbound connection). Alternatively, $\mathcal{A}$ can try to compromise a neighbor of $\mathcal{V}$.

**Satisfying Requirement 2** To optimize bandwidth, recall that objects are only requested from one peer at a time (cf. Measure 2). This means that if $\mathcal{A}$ is the first node to advertise $O$ to $\mathcal{V}$, $\mathcal{V}$ will simply wait for $\mathcal{A}$ to send $O$ and will not request the same object from any other peer. During this period, $\mathcal{A}$ simply has to respond to the active *ping* messages of $\mathcal{V}$. Notice that if $\mathcal{A}$ does not actively respond to *ping* messages from $\mathcal{V}$ in this period, then $\mathcal{V}$ disconnects eventually before the object transmission timeout.

For blocks, the default timeout for $\mathcal{V}$ is set to 20 minutes; transactions, however, have a waiting timeout of 2 minutes. Given Measure 4, it is easy for $\mathcal{A}$ to increase the timeout for transactions simply by sending $x$ back to back *inv* messages for the same transaction. By doing so, $\mathcal{A}$ effectively increases the timeout specific to the advertised transaction by $2x$ minutes. We validate this analysis experimentally in

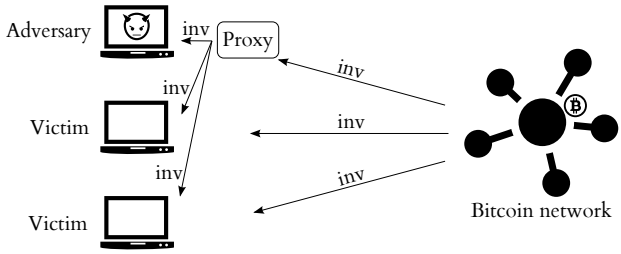



**Figure 6.4:** Satisfying Requirement 1. The adversary can use a simple relay proxy to forward *inv* messages without validating the correctness of the corresponding object.

Section 6.3.2.

### 6.3.2 Experimental Validation

In what follows, we empirically assess the probability that $\mathcal{A}$ succeeds in relaying first an *inv* message of an object $O$ to a given node $\mathcal{V}$ (cf. Requirement 1). For this purpose, we implement a proxy which relays specific *inv* messages directly before passing them to the Bitcoin software. Based on this proxy, we evaluate how likely $\mathcal{A}$ can be the first node to advertise an *inv* message to $\mathcal{V}$.

Our experimental setup is sketched in Figure 6.4. Here, we assume that $\mathcal{A}$ is located in Europe and operates an Intel i7 CPU (3.40GHz) with 32 GB RAM and a 500/400 Mbps Internet connection. In our setup, $\mathcal{A}$ connects to $\mathcal{V}$ through a direct TCP connection.

To remove any bias that might occur from a given network topology, we consider 10 different nodes emulating $\mathcal{V}$ geographically dispersed around the globe. We also vary the connectivity of $\mathcal{A}$ and $\mathcal{V}$ during our measurements (i.e., between 40 and 800 TCP connections to different full Bitcoin nodes for $\mathcal{A}$ and between 40 and 200 for $\mathcal{V}$). Because only full Bitcoin nodes forward blocks and transactions, only connections to full Bitcoin nodes are relevant for this experiment. We, therefore, modify the Bitcoin client, such that only connections to full Bitcoin clients are established and such that the maximum connection limit is kept constantly.

We measure the probability $P_r = \frac{r}{N}$, that $\mathcal{A}$ satisfies Requirement 1 as follows. We relay $N$ advertisements for objects to $\mathcal{V}$, and we compute the number of *getdata* and *getheaders* replies $r$ originating

from $\mathcal{V}$ received within the subsequent 30 seconds; recall that these messages provide sufficient evidence that $\mathcal{A}$ is the first relayer of an object to $\mathcal{V}$, and as such $\mathcal{V}$ will not request this object from elsewhere.

**Denying the Delivery of Blocks** We start by investigating the success probability of $\mathcal{A}$ in denying the delivery of blocks. Here, for different combinations of the number of connections of $\mathcal{A}$ and $\mathcal{V}$ (cf. Table 6.1), we forward $N = 100$ blocks generated after block height 350,000 and compute $P_r$.

Recall that to prevent the delivery of a block object $O$, $\mathcal{A}$ needs to directly relay the *inv* corresponding to $O$, without validating it. By doing so, $\mathcal{A}$ is faster in relaying the *inv* message than any other node in the system. To better assess the advantage of $\mathcal{A}$ in this case, we measured the time required to validate 100 blocks (from block height 353,000 onwards). Our results show that a single block requires on average 174 milliseconds to validate on an Amazon EC2 dual-core instance; notice that most of the overhead is spent in verifying the correctness of the transactions. In addition to the network latency for retrieving a Bitcoin block (cf. Measure 3), this gives 174 milliseconds advantage on average for $\mathcal{A}$ to succeed in denying the delivery of a block object.

In Table 6.1, we measure $P_r$ with respect to *(i)* the location of $\mathcal{V}$, *(ii)* $\mathcal{A}$'s number of connections and *(iii)* $\mathcal{V}$'s number of connections (cf. Requirement 1). By gradually increasing the number of connections of $\mathcal{A}$ from 40 to 800, we observe, that $\mathcal{A}$'s success increases with more connections. Namely, our results show that $P_r \approx 0.89$ when $\mathcal{A}$ has 800 connections. The more connections $\mathcal{A}$ maintains, the bigger is $\mathcal{A}$'s likelihood to receive a new block *inv* message before $\mathcal{V}$. Similarly, by gradually increasing the number of connections of $\mathcal{V}$ from 40 to 200, we observe, that the adversary's success decreases when $\mathcal{V}$ maintains more connections. We do not observe a strong correlation between the physical location, and the resulting network latency from $\mathcal{A}$ to $\mathcal{V}$. This implies, that it is crucial for $\mathcal{A}$ to be connected to nodes advertising blocks first.

In summary, our results clearly indicate that an adversary can successfully prevent the delivery of blocks to a particular node with considerable probability.

In another experiment, we measured the amount of time that a particular Bitcoin block can be denied to $\mathcal{V}$ (cf. Requirement 2). Here, when $\mathcal{A}$ forwards a block $B$ successfully as a first node to $\mathcal{V}$, $\mathcal{A}$ actively

| Connections of $\mathcal{A}$ | 40 | 80 | 200 | 800 | 80 | 80 | |
|---|---|---|---|---|---|---|---|
| Connections of $\mathcal{V}$ | 40 | 40 | 40 | 40 | 80 | 200 | |
| Victim node | $P_r$ | $P_r$ | $P_r$ | $P_r$ | $P_r$ | $P_r$ | avg. latency $\mathcal{A}$, $\mathcal{V}$ (ms) |
| Zurich | 0.63 | 0.40 | 0.97 | 0.90 | 0.35 | 0.29 | $0.95 \pm 0.3$ |
| Frankfurt | 0.29 | 0.46 | 0.61 | 0.83 | 0.29 | 0.34 | $43.1 \pm 0.3$ |
| Ireland | 0.67 | 0.47 | 0.94 | 0.85 | 0.24 | 0.18 | $28.6 \pm 0.5$ |
| N.Virginia | 0.55 | 0.92 | 0.88 | 0.96 | 0.34 | 0.18 | $91.0 \pm 0.0$ |
| Oregon | 0.38 | 0.80 | 0.82 | 0.90 | 0.36 | 0.12 | $171.0 \pm 3.0$ |
| N.California | 0.36 | 0.46 | 0.83 | 0.96 | 0.68 | 0.25 | $180.0 \pm 0.0$ |
| Tokyo | 0.55 | 0.86 | 0.96 | 0.98 | 0.22 | 0.16 | $246.0 \pm 4.9$ |
| Singapore | 0.40 | 0.51 | 0.76 | 0.92 | 0.63 | 0.19 | $303.0 \pm 4.6$ |
| Sydney | 0.31 | 0.37 | 0.60 | 0.77 | 0.35 | 0.21 | $303.0 \pm 4.6$ |
| São Paulo | 0.29 | 0.45 | 0.63 | 0.83 | 0.20 | 0.20 | $400.0 \pm 6.3$ |
| Average $P_r$ | $0.44 \pm 0.14$ | $0.57 \pm 0.20$ | $0.80 \pm 0.14$ | $0.89 \pm 0.07$ | $0.37 \pm 0.16$ | $0.21 \pm 0.06$ | $176.67 \pm 128.3$ |

**Table 6.1:** $P_r$ with respect to the number of connections of $\mathcal{A}$ and $\mathcal{V}$. Each experiment is measured over 100 consecutive blocks and across 10 different geographical locations. Each data point of $P_r$ corresponds to the average of 100 measurements; where appropriate, we report the standard deviation (labelled as '$\pm X$'). Note that we exclusively report the number of connections to full Bitcoin nodes.

responds to $\mathcal{V}$'s *ping* requests, but refrains from answering with a *block* message. We repeated this experiment for a number of times and observed that in all cases, $\mathcal{V}$ actively disconnects from $\mathcal{A}$ after exactly 20 minutes.

Notice that the costs borne by $\mathcal{A}$ to deny $\mathcal{V}$ a given block are modest. Namely, $\mathcal{A}$ needs to maintain an active TCP connection with $\mathcal{V}$ and to frequently reply to $\mathcal{V}$'s *ping* requests. Moreover, $\mathcal{A}$ needs to simply transmit 101 bytes comprising the *inv* message (i.e., 40 bytes for IP and TCP header, and 61 bytes for an *inv* message advertising one block object).

**Denying the Delivery of Transactions** In the following, we investigate the probability of $\mathcal{A}$ in successfully denying the delivery of a transaction. In our experiments, we assume that the transaction $T$ is created by $\mathcal{A}$ (for the reasoning why, refer to Section 6.5.2), and therefore, $\mathcal{A}$ is guaranteed to be the first node to relay the corresponding *inv* message.

Our evaluation is conducted as follows. We connect only two nodes to $\mathcal{V}$; one node corresponds to a machine controlled by $\mathcal{A}$, while the other node $\mathcal{C}$ emulates an honest machine in the Bitcoin network. In our experiments, $\mathcal{A}$ creates an *inv* message for transaction $T$ and forwards the *inv* to $\mathcal{V}$ and $\mathcal{C}$. Upon reception of the *inv* message, $\mathcal{C}$ also forwards the message to $\mathcal{V}$. This captures a realistic transaction announcement

in the Bitcoin network. We repeated this experiment 100 times during which $\mathcal{A}$ was able to successfully deny the delivery of $T$ each time for 2 minutes. Namely, for all 100 experiments, $\mathcal{V}$ was always requesting $T$ with a *getdata* message from $\mathcal{A}$, which was not responding to the request. After 120 seconds, $\mathcal{V}$ issued a *getdata* request for $T$ to the other node $\mathcal{C}$.

We then extended this attack such that $\mathcal{A}$ sends $T$'s *inv* message 10 times to $\mathcal{V}$, even before $\mathcal{C}$ issues $T$'s *inv* message. We repeated this experiment 100 times during which $\mathcal{V}$ did not request $T$ from any other peer for a period of 20 minutes. This is consistent with our observations in Section 6.2. Namely, every node which advertises a transaction object $T$ is inserted in a FIFO buffer and allocated a 2-minute timeout to deliver $T$. After 2 minutes, $T$ is requested from the next node in the FIFO buffer. By advertising $T$ $x$-times, the adversary can therefore deny the delivery of $T$ for $2x$ minutes and arbitrarily extend the denial time.

We point out that the communication overhead of this attack basically consists of 101 bytes for each 2 minutes of delay[8].

## 6.4 Extending the Block Delivery Time

We explained in Section 6.3.2 how $\mathcal{A}$ can deny the delivery of a block object to $\mathcal{V}$ for at least 20 minutes. In this section, we build on our findings and study the possibility of extending the time a block is denied from $\mathcal{V}$. The necessary requirements are that *(i)* $\mathcal{V}$ accepts at least one connection from $\mathcal{A}$, and *(ii)* $\mathcal{A}$ is capable of filling $\mathcal{V}$'s *remaining* open connection slots. For example, if $\mathcal{V}$ maintains 50 connections, $\mathcal{A}$ initiates one connection for advertising block *inv* messages, and fills the remaining $125 - 51 = 74$ open connection slots with default Bitcoin connections. Note that $\mathcal{V}$'s existing 50 connections do not prevent the attack.

In Bitcoin, blocks are transmitted by performing a header-first synchronization. Given the header of a block, any node can find the longest chain and verify the Proof of Work. Once the headers have been synchronized, Bitcoin nodes can selectively request the corresponding blocks from their peers.

Headers are typically synchronized on two different occasions:

[8]Note that multiple *inv* messages could be embedded into one *inv* message in order to lower the TCP overhead.

1. If $\mathcal{V}$ receives a new, previously unknown block *inv* advertisement from a peer $\mathcal{A}$, $\mathcal{V}$ actively requests the block headers from $\mathcal{A}$ with a *getheaders* message.

2. Once a node $\mathcal{A}$ connects to $\mathcal{V}$, both nodes exchange a Bitcoin *version* message. Each version message contains a counter for the most recent block height the respective node is aware of. If $\mathcal{A}$ has a higher block height than $\mathcal{V}$, $\mathcal{V}$ actively requests the block headers from $\mathcal{A}$ with a *getheaders* message.

The header-first synchronization is especially important for the purpose of our analysis, since it allows $\mathcal{V}$ to actively request a block from its peers as soon as it learns about new headers. More specifically, if $\mathcal{A}$ prevents the delivery of block $B$ to $\mathcal{V}$ and $\mathcal{V}$ receives $B$'s header from another peer, then $\mathcal{A}$ can prevent the delivery of $B$ for at most 20 minutes. That is, because after 20 minutes, $\mathcal{V}$ actively disconnects from $\mathcal{A}$ and requests the block $B$ from another peer. Clearly, if the latter peer is not under $\mathcal{A}$'s control, then $\mathcal{V}$ is likely to synchronize with the current main blockchain in the network. If $\mathcal{V}$, however, does not receive the header for block $B$, $\mathcal{V}$ does not actively request block $B$ from another peer after the 20-minute timeout, even if $\mathcal{V}$ received the block *inv* message from additional peers during the 20-minute timeout.

In order to continuously deny the delivery of block information from $\mathcal{V}$, $\mathcal{A}$ therefore needs to make sure that $\mathcal{V}$ does not receive block headers from the network. This can be achieved when the following two conditions are met.

**First relayer for all blocks:** $\mathcal{A}$ must be the first node to forward all block *inv* messages to $\mathcal{V}$. For example, when $\mathcal{A}$ starts to deny the delivery of block $B$ from $\mathcal{V}$, and wants to deny the delivery of all blocks up to and including block $B + 5$, $\mathcal{A}$ needs to be the first node relaying all block *inv* messages between and including $B$ and $B + 5$. Note that after a 20-minute timeout, $\mathcal{A}$ is required to resend the corresponding block *inv* message before other peers. Other nodes are unlikely to advertise older blocks, and therefore $\mathcal{A}$ can reliably extend the time a block is denied.

**Connection depletion:** In the mean time, $\mathcal{V}$ must not receive a new version message from another peer. This can be ensured if $\mathcal{A}$ can keep all *remaining* open connection slots of $\mathcal{V}$ occupied, such that $\mathcal{V}$ does not receive any *version* message from other peers. Notice

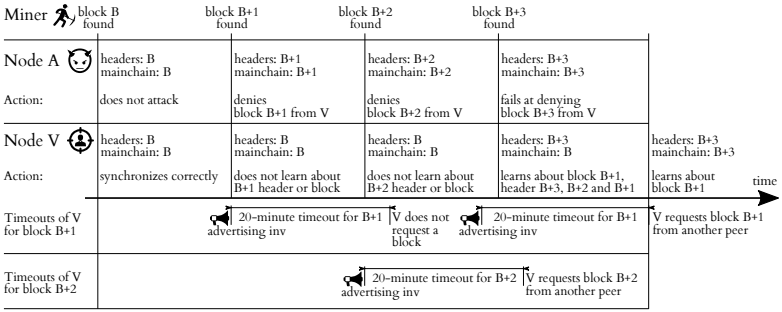



**Figure 6.5:** Example process of denying the delivery of multiple blocks. Here, $\mathcal{A}$ succeeds to deny the delivery of 2 consecutive blocks.

that existing connections do not exchange *version* messages and, do not impact this condition.

We exemplify the process of denying the delivery of 2 consecutive blocks in Figure 6.5. Here, we assume that $\mathcal{V}$ successfully receives $B$'s header and appends block $B$ to its local chain. Once block $B+1$ is found, $\mathcal{A}$ prevents it from being delivered to $V$. Here, a 20-minute timeout for $\mathcal{A}$ starts for block $B+1$. Once block $B+2$ is found, $\mathcal{A}$ also prevents the delivery of this block to $\mathcal{V}$. Here again, a 20-minute timeout for $\mathcal{A}$ starts for block $B+2$. When the timeout for block $B+1$ expires, $\mathcal{V}$ does not actively request a block, because it has not learned about any new headers. $\mathcal{V}$ would only receive the headers if $\mathcal{V}$ establishes a new connection to a peer with an up-to-date blockchain, or receives $B+1$'s *inv* message from another peer after the timeout. $\mathcal{A}$ now, however, can *re-advertise* block $B+1$ and thus re-activate another 20-minute timeout for block $B+1$.

Now, assume that when block $B+3$ is found, $\mathcal{A}$ is not the first node to relay $B+3$'s *inv* message. $\mathcal{V}$ consequently requests *getheaders* and *getdata* from the advertising peer. Because blocks $B+1$ and $B+2$, however, are currently being awaited from $\mathcal{A}$, $\mathcal{V}$ only receives block $B+3$. In order to validate block $B+3$ in the main blockchain, $\mathcal{V}$ requires the intermediate blocks $B+1$ and $B+2$. When the timeout for $B+2$ expires, $\mathcal{V}$ disconnects from $\mathcal{A}$ and requests block $B+2$ immediately from another peer. Still, $\mathcal{V}$ is not able to connect $B+2$ and $B+3$ to the blockchain, because $B+1$ is missing. Finally, when the timeout for $B+1$ expires, $\mathcal{V}$ requests block $B+1$ from another peer

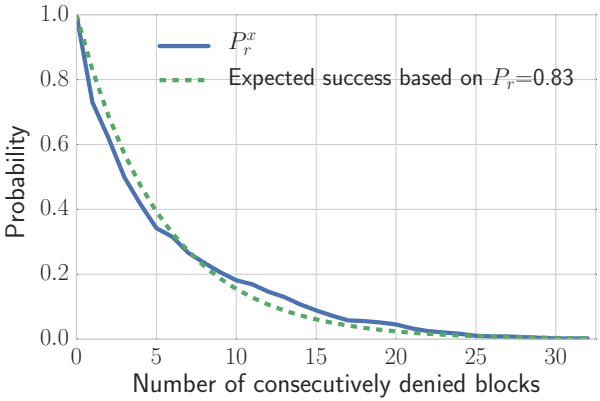



**Figure 6.6:** $P_r^x$ w.r.t. the number $x$ of consecutively denied blocks. Here, $\mathcal{A}$ maintains 80 and $\mathcal{V}$ between 11 and 25 connections to full Bitcoin nodes.

and is able to synchronize with the main blockchain.

Let $P_r^x$ denote the probability that $\mathcal{A}$ can successfully prevent the delivery of $x$ consecutive blocks. Clearly, the delivery of different blocks is an independent process, then $P_r^x = (P_r)^x$, where $P_r$ denotes the average probability of denying the delivery of a single Bitcoin block. We confirm this analysis by means of experiments in the following paragraphs.

## Experimental Validation

In what follows, we assess the probability $P_r^x$ that an adversary $\mathcal{A}$ can prevent the delivery of at least $x$ blocks from $\mathcal{V}$.

Our experimental setup is designed as follows. The adversary, in addition to trying to be the first to advertise consecutive blocks, makes sure that $\mathcal{V}$ does not establish new connections by filling all remaining connection slots of $\mathcal{V}$. In our setup, we attempt to continuously prevent the delivery of blocks to 5 different nodes (emulating 5 different $\mathcal{V}$) running the default Bitcoin clients[9]. During our study, the nodes' connections to full Bitcoin nodes[10] varied over time (cf. Figure 6.7).

[9]These nodes were synchronized to the blockchain for almost 72 hours and had 11 to 25 connections to full Bitcoin nodes.

[10]Recall that lightweight Bitcoin clients do not forward blocks or transactions

On the other hand, the adversary maintained at all times a fixed number of 80 connections to full Bitcoin clients. Here, all of the machines emulating $\mathcal{V}$ were located in Europe, with a latency below 200ms to $\mathcal{A}$.

In our experiments, the adversary performed a total of 2849 block denying attempts on all 5 nodes; $\mathcal{A}$ was successful in denying the delivery of a total of 2364 blocks—resulting in $P_r = 0.83$. In Figure 6.6, we measure $P_r^x$, the probability to prevent the delivery of at least $x$ blocks, as follows. We count the number of consecutive blocks, $x$, that are only requested (i.e., using a *getheaders* and *getdata*) from the adversary. Here, if we do not receive a *getheaders* and *getdata* message within 30 seconds for a given advertised block or any intermediate re-advertised block, we assume that the victim has requested that block from elsewhere, compute the resulting $x$, and restart the process. Our results (cf. Figure 6.6) confirm the analysis in Section 6.4, indeed show that $P_r^x \approx (P_r)^x$ and indicate that the adversary can succeed in preventing the delivery of consecutive blocks with considerable probability. For instance, the probability to prevent the delivery of 5 consecutive blocks is approximately 0.4.

In Figure 6.7, we evaluate the number of consecutively denied blocks with respect to the number of connections of $\mathcal{V}$. Our results confirm our previous observation (cf. Table 6.1) that the fewer connections $\mathcal{V}$ maintains, the more blocks can be denied. Recall that the more connections a node has (to full nodes), the earlier the node can receive information from the network.

## 6.5 Implications

In the previous sections, we thoroughly investigated how Bitcoin's request management system can be abused by a malicious adversary in order to delay information delivery in the Bitcoin network. In what follows, we evaluate the impact of our findings on the security of the Bitcoin system.

### 6.5.1 Increasing Mining Advantage

In [84], Eyal and Sirer show that a mining pool which controls more than 33% of the computing power in the network can considerably increase its mining advantage (by more than 10%) by withholding its

and are therefore not relevant for our study.

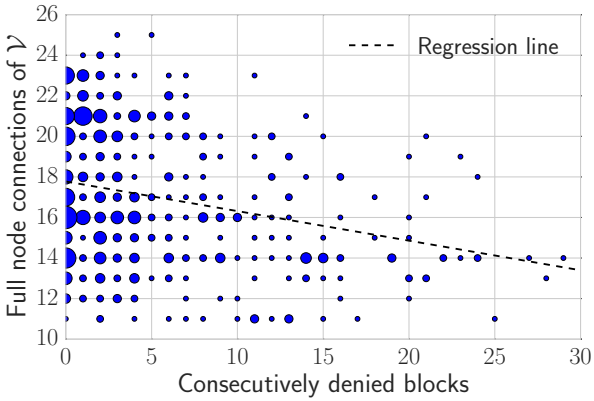



**Figure 6.7:** Number of consecutively denied blocks w.r.t. the connections of $\mathcal{V}$. Here, $\mathcal{A}$ maintains 80 full Bitcoin node connections.

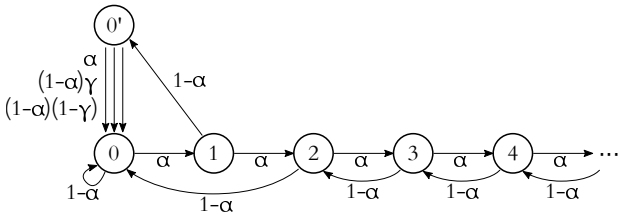



**Figure 6.8:** Selfish mining state machine adapted from Eyal and Sirer [84].

mined blocks until new blocks are found in the network. We show in what follows that the 33% bound of Eyal and Sirer can be even further reduced by leveraging our findings.

**Eyal and Sirer's Selfish Mining** In the selfish mining game (adapted from [84]), a selfish miner does not directly announce its newly mined blocks in the network and instead keeps them secret, until the remaining network finds new blocks. This strategy is captured in the state machine shown in Figure 6.8.

The state machine depicts the adversary's view on the current state of the selfish mining attack. In state 0, $\mathcal{A}$ and the network have the same view of the currently longest blockchain. $\mathcal{A}$ controls a fraction $\alpha$ of the computing power in the network, and is therefore likely to mine a

block with probability $\alpha$. With probability $1-\alpha$, the network finds and publishes a block, leading to state 0. Once $\mathcal{A}$, however, finds a block, she keeps it secret from the network and state 1 is reached. Moving on, different cases can arise.

First, the network can find a competing block $B_N$ with probability $1-\alpha$, moving the state to $0'$. In this case, $\mathcal{A}$ has an incentive to spread his single secret block $B_\mathcal{A}$ in the network as fast as possible, such that a fraction $\gamma$ of the network continues mining on $B_\mathcal{A}$. Namely, $\gamma$ is the fraction of the network that received $B_\mathcal{A}$ before $B_N$. Subsequently, three cases arise: *(i)* $\mathcal{A}$ finds a block with probability $\alpha$, *(ii)* the network finds a block by building upon $B_\mathcal{A}$ with probability $(1-\alpha)\gamma$, or *(iii)* the network finds a block building on $B_N$, with probability $(1-\alpha)(1-\gamma)$. For $\mathcal{A}$, these three cases generate two, one and zero block rewards, respectively.

Second, $\mathcal{A}$ can find a second block and also keep it secret, moving to state 2. Reaching state 2 guarantees $\mathcal{A}$ at least two block rewards. If the network finds a block (transition from $2 \rightarrow 0$ with probability $1-\alpha$), $\mathcal{A}$ can publish the two secret blocks, generate the longest chain and earn two block rewards. $\mathcal{A}$ keeps additional blocks secret moving to states further right and publishes these blocks individually while moving left in the state machine whenever the network finds a block.

**Beyond Selfish Mining** To deter this misbehavior, Eyal and Sirer [84] propose the following countermeasure. When a miner is aware of two competing blocks, the miner should propagate both blocks and select a random block to mine on. This solution does not take into account the case where an adversary can selectively deny miners from receiving a particular block—as confirmed by our results.

Even worse, by preventing the delivery of blocks to a fraction of the network, an adversary can create multiple virtual partitions in the network—each mining on different blocks. As we show below, this grants the adversary with an additional advantage in the selfish mining scenario. Notice that miners can make use of additional relay networks in order to receive blocks and transactions faster than the official P2P network can provide [23][11]. For instance, Matt Corallo's relay network [23] is optimized to transmit blocks and transactions as fast as possible, and is designed as a fallback mechanism for the official P2P

[11] Note that some of the largest mining pools do not make use of the relay network [22].

network. Namely, Corallo's relay network does not follow the *inv* and *getdata* protocol of the official P2P network, but instead directly relays blocks and transactions. Clearly, miners which rely on an additional relay network will immediately receive information about blocks and transactions. In what follows, we show that the advantage of an attacker is considerable in selfish mining assuming that a fraction $1 - P$ of the miners is connected to the additional relay network.

To this end, we extend the state machine from Eyal and Sirer by modeling the fact that $\mathcal{A}$ can deny the delivery of blocks to a fraction of the mining power. The modified state machine is depicted in Figure 6.9. Let's assume that the adversary currently reached state 3, representing 3 secret blocks. With probability $1 - \alpha$ the honest network finds a block $B_{1'}$, resulting in state change to state $2_1$. $\mathcal{A}$ manages to deny the delivery of this block to a fraction $P$ of the network, and consequently, only a fraction $1-P$ has learned about the new block $B_{1'}$. Three states arise: *(i)* the fraction $1-P$ finds a block (with probability $(1-P)(1-\alpha)$) and $\mathcal{A}$ publishes his remaining 2 secret blocks, resulting in state 0; *(ii)* $\mathcal{A}$ finds a block with probability $\alpha$ (resulting in state $3_1$), or *(iii)* the fraction $P$ which has not yet seen block $B_{1'}$ finds with probability $P(1 - \alpha)$ a block resulting in state $2_2$. All other cases follow the same principle; $\mathcal{A}$ virtually partitions the miners by denying the delivery of new blocks to other honest miners. This strengthens the selfish mining game as it leads to even more wasted computations by the honest miners.

Based on the state machine of Figure 6.9, we measure the revenue of the adversary in comparison with findings of [84]. For this purpose, we simulate both state machines; in each state, our simulator chooses the next state by generating the probability distribution of the possible transitions according to the given parameters. We performed up to 1,000,000 iterations for both state machines and per data point of Figure 6.10. Here, we assume that $\mathcal{A}$ is able to deny the delivery of 2 and 5 consecutive blocks with probabilities 0.5 and 0.25, respectively. This assumption conforms with our findings in Section 6.4. To compute the revenue, we adapt the scheme from [84]; we refer the readers to Appendix 6.8 for more details on the adopted revenue estimation.

Our results (cf. Figure 6.10) show that the 33% bound advertised by Eyal and Sirer can be considerably lowered. For instance, an adversary which succeeds in denying the delivery of 2 consecutive blocks from 50% of the network will profit from selfish mining if he controls 26.5% of the computing power in the network. This adversary effec-

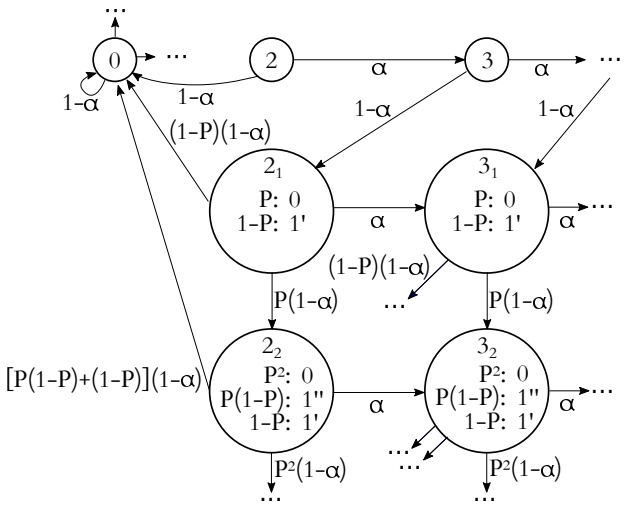



**Figure 6.9:** Extending Eyal and Sirer's state machine to capture the case where a selfish miner can deny the delivery of recently mined blocks to a fraction $P$ of the network.

tively controls the Bitcoin mining network given only 35% of the actual computing power—even if 50% of the miners leverage an additional relay network to receive transactions/blocks. *Even worse, our results show that an adversary (with $P = 0.5$ and 5 consecutively denied blocks) which commands less than 34% of the computing power in the network can effectively sustain the longest blockchain and therefore control the entire network.*

### 6.5.2 Double-Spending

In what follows, we show how an adversary can leverage our findings to double-spend *(i)* fast payments in which the corresponding transactions have not yet been included in any block of the main blockchain (also referred to as zero-confirmation transactions) [26], and *(ii)* 1-confirmation transactions, i.e., transactions which have been already confirmed in one Bitcoin block in the main blockchain. Notice that zero-confirmation and one-confirmation transactions are common to handle payments in which the time between exchange of the currency and goods is short.

Here, we assume that the adversary $\mathcal{A}$ creates two transactions $T_d$

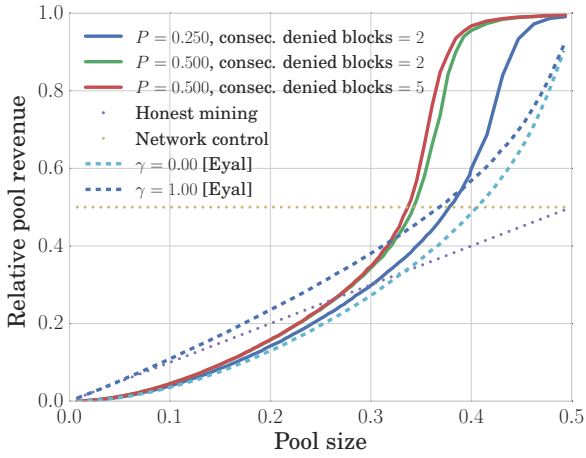



**Figure 6.10:** Relative revenue gain of selfish miners with and without the delaying of block information. For our simulations, we assume $\gamma = 0$ and always perform better than [84].

and $T_l$ sharing the same input coins. $T_d$ is a transaction whose output addresses are owned by $\mathcal{A}$, while the outputs of $T_l$ are owned by a vendor $\mathcal{V}$. Similar to [26], the goal of $\mathcal{A}$ is to convince $\mathcal{V}$ to accept $T_l$, acquire service from $\mathcal{V}$, while ensuring that $T_d$ is confirmed by miners and included in the main blockchain.

**Double-Spending of Zero-Confirmation Transactions** Zero - confirmation transactions are essential for the operation of several businesses. Recall that Bitcoin blocks are generated within 10 minutes on average which prevents the daily operation of businesses where the exchange between the currency and the service is short (e.g. in the order of few minutes). In light of the double-spending analysis in [98], an enhanced version of Bitcoin, referred to as Bitcoin XT [24], currently broadcasts the first double-spend transaction in the network. This allows a vendor to observe any double-spending attempt for an upcoming payment, and to consequently deny the processing of a fast payment. By February 2015, almost 16 full nodes in the Bitcoin network had adopted the double-spend relay protection of Bitcoin XT.

Motivated by our findings, we show in what follows that the pro-

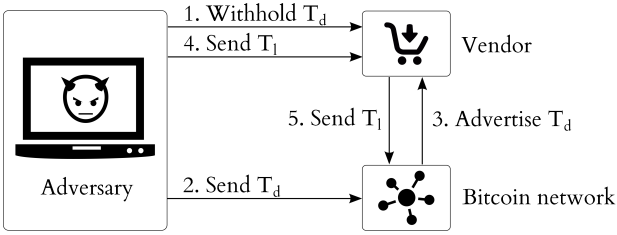



**Figure 6.11:** Circumventing the double-spend protection of Bitcoin XT.

tection of Bitcoin XT is not effective in preventing double-spending attacks of fast payments. *We also show that an adversary can perform double-spending attacks—without the risk of loosing money.*

To do so, the adversary first sends an *inv* message advertising $T_d$ to $\mathcal{V}$; this prevents $\mathcal{V}$ from receiving $T_d$ from the network—in spite of the protection embedded in Bitcoin XT. $\mathcal{A}$ subsequently broadcasts $T_d$ in the network, and sends $T_l$ to $\mathcal{V}$ shortly after (cf. Figure 6.11). From the viewpoint of $\mathcal{V}$, $T_l$ does not conflict with any other transaction, and therefore the trade can be concluded. Although other nodes in the network can observe the two conflicting transactions, $T_d$ and $T_l$, and issue the corresponding warnings, $\mathcal{V}$ cannot receive $T_d$ from its neighbors, and is therefore unable to detect this attack. Shortly after, the first observed transaction in the network (i.e. $T_d$) is typically included in the blockchain.

$\mathcal{A}$ can further prevent nodes in the network from detecting the attack, as follows. Similar to before, $\mathcal{A}$ issues $T_d$ in the network, while denying its delivery to $\mathcal{V}$ (e.g. for a 20 minute period). As soon as $T_d$ is confirmed in a block $B_d$, $\mathcal{A}$ directly prevents the delivery of $B_d$ to $\mathcal{V}$ by sending the latter an *inv* advertising $B_d$ (cf. Section 6.3.2). This ensures that $\mathcal{V}$ does not receive $B_d$ from the network for at least 20 minutes. In the meantime, $\mathcal{A}$ sends $T_l$ to $\mathcal{V}$ which will broadcast it in the network. Because Bitcoin XT, however, only relays conflicting transactions from the memory pool, a pool of not yet confirmed transactions, $T_l$ is not considered double-spent. $T_l$, nonetheless, is considered a conflicting transaction w.r.t. the already confirmed $T_d$, and therefore is not relayed in the network. In this way, $\mathcal{A}$ does not bear any risk of loosing her money since $T_d$ was already included in a block and will likely be accepted by most nodes in the network.

We empirically confirm our analysis using a testbed comprising of

three Bitcoin nodes emulating a vendor $\mathcal{V}$, the adversary $\mathcal{A}$ and a Bitcoin XT node[12]. All three nodes maintain a direct TCP connection in order to exchange Bitcoin transactions. We performed the aforementioned double-spending attack a number of times; our results confirm that although the Bitcoin XT node attempts to broadcast $T_d$, $\mathcal{V}$ does not receive $T_d$. This clearly shows that the protection of Bitcoin XT cannot deter double-spending.

**Double-Spending of 1-Confirmation Transactions** Notice that if $\mathcal{A}$ is connected directly to a (honest) miner $\mathcal{M}$, then $\mathcal{A}$ can also attempt to double-spend a transaction $T_l$ which has already been confirmed by one block $B_l$ in the blockchain. In this case, the only means for $\mathcal{A}$ to double-spend $T_l$ would be to include the double-spending transaction $T_d$ in a fork of the blockchain. Recall that forks frequently occur in the Bitcoin network [75], and are resolved automatically by choosing the longest fork chain. If the blockchain fork, which contains $T_d$, eventually emerges as the longest fork, $T_d$ will be accepted by the majority of the peers in the network and double-spending is successful, since $\mathcal{V}$ can no longer redeem $T_l$.

To double-spend $T_l$, $\mathcal{A}$ connects directly to both $\mathcal{V}$ and $\mathcal{M}$ in order to prevent the delivery of $T_d$ to both prior to broadcasting it in the network. As mentioned earlier, this ensures that $\mathcal{V}$ and $\mathcal{M}$ do not receive $T_d$, but the rest of the network receives and mine for blocks which confirm $T_d$. $\mathcal{A}$ then sends $T_l$ to $\mathcal{M}$, with the aim that $\mathcal{M}$ confirms $T_l$ in a block.

With some probability, the Bitcoin network eventually includes $T_d$ in block $B_d$. Because $B_d$ contains the double-spend transaction $T_d$, $\mathcal{A}$ has to again prevent $B_d$ from being received by $\mathcal{V}$ and $\mathcal{M}$. Later on, if $\mathcal{M}$ finds and broadcasts a block $B_l$, then $B_l$ will be accepted by $\mathcal{V}$ since the latter does not receive any conflicting transaction $T_d$ nor conflicting block $B_d$. This process is summarized in Figure 6.12. Here, we assume that $\mathcal{M}$ does not control a large fraction of the computing power—otherwise $B_l$ will be likely included in the main blockchain. Notice that $\mathcal{A}$ can collude with $\mathcal{M}$ (or can mine for blocks by herself) to further increase the success probability of the attack.

[12]We employ the same setup as visualized in Figure 6.11, but replace the Bitcoin network with the Bitcoin XT node.

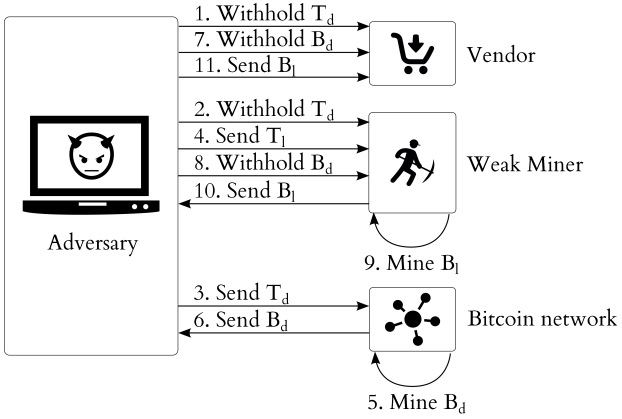



**Figure 6.12:** The adversary $\mathcal{A}$ performs double-spending of a *1-confirmation* transaction against a vendor, by leveraging the computing power of a weak miner. $T_d$ corresponds to the double-spending, and $T_l$ to the legitimate transaction.

### 6.5.3 Denial of Service

Delaying information propagation in a P2P-based cryptocurrency network such as Bitcoin can be very damaging as it results in the delay of processing payments. Our findings clearly suggest that e.g., Denial of Service (DoS) attacks on Bitcoin can be made more easily realizable by exploiting the object request management of Bitcoin. That is, an adversary that controls a number of nodes in the network (and establishes connections with many Bitcoin nodes) can effectively prevent the dissemination of information, such as blocks and transactions, in the entire network.

Notice that there are approximately 6000 reachable Bitcoin nodes in the network[13]. A resource-constrained adversary can simply connect to all these nodes and deny them the delivery of transactions and blocks.

Given an average block generation time $g$, and a timeout of 20 minutes per block (cf. Section 6.3.2), we estimate the number $n$ of required *inv* messages to prevent the delivery of consecutive blocks

[13] Available from: `https://getaddr.bitnodes.io/`

during time $t$:

$$n(t) = \sum_{i=1}^{\lceil \frac{t}{g} \rceil - 1} \left\lceil \frac{t - g \cdot i}{20} \right\rceil \tag{6.1}$$

In addition, $\mathcal{A}$ needs to fill and maintain the open connection slots of $\mathcal{V}$. To prevent the delivery of blocks to all 6000 reachable full Bitcoin nodes, $\mathcal{A}$ needs to maintain approximately 450,000 active Bitcoin TCP connections[14] and transmit about $101 \cdot 6000 \approx 600$KB of *inv* messages per block. During time $t$, every node requires moreover $n(t)$ *inv* messages. Assuming that $\mathcal{A}$ can operate 10000 connections per node[15], $\mathcal{A}$ requires approximately 45 nodes for attempting to constantly prevent block information propagation in the Bitcoin network.

Notice that by denying the delivery of blocks from full Bitcoin nodes, $\mathcal{A}$ implicitly prevents the reception of these blocks by the various lightweight Bitcoin clients[16].

*This analysis complements the work of [92] and shows that $\mathcal{A}$ can deny the delivery of blocks from the entire network using almost 45 nodes, and by transmitting only 600 KB per denied block for every 20 minute delay.*

## 6.6 Countermeasures

Based on our findings, we discuss and explore possible avenues for enhancing the security of Bitcoin without deteriorating its scalability.

**Alternative Relay Networks** As outlined in Section 6.5.1, miners (and users) can rely on additional relay networks to receive up-to-date information from the network. These networks allow miners to directly

[14]Assuming all 6000 Bitcoin nodes accept no more than 125 connections and maintain on average 50 connections. Clearly, some Bitcoin nodes accept significantly more than 125 connections (1500 or more).

[15]To limit the amount of information received from these 10,000 nodes, $\mathcal{A}$ can outsource a Bloom filter (similar to existing lightweight clients [88]) which match a small fraction of transactions in the system. By doing so, $\mathcal{A}$ reserves his bandwidth to perform his DoS attack.

[16]Lightweight Bitcoin clients (e.g. SPV nodes) do not validate nor maintain the full blockchain, and therefore get their information from a subset of the reachable Bitcoin nodes. Users typically prefer operating SPV clients, because they require significantly less processing power and disk storage.

exchange information, which would effectively prevent an adversary from denying/delaying object delivery.

Matt Corallo's relay network is one of the most prominent instantiations of an alternative relay network, currently operating 5 relay nodes—each serving between 20 and 40 clients. Corallo's network performs partial object validation and does not follow the request management system of Bitcoin to ensure a faster spread of information in those networks. This allows any entity to flood the network with ill-formed objects—which might explain the reason why Corallo's network is still not widely used [22].

Notice, however, that alternative relay networks can be built with different trust models, and can incorporate arbitrary policies to counter DoS; for instance, one can construct a small and trusted relay network only comprising one representative node from each centralized mining pool.

**Dynamic Timeouts** As described in Measure 3, Bitcoin relies on static timeouts in order to tolerate network delays. This implicitly assumes that all nodes and resources in the network are homogeneous—which is clearly not the case. As we show in Figure 6.3, slow nodes require considerable time to download blocks, while fast nodes can secure the download of blocks in few seconds.

We believe that dynamic timeouts would suit better the heterogeneity of resources in the Bitcoin network. For this purpose, we suggest the inclusion of the size of the message at advertisement time, which would allow each node to dynamically estimate the timeout value according to its resources, and the object size. For instance, when sending block advertisements, we suggest that the miner includes the block size into the block header[17]—which would allow receiving nodes to know the total block size and to appropriately estimate a dynamic timeout for any given block.

By doing so, we argue that the advantage of an adversary in abusing timeouts to delay block delivery can be considerably reduced.

We observe that the current timeouts employed by Bitcoin only capture the time starting from the *getdata* advertisement until the full data reception. Here, a timeout between the data request and the beginning of the data transfer could additionally be considered. We

[17]Block headers contain an unused field which was intended for the number of transactions confirmed in the block.

argue that such an approach would increase the costs of the adversary in delaying the delivery of information.

**Updating Block Advertisements** Based on our observations, we recommend updating the current block request system as follows:

**No *inv* messages:** We suggest to drop the advertisement of *inv* messages for blocks, and solely advertise the block headers before transmitting the blocks. By doing so, every receiver can immediately verify the correctness of the PoW, and learn about any new discovered blocks in the network. As mentioned in Section 6.4, this will ensure that an honest node will always learn about new blocks in the network even if the adversary fills all of its remaining connections to deny the delivery of such information. Notice that each block header is 80 bytes, while a block's *inv* message occupies 36 bytes. Therefore, we do not expect a considerable increase in the communication overhead due to this modification.

**Keep track of block advertisers:** Similar to transaction advertisements, we suggest that Bitcoin nodes keep track of the block headers' advertisers. This recommendation goes hand in hand with the advertisement based on block headers since it allows the node to request the blocks from the peers announcing the longest chain. Additionally, this allows the node to request the block from (a randomly chosen) advertising peer in case the chosen relay delays the delivery of the block.

**Handling Transaction Advertisements** As mentioned in Measure 4, a transaction is currently requested from the peers that advertised it first. If a given peer does not respond within an appropriate timeout, the transaction will be requested from the next peer stored in the FIFO queue. As mentioned in Section 6.3.2, this gives considerable advantage for the adversary to prolong the timeout before the receiver requests the transaction from any other peer. To remedy this, we suggest the following hardening measures:

**Filtering by IP address:** One way to deter against such an adversary would be to accept only one *inv* of the same transaction per IP address. Notice that this cannot entirely prevent the adversary from advertising the same transaction using different IP addresses.

**Randomly choosing sender:** Another complementary approach would be to randomly choose an incrementing number of peers to contact from the list of advertising peers if the first peer did not answer to the *getdata* request. Here, a transaction is first queried from the first advertising peer. If this peer however does not transmit the transaction within the specified timeout, the transaction is requested from two randomly chosen peers simultaneously, then from three peers, until the transaction is finally received. This will limit the advantage of an adversary which tries to advertise first the same transaction several times.

Given these suggestions, the probability to receive a transaction after the $n$'s timeout, when the first advertising peer is controlled by $\mathcal{A}$, given $i_a$ *inv* messages sent by $\mathcal{A}$ and a total of $i_t$ *inv* messages is computed as follows:

$$P_t(n) = \sum_{i=0}^{n}[\prod_{j=1}^{i}(\frac{\binom{i_a - \frac{j(j+1)}{2}}{j+1}}{\binom{i_t - \frac{j(j+1)}{2}}{j+1}}) \cdot (1 - \frac{\binom{i_a - \frac{(i+1)(i+2)}{2}}{i+2}}{\binom{i_t - \frac{(i+1)(i+2)}{2}}{i+2}})] \tag{6.2}$$

Equation 6.2 allows one to compute the probability $P_t(n)$ that a fast payment in Bitcoin is secure after a waiting time of $nt$, where $t$ is the timeout set by the node when handling transactions. Figure 6.13 depicts $P_t(n)$ with respect to the waiting time. Here, we assume that $i_t = 125$,[18] $t = 30$ seconds and that the adversary controls a fraction $\frac{i_a}{i_t}$ of the advertised *inv* messages. Our results show, that after 5 minutes waiting time, the node would be almost certain to receive a transaction, even if the adversary controls 95% of the advertised *inv* messages (in return about 6 out of 125 *inv* messages originate from honest peers). Notice that the receiving peer can be alerted if a sudden increase in *inv* advertisements occurs.

*We conclude, that fast payments should only be accepted after a waiting time nt to ensure with probability $P_t(n)$ that the transaction cannot be double-spent—even if the double-spend first-relay protection is implemented in the network. For a transaction request timeout of $t = 30$ seconds, this waiting time amounts to almost 5 minutes— which is half the duration of the block generation time.*

**Operating Several Bitcoin Nodes** Another effective alternative to harden the realization of our attack consists of the installation of

[18] The default maximum of neighbors per node is 125.

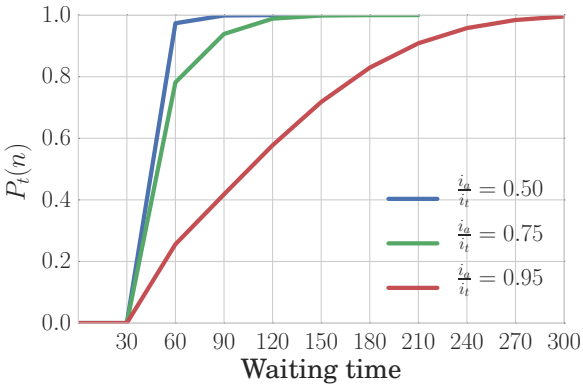



**Figure 6.13:** Waiting time vs $\frac{i_a}{i_t}$. Here, $t = 30$ seconds and $i_t = 125$.

several Bitcoin nodes (operated by the same entity) in the official Bitcoin network. By doing so, an adversary would have to prevent the delivery of objects to all these nodes in order to deny the entity from receiving an object of interest. As shown in Section 6.3.2, the more connections a Bitcoin entity exhibits, the less likely an adversary can delay the propagation of blocks and transactions. As shown in [98], this countermeasure can also effectively harden double-spending attacks on zero-confirmation payments in Bitcoin.

**Penalizing Non-responding Nodes** We also suggest extending the penalty system employed by Bitcoin (cf. Measure 1) to penalize non-responding peers. Namely, nodes which constantly delay information delivery after a *getdata* message should be penalized, and when appropriate disconnected from. Here, a careful design of the appropriate penalty is needed in order not to penalize slow nodes in the network.

## 6.7 Conclusions

In this chapter, we showed that the current scalability measures adopted by Bitcoin come at odds with the security of the system. More specifically, we showed that an adversary can exploit these measures in order to effectively delay the propagation of transactions and blocks to spe-

cific nodes—without causing a network partitioning in the system.

We analyzed the implication of our findings and showed that these threats enable an adversary to easily mount Denial-of-Service attacks on the entire network by preventing the delivery of blocks in the system. Moreover, mining pools can exploit this vulnerability to claim a higher mining advantage in the network. When combined with the results of Eyer and Sirer [84], our findings therefore suggest that selfish mining pools which command less than 33% of the computing power can considerably increase their mining advantage. Finally, our findings show that the countermeasure adopted in Bitcoin XT to prevent the double-spending of fast payments can be easily circumvented by a resource-constrained adversary.

Based on our findings, we explored a number of countermeasures in order to enhance the security of Bitcoin without deteriorating its scalability. Notice that our findings are not particular to Bitcoin and also apply to P2P networks and crypto-currencies which use a similar request management system such as Litecoin, and Dogecoin. We therefore hope that our findings solicit more research towards the re-design of the request management system of Bitcoin.

## 6.8 Appendix: Revenue for Selfish Mining

In what follows, we detail Eyal and Sirer's revenue scheme.

1. **Current state:** Any state, except 2 branches of length 1.
   **Event:** The pool finds a block, adds it to its secret chain.
   **Reward:** Block reward is determined later.

2. **Current state:** Two branches of length 1.
   **Event:** Pool finds a block.
   **Reward:** Pool publishes two blocks, revenue of two blocks.

3. **Current state:** Two branches of length 1.
   **Event:** Other find block on previous pool block.
   **Reward:** Pool and other obtain revenue of one block each.

4. **Current state:** Two branches of length 1.
   **Event:** Other find block on previous other block.
   **Reward:** Other obtain revenue of two blocks.

5. **Current state:** No secret block.
   **Event:** Other find block.
   **Reward:** Other obtain revenue of one block.

6. **Current state:** One secret block.
   **Event:** Other find block. Pool publishes secret block.
   **Reward:** Block reward is determined later, depends on $\gamma$.

7. **Current state:** Leading two secret blocks.
   **Event:** Other find block, only 1 secret left.
   **Reward:** Pool publishes secret blocks, two blocks revenue.

8. **Current state:** Leading more than two secret blocks.
   **Event:** Other find block, only 1 secret left.
   **Reward:** Pool publishes one block, revenue of one block.

# Part IV

# PoW Blockchain Privacy

# Chapter 7

# Privacy of Lightweight Clients

## 7.1 Introduction

Currently, a typical Bitcoin installation requires more than 70 GB of disk space, and requires considerable time to download and locally index blocks and transactions that are contained in the blockchain. In addition to disk space usage, the continuous growth of Bitcoin's transactional volume [9] incurs considerable overhead on the Bitcoin clients that need to verify the correctness of broadcasted blocks and transactions in the network. This problem becomes even more evident when users wish to perform/verify Bitcoin payments using resource-constraint devices, such as mobile devices.

To remedy this, the Bitcoin developers released a lightweight client, BitcoinJ [10], which supports a simplified payment verification (SPV) mode where only a small part of the blockchain is downloaded— thus supporting the typical usage of Bitcoin on constrained devices (e.g., smartphones, cheap virtual private servers). SPV clients were originally proposed by Nakamoto in [125] and were later extended to rely on Bloom filters [91] in order to receive transactions that are relevant to their local wallet. These Bloom filters embed all the addresses used by the SPV clients, and are outsourced to more powerful Bitcoin nodes; these nodes will then forward to the SPV clients those transactions relevant to their Bloom filters.

Bloom filters can be defined with a target false positive rate; by appropriately setting the target false positive rate of Bloom filters, Bitcoin developers aim to provide a suitable anonymity set to hide the addresses of SPV clients. As far as we are aware, the information leakage associated with the reliance on Bloom filters has not been yet thoroughly analyzed in the context of Bitcoin [10].

In this chapter, we address this problem, and explore the privacy provisions due to the use of Bloom filters in existing SPV client implementations. We show analytically and empirically that the current integration of Bloom filters within Bitcoin leaks considerable information about the addresses of Bitcoin users. More specifically, we show that the information leakage due to Bloom filters significantly depends on the number of addresses that each user possesses; notably, users who have a modest number of addresses ($< 20$) risk leaking all of their addresses by embedding them in a Bloom filter. This information leakage is further exacerbated when nodes restart their client, or generate additional Bitcoin addresses to their SPV clients; in these cases, the SPV clients re-compute new Bloom filters. We show that the computation of new Bloom filters considerably reduces the privacy of SPV clients.

Our work therefore motivates a careful assessment of the current implementation of SPV clients, prior to any large-scale deployment. In this work, we make the following contributions:

- We show that considerable information about users who possess a modest number of Bitcoin addresses (e.g., $< 20$) is leaked by a single Bloom filter in existing SPV clients.
- We show that an adversary can easily link different Bloom filters which embed the same elements—irrespective of the target false positive rate. This also enables the adversary to link, with high confidence, different Bloom filters which pertain to the same originator.
- We show that a considerable number of the addresses of users are leaked if the adversary can collect at least two Bloom filters issued by the same SPV client—irrespective of the target false positive rate and of the number of user addresses.
- Finally, we propose a lightweight and efficient countermeasure to enhance the privacy offered by SPV clients. Our countermeasure can be integrated with minimum modifications within existing SPV client implementations.

The remainder of this chapter is organized as follows. In Section 7.2, we briefly overview the operation of SPV clients. In Section 7.3, we introduce our system and attacker model. In Section 7.4, we analyze the privacy provisions of existing SPV client implementations when the adversary captures a single Bloom filter of an SPV client. In Section 7.5, we discuss the information leakage when the adversary can acquire multiple Bloom filters of an SPV node. In Section 7.6, we propose an efficient solution to enhance the privacy of users in SPV clients. We conclude the chapter in Section 7.7.

## 7.2 SPV Background

The continuous growth of Bitcoin transactional volume incurs significant computational overhead on the Bitcoin clients, when verifying the correctness of broadcasted blocks and transactions in the network. This problem becomes even more evident when users wish to perform/verify Bitcoin payments from resource-constrained devices such as mobile devices, tablets, etc.

To remedy that, lightweight client implementations (or simplified payment verification, SPV) have been proposed in [125]; SPV clients do not store the entire blockchain, nor do they validate all transactions in the system. Notably, SPV clients only perform a limited amount of verifications, such as the verification of block difficulty and the presence of a transaction in the Merkle tree, etc., and offload the verification of all transactions and blocks to the full Bitcoin nodes. Note that in order to calculate their own balance, SPV clients request full blocks from a given block height on; here, the full Bitcoin nodes can also provide "filtered blocks" to the SPV client that only contain relevant transactions from each block.

Unlike full Bitcoin nodes, SPV clients do not receive all the transactions that are broadcasted within the Bitcoin P2P network, but instead receive a subset of transactions, filtered for them by the full nodes to which they are connected[1]. To reduce bandwidth consumption, SPV clients make use of Bloom filters [91]. A Bloom filter [56] is a space-efficient probabilistic data structure which is used to test membership of an element. An SPV client constructs a Bloom filter by embedding all the Bitcoin addresses which appear in its wallets. Upon connection

[1]Currently, SPV clients connect to a default of four different randomly chosen nodes

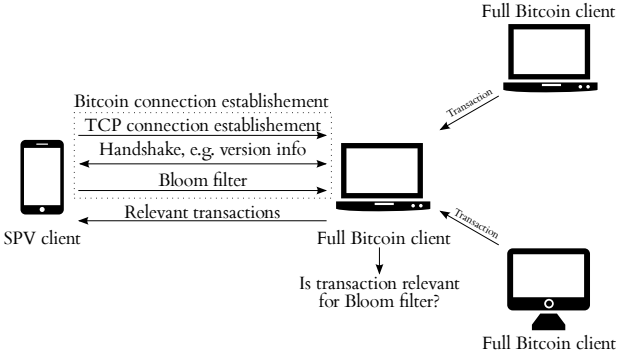



**Figure 7.1:** Sketch of the operation undergone by an SPV client. SPV clients connect to a full Bitcoin node, which only forwards to the SPV clients the transactions relevant to their Bloom filters.

to a full Bitcoin node, the constructed Bloom filter is outsourced to the full node following an initial handshake protocol (cf. Figure 7.1). Whenever the full node receives a transaction, it first checks to see if its input and/or output addresses match the SPV client's Bloom filter. If so, the full node forwards the received transaction to the SPV client.

Bloom filters have been first proposed by Bloom in 1970; we refer the readers to [66] for detailed information on Bloom filters. In SPV clients [10], a Bloom filter $B$ of an SPV client is specified by the maximum number of elements that it can fit, denoted by $M$, without exceeding its target false-positive rate $P_t$. Let $m \leq M$ denote the number of elements that are inserted in $B(M, P_t)$. In SPV clients, the size of the filter $n$ is computed as follows:

$$n = -\frac{M \ln(P_t)}{(\ln(2))^2} \tag{7.1}$$

A Bloom filter $B$ basically consists of an array $B[.]$ of $n$ bits accessed by $k$ independent hash functions $H_1(.), \ldots, H_k(.)$, each of which maps an input string $x \in \{0,1\}^*$ to one of the $n$ bits of the array.

In SPV clients, $k$ is computed as follows:

$$k = \ln(2)\frac{n}{M} \tag{7.2}$$

To insert an element $x \in \{0,1\}^*$ to a Bloom filter $B$, then $\forall j \in \{1, \ldots, k\}, B[H_j(x)] \leftarrow 1$. Similarly, to query the presence of an ele-

ment $x \in \{0,1\}^*$ in $B$, then this entails computing $\wedge_{j=1}^{k} B[H_j(x)]$ (thus returning 1, if all corresponding bits are 1).

Bloom filters can generate a number of false positives, but cannot result in false negatives. The literature features a number of techniques to estimate the false positive rate of Bloom filters [66, 112]. In this chapter, we focus on the proposal due to Mullin et al. [112] to estimate the false positive rates in Bloom filters (which is computed over all possible inputs to the Bloom filter). More specifically, we compute the false positive rate of a filter $B(M, P_t)$ which has $m$ elements, $P_f(m)$, as follows [66]:

$$P_f(m) = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k \tag{7.3}$$

We acknowledge that there might be more accurate techniques for computing the false positive rates (e.g., [66]); our findings, however, show that the difference in false positives resulting from [66] and [112] was negligible and did not affect our results. We therefore elected to rely on the estimation of $P_f$ which appears in [112].

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

## 7.3 Model

We assume that lightweight SPV clients connect to the Bitcoin P2P network through full Bitcoin nodes. As described earlier, full nodes only inform the SPV clients about transactions specific to their corresponding Bloom filters.

We assume that the adversary can compromise one or more full Bitcoin nodes and eavesdrop on communication links in order to acquire one or more Bloom filters pertaining to an SPV client. Here, the goal of the adversary is to identify the Bitcoin addresses that are inserted within a Bloom filter created by a particular SPV client. The addresses inserted in the Bloom filter typically correspond to addresses that the SPV client is interested in receiving information about (e.g., these addresses typically belong to the wallet of the SPV client). For example, the adversary might be connected to the node which generated the Bloom filter or might try to assign an identity to nodes according to

their addresses, etc. Note that since the Bitcoin network provides currently less than 10,000 reachable full Bitcoin nodes, it is likely that regular nodes receive one or more filter pertaining to each SPV client over a sufficiently long period of time.

In our analysis, we assume that the adversary knows the parameters used to create a Bloom filter. This includes, for instance, the target false positive rate, $P_t$, that the Bloom filter is designed to achieve, and the number of hash functions $k$ used in the Bloom filter. Since Bitcoin is an open payment system, we also assume that the adversary has access to all addresses/transactions which appear in the blockchain, and to their respective order of execution.

Clearly, we assume that the adversary is computationally bounded. However, since the Bitcoin network only contains a bounded number of addresses, the adversary can simply check whether all existing addresses match to a given Bloom filter.

Note that an adversary who is connected to an SPV client can see the transactions issued by the client and could potentially use this in order to learn the clients' addresses. This can be countered e.g., by SPV clients using TOR whenever they issue Bitcoin transactions. Instead, in this work, we focus on analyzing the privacy issues of the use of Bloom filters within existing SPV client implementations, which as we show, cannot be solved by simply relying on anonymizing networks.

Let $\mathbb{B}$ refer to the set of all existing Bitcoin addresses. Furthermore, let $\mathcal{B}_i$ refer to the set of all elements $y$ in $\mathbb{B}$ that are members of Bloom filter $B_i$ (i.e., for which a query in $B_i$ returns true), and $\mathcal{F}_i \subset \mathcal{B}_i$ denote the set of false positives of $B_i$.

In our analysis, we assume that all elements map uniformly at random to the bits of the Bloom filter; that is, we assume that all addresses $y \in \mathcal{B}_i$ are equally likely to be a true member of $B_i$.

We further assume that the adversary can collect additional information from the system which can help her classify (a fraction of) the false positives exhibited by Bloom filters; for example, the adversary can try to identify some false positives generated by filters by analyzing two different Bloom filters which embed the same true positives (cf. Section 7.5). We capture this additional knowledge using the set $\mathbb{K} \subseteq \mathcal{F}_i$ which contains addresses in $\mathcal{F}_i$ that the adversary can correctly classify. We acknowledge that the adversary could also try to gather prior knowledge about the addresses inserted in the Bloom filter; in this work, we focus however on the case where the adversary does not have any prior knowledge about the true positives of the filter.

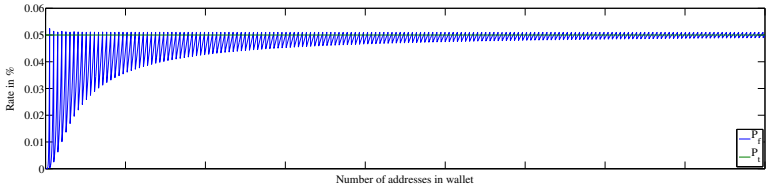



**Figure 7.2:** $P_f$, and $P_t$ computed analytically with respect to the number of addresses $N$. Here, we assume that the SPV client did not restart since initialization.

Our analysis throughout the rest of the chapter does not exploit the fact that the adversary knows the public keys of Bitcoin addresses. Indeed, in current implementations of SPV clients, both the addresses and their public keys are inserted in the outsourced Bloom filter. As such, if the adversary knows both the address and its public key, then she can trivially test whether an address is a true positive of the filter by checking whether both the address and its public key are inserted within the filter. If not, then it is highly likely that the address is a false positive of the filter. We believe that the inclusion of both the address and its public key in the Bloom filter is a severe flaw in the current SPV client implementations—and can be easily countered; we thus do not exploit this flaw in our analysis. In fact, more than 99% of all Bitcoin transactions consist of payments to Bitcoin addresses (or the public key hash); moreover, only 4587 out of 33 million total addresses in the system received transactions destined for *both* their public keys and their public key hashes[2]. This means that for the vast majority of Bitcoin clients, there is no need to include both the public keys and their hashes (i.e., the Bitcoin addresses) in the Bloom filters; inserting one or the other would suffice (in more than 99% of the cases)[3].

**Privacy Metric** We quantify the privacy offered by a Bloom filter using the probability, $P_{h_{(j)}}$, that the adversary correctly guesses any $j$ true positives of a Bloom filter among all positives that match the filter

[2]We obtained these numbers by parsing the Bitcoin blockchain till block # 296000.

[3]Note that only inserting the addresses in the Bloom filter would suffice since regular nodes can easily hash the public keys and check whether they match the Bloom filter. However, this clearly incurs computational overhead on regular Bitcoin nodes.

and which are not included in the knowledge of the adversary. More specifically, we measure $P_{h_{(j)}}$ achieved by a Bloom filter $B_i$, as follows: $P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} = \frac{N}{N+S} \cdot \frac{N-1}{N+S-1} \cdot \ldots$. Here, $N$ refers to the number of Bitcoin addresses inserted into $B_i$, and $S$ denotes the cardinality of the set $\{\mathcal{F}_i - \mathbb{K}\}$; $S$ therefore corresponds to all false positives that match $B_i$, but for which the adversary does not have any knowledge about. Therefore, the probability that the adversary correctly guesses *all* the addresses of $B_i$ is given by: $P_{h_{(N)}} = \prod_{k=0}^{N-1} \frac{N-k}{N+S-k} = \frac{N!S!}{(N+S)!}$. Clearly, the higher is $P_{h_{(.)}}$, the smaller is the privacy of the SPV node. Note that if the adversary is able to identify an SPV client (e.g., by some side channel information), then simply identifying any address pertaining to that client would be a considerable violation of its privacy. Otherwise, if the adversary can link a number of addresses to the same anonymous client, then the information offered by the clustering of these addresses offers the adversary considerable information about the profile of the client, such as its purchasing habits, etc.

Note that our privacy metric only assesses the probability of guessing addresses inserted within the Bloom filter. As mentioned earlier, these are the addresses that the SPV client is interested in receiving information about. $P_{h_{(.)}}$, however, does not necessarily capture the probability of guessing addresses which belong to the user of the SPV client. Indeed, addresses inserted within the filter do not necessarily have to belong to the user; for instance, the privacy of users—who only embed in their Bloom filters $L \leq N$ addresses—can be quantified by computing $\frac{L}{N} P_{h_{(.)}}$.

Table 7.1 summarizes the notation used throughout the chapter.

## 7.4 Information Leakage due to a Single Bloom Filter

In this section, we start by analyzing the privacy provisions of existing SPV clients when the adversary acquires a single Bloom filter pertaining to an SPV client. In Section 7.5, we address the case when multiple Bloom filters are acquired by the adversary.

| | |
|---|---|
| $\mathbb{B}$ | All existing Bitcoin addresses |
| $B_i$ | Bloom filter $i$ |
| $\mathcal{B}_i$ | Positives of the Bloom filter |
| $\mathcal{F}_i$ | False positives of the Bloom filter |
| $N$ | Number of addresses inserted in the Bloom filter |
| $S$ | False positives, the adversary has no knowledge about |
| $M$ | Maximum number of elements the Bloom filter fits |
| $m$ | Number of elements inserted in the Bloom filter |
| $n$ | Size of the Bloom filter (in bits) |
| $k$ | Number of hash functions of the Bloom filter |
| $P_f$ | Actual false positive rate |
| $P_t$ | Target false positive rate |
| $P_{h_{(j)}}$ | Probability of correctly guessing any $j$ true positives |
| $X$ | Number of bits in the Bloom filter set to one |

**Table 7.1:** Notations used throughout the chapter.

### 7.4.1 Leakage under a Single Bloom Filter

In existing SPV clients (which use the Bitcoinj library), a node initializes its Bloom filter $B_i$ with a random nonce $r$, and specifies its target false positive rate $P_t$ which can be achieved when a number of elements $M$ have been inserted in the filter. $M$ is equal to $M = m + 100 = 2N + 100$, where $N$ is the number of Bitcoin addresses inserted in $B_i$.[4] Note that a Bitcoin address is inserted into the Bloom filter by adding both the corresponding public key and the public key hash to the filter; therefore $m = 2N$.

By default, the target false positive rate of the Bloom filter $P_t$ is set to 0.05%[5]. The size of the filter $n$ and the number of hashes $k$ are computed given Equations 7.1 and 7.2, respectively.

Note that at initialization time, the SPV client is only equipped with $j = 1$ Bitcoin address. The corresponding Bloom filter will then be initially created to fit $M = 102$ elements, and will only contain $m = 2$ elements. However, if the SPV client restarts (e.g., mobile phone reboots, mobile application is restarted), then the Bloom filter will be re-computed with $M = 2j + 100$ (the SPV client stores the Bloom filter in volatile memory). In either cases, when the user acquires 50 or more additional addresses such that $m > M$, then the SPV client will resize

[4]The additional number '100' is added to $m$ by the Bitcoin developers in order to avoid the recomputation of a new filter in the case where a user inserts up to 50 additional Bitcoin addresses.

[5]The Bitcoin developers claim that a target false positive rate of 0.1% should provide "very good privacy" [18].

the Bloom filter by re-computing $M = 2N + 100$, and will send the updated Bloom filter to the full Bitcoin nodes that it is connected to. This process re-iterates whenever the number of addresses inserted in $B_i$ increase such that $m > M$.

In Figure 7.2, we analytically compute $P_t$ with respect to the number of addresses $N$ that an SPV client is equipped with. Here, we assume that the adversary has access to only one Bloom filter pertaining to the SPV client, and that the adversary has no prior knowledge about addresses in Bitcoin (i.e., $\mathbb{K} = \phi$). Note that given $n$ and $k$, the number of elements contained in a Bloom filter can be estimated by the adversary as follows [130]:

$$m \approx -n\frac{ln(1 - \frac{X}{n})}{k} \tag{7.4}$$

Here, $X$ corresponds to the number of bits of the Bloom filter set to one. Given $n$ and $P_t$, $M$ can also be computed by the adversary from Equation 7.1.

Since $\mathbb{K} = \phi$, $S + N = |\mathcal{B}_i|$ (i.e., the number of all existing Bitcoin addresses which match $B_i$)[6]. Note that, in April 2014, the Bitcoin blockchain comprised nearly $|\mathbb{B}|$ =33 million addresses. This means that an adversary can simply try all possible addresses in the Bitcoin system in order to compute $\mathcal{B}_i$. In doing so, it is straightforward to see that:

$$P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} \approx \prod_{k=0}^{j-1} \frac{N-k}{N+|\mathbb{B}-N|P_f(2N)-k}, \tag{7.5}$$

where $N \ll |\mathbb{B}|$, and $m = 2N$ is the number of elements contained in the Bloom filter seen by the adversary. In analyzing our findings, we distinguish two cases:

**1)** $2N/M \leq 0.4$ **and** $N < 100$ In Figure 7.2, we compute $P_t$ and $P_f$ with respect to the number of addresses $N$. Our results show that $P_{h_{(1)}}$ (the probability of correctly guessing one address as a true positive) is large when $2N/M \leq 0.4$, as long as $N < 100$. Given a modest number of addresses in the Bloom filter (i.e, $N < 100$), $P_f(m = 2N)$ is small when $\frac{m}{M}$ is small. As $\frac{m}{M}$ increases, $P_f(m = 2N)$ increases

[6] $|X|$ denotes the cardinality of set $X$.

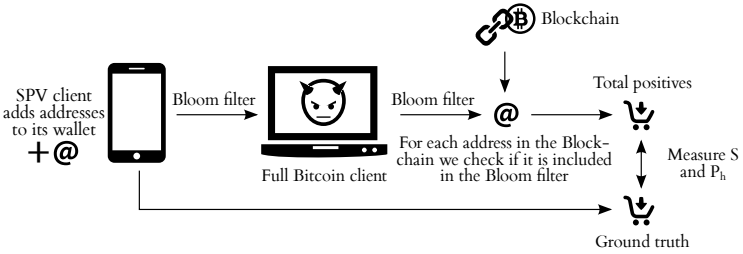



**Figure 7.3:** Experimental setup. We constructed around 18,060 different Bloom filters of various sizes and pertaining to 10 different wallets.

(and $P_{h_{(1)}}$ decreases). As shown in Figure 7.2, $P_f(m = 2N)$ is less affected by $\frac{m}{M}$ for larger $M$. For example, when $N = 10$, $P_{h_{(10)}} = 0.99$ which corresponds to the probability of correctly guessing all the true positives when the SPV client has 10 addresses.

This means that the information leakage in SPV clients is considerable for new Bitcoin users, and for Bitcoin users which restart their SPV clients. For instance, at initialization time, the Bloom filter of SPV clients is typically instantiated using $M = 102$. Our results show that if the user is new in the Bitcoin system and only has 1 Bitcoin address, $P_{h_{(1)}} \approx 1$, which signals complete lack of privacy. Recall that this observation also holds when the SPV client restarts and $N < 100$; for instance, when $N = 20$, $m = 40$, $P_{h_{(1)}} \approx \frac{N}{N+|\mathbb{B}|P_f(m)} \approx 0.98$ and $P_{h_{(20)}} \approx 0.20$ (cf. figure 7.4). We validate our analysis experimentally in Section 7.4.2.

**2)** $2N/M > 0.4$ Conforming with our previous analysis, when $2N/M > 0.4$, $P_f(m = 2N)$ is close to $P_t$. Recall that in this case, the probability of correctly guessing one address reaches a local minimum when $N = \frac{M}{2}$ (cf. Figure 7.2). Our results also show that the global minimum achieved by $P_{h_{(1)}}$ is 0.002961 and is reached when the user has $N = 51$ addresses.

In addition, we analytically compute $P_{h_{(j)}}$ when the SPV client has 5, 10, 15 and 20 addresses. Our results in Figure 7.4 show that guessing all addresses given one filter which embeds less than 15 addresses can be achieved with almost 0.80 probability. This probability decreases as the number of addresses embedded within the filter increases beyond 15.

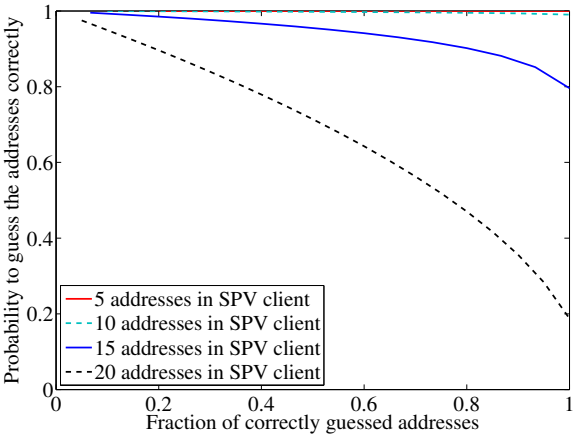



**Figure 7.4:** $P_{h_{(j)}}$ with respect to the number of addresses inserted in the Bloom filter. Given an SPV client with 5 addresses, all addresses can be guessed; when the SPV client has 20 addresses, 20% of the addresses can be guessed with almost 0.90 probability. Here, we assume that the user restarts its SPV client.

### 7.4.2 Experimental Evaluation

We now proceed to validate our analytical results empirically, by means of implementation. For that purpose, we parsed the blockchain from the genesis block mined in 2009 until the beginning of April 2014 using the parser in [15] and collected nearly 33 million distinct addresses. In our evaluation, we rely on *Bitcoin Wallet* [17], which builds upon the standard Bitcoinj library (0.12-SNAPSHOT of end of April 2014).

As mentioned earlier, Bitcoinj initializes by constructing one address by default. The user can subsequently add addresses to his wallet. Our implementation setup is depicted in Figure 7.3. Here, we construct 10 different Bitcoin wallets, and we gradually increase the number of Bitcoin addresses which populate each wallet from 1 address to 9,000 Bitcoin addresses. Whenever new addresses are added to the wallets, we compute the modified Bloom filters; we increase the number of addresses by a step of 2 addresses to reach 19 addresses, and then by a step of 5 addresses to reach 8,999 addresses. As such, our experiments resulted in the evaluation of a total of 18,060 different Bloom filters

| $N$ / $P_t$ | $P_{h_{(1)}}$ (0.05%) | $P_{h_{(1)}}$ (0.1%) | $P_{h_{(1)}}$ (0.5%) | $P_{h_{(\lceil N/2 \rceil)}}$ (0.05%) | $P_{h_{(\lceil N/2 \rceil)}}$ (0.1%) | $P_{h_{(\lceil N/2 \rceil)}}$ (0.5%) | $P_{h_{(N)}}$ (0.05%) | $P_{h_{(N)}}$ (0.1%) | $P_{h_{(N)}}$ (0.5%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1(±0) | 1(±0) | 1(±0) | - | - | - | 1 | 1 | 1 |
| 3 | 1(±0) | 1(±0) | 1(±0) | 1 | 1 | 1 | 1 | 1 | 1 |
| 19 | 0.76(±0.03) | 0.42(±0.03) | 0.03(±0.002) | 0.0433 | 0.000026 | 0 | 0 | 0 | 0 |
| 49 | 0.004(± 0.00032) | 0.0021(±0.00019) | 0.00035(±0.00002) | 0 | 0 | 0 | 0 | 0 | 0 |
| 54 | 0.36(±0.02) | 0.14(±0.0059) | 0.01(±0.00089) | 0 | 0 | 0 | 0 | 0 | 0 |
| 8,999 | 0.35(±0.002) | 0.21(±0.00075) | 0.05(±0.00032) | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 7.2:** $P_{h_{(.)}}$ with respect to $P_t$ and $N$. Each data point is averaged over 10 independent runs; we also show the corresponding 95% confidence intervals.

which contain various number of elements, and pertain to 10 different SPV clients. For each Bloom filter, we compute the matching set of Bitcoin addresses, we compare this set with the actual addresses inserted in each Bloom filter in order to compute $S$, and $P_{h_{(.)}}$. Since we assume here that the adversary has no a priori knowledge about Bitcoin addresses, $S$ corresponds to the number of existing Bitcoin addresses (among the 33,000,000 total Bitcoin addresses) that match the Bloom filter of the SPV client, and that are not the addresses of the SPV client (i.e., $S$ corresponds to the false positives generated by the Bloom filter).

Each data point in our experiments corresponds to the average of 10 independent measurements obtained from each of the 10 wallets. Our results (cf. Table 7.2) confirm our analysis in Section 7.4.1. More specifically, our results show that given few addresses (small $N$), $P_{h_{(.)}}$ is large; for example, when $N = 19$, $B_i$ results in an average of 6.1 false positives among all 33,000,000 addresses which corresponds to $P_{h_{(1)}} = 0.76$ and $P_{h_{(N/2)}} = 0.0433$. As $N$ increases to 49 addresses, $P_{h_{(1)}}$ and $P_{h_{(N/2)}}$ decrease to 0.004 and 0, respectively. Indeed, when $N = 49$, $N/M \approx 0.5$, since initially $B_i$ is computed with $M = 102$. Here, $B_i$ incurs $P_f(98) \approx P_t$ false positive rates. When $N$ increases beyond 51, then $m > M$, and the Bloom filter is resized to fit $M = 2N + 100$ elements. In this case, when $2N/M$ is small, $P_{h_{(.)}}$ is large. For instance, when $N = 54$, $P_{h_{(1)}}$ is 0.36, which corresponds to a total of 96 false positives seen by the adversary among all addresses in $\mathbb{B}$. This process re-iterates whenever $B_i$ is resized, however as $N$ increases, we observe that the fluctuations in $P_{h_{(.)}}$ decrease as $P_f(2N)$ converges towards $P_t$.

In Figure 7.5, we also measure $P_{h_{(1)}}$ and the number of false positives when the SPV client restarts and has to re-compute its Bloom filter $B_i$. As mentioned earlier, our results show that the restarting of an SPV client causes $P_{h_{(1)}}$ to significantly increase when compared to the case

where the client is not restarted. As mentioned earlier, at any restart and given $N$ addresses, the SPV client re-computes $B_i$ with a size of $M = 2N + 100$. Therefore, for modest values of $N$ (e.g., $N < 100$), $2N/M$ becomes small, thus resulting in modest $P_f(2N)$ and a large $P_{h_{(1)}}$. Our findings also show that as $N$ increases above 500 addresses, $P_{h_{(1)}}$ also increases. This is the case since the number of false positives provided by $B_i$ is bounded by $P_t|\mathbb{B}|$, and does not depend on $N$.

**Summary**

- We show that the probability $P_{h_{(.)}}$ that the attacker correctly guesses the addresses of the SPV client is large when the number $N$ of addresses inserted in the filter is smaller than 20. For example, if the SPV client has 15 addresses in its wallet and restarts, correctly guessing all of its address can be achieved with a probability of at least $P_{h_{(15)}} = 0.8$.
- Due to addition of the constant '100' to $M$ (the maximum number of elements the Bloom filter is designed for), the closer $m$ (the actual number of elements in the filter) is to $M$, the higher is the privacy offered by the Bloom filter.

## 7.5 Information Leakage due to Multiple Bloom Filters

In Section 7.4.1, we analyzed the information leakage in the presence of an adversary who has access to a single Bloom filter instance of an SPV client. In this section, we extend our analysis to cover a more powerful adversary who can acquire multiple Bloom filters pertaining to SPV clients.

### 7.5.1 Leakage under Multiple Bloom Filters

In what follows, we assume that the adversary can acquire $b > 1$ Bloom filters pertaining to different users. For example, the adversary might be connected to SPV clients for a long period of time, and receive their updated Bloom filter. Alternatively, the adversary can acquire additional Bloom filters by compromising/colluding with other full Bitcoin nodes. Note that in our analysis, we do not assume that the adversary

knows the correct association of the Bloom filters to the respective users.

**Two Bloom Filters** We start by analyzing the case where the adversary acquires two different Bloom filters $B_1$ and $B_2$. In the sequel, we focus on computing $P_{h_{(.)}}$ corresponding to filter $B_1$, which we assume to be the smallest of the two filters (in size). In analyzing the information leakage due to the acquisition of two Bloom filters, we distinguish two cases.

**1) $B_1$ and $B_2$ pertain to different users** Recall that each Bloom filter is initialized with a random seed, chosen uniformly at random from $\{0,1\}^{64}$. Therefore, if $B_1$ and $B_2$ pertain to different users, then it is highly likely that they are initialized with different random seeds. This means that the false positives generated by each filter are highly likely to correspond to different addresses. Moreover, since different users will have different Bitcoin addresses, $B_1$ and $B_2$ will contain different elements. Therefore, $\mathcal{B}_1 \cap \mathcal{B}_2$ is likely to be comprised of only few addresses, if any. Notably, when $B_1$ and $B_2$ pertain to different users, then $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx (|\mathcal{B}_1| - N_1)|\mathcal{B}_2|\frac{1}{|\mathbb{B}| - N_1} \tag{7.6}$$

$$\approx \frac{P_f(m_1)P_f(m_2)|\mathbb{B}|^2}{|\mathbb{B}| - N_1} \tag{7.7}$$

$$\approx P_f(m_1)P_f(m_2)|\mathbb{B}|, \tag{7.8}$$

where $N_1$ corresponds to the number of elements inserted in $B_1$. $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ is the expected number of elements which match $\mathcal{B}_2$ and $\mathcal{B}_1$. The number of elements in $\mathbb{B}$ which match $\mathcal{B}_2$ is given by $P_f(m_2)|\mathbb{B}|$. Then, $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ can be computed by assuming a binomial distribution with success probability $P_f(m_2)$, and with $P_f(m_2)|\mathbb{B}|$ number of trials.

Note that the adversary can compute $m_1$ (using Equation 7.4); if $m_1 > |\mathcal{B}_1 \cap \mathcal{B}_2|$, then this offers a clear distinguisher for the adversary that the two acquired Bloom filters $B_1$ and $B_2$ pertain to different user wallets.

**2) $B_1$ and $B_2$ pertain to the same user** On the other hand, in the case where $B_1$ and $B_2$ correspond to the same SPV client, three

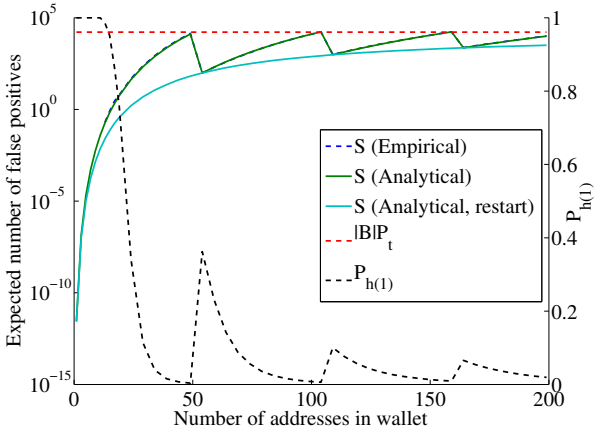



**Figure 7.5:** $P_{h_{(.)}}$ and $S$ computed experimentally for the first 200 insertion of Bitcoin addresses into the Bloom filter (with and without restart of the SPV client).

sub-cases emerge:

**$B_1$ and $B_2$ use the same size/seed:** This is the case when users, e.g., create additional Bitcoin addresses and need to update their outsourced Bloom filters to include those addresses. In this case, $\mathcal{B}_1$ and $\mathcal{B}_2$ are likely to comprise of similar Bitcoin addresses. This includes both the actual elements of the filters, i.e., the Bitcoin addresses of the user, and the false positives generated by the Bloom filter. In this case, $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + P_f(2N_1)|\mathbb{B}| \tag{7.9}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(2N_1)|\mathbb{B}| - k} \tag{7.10}$$

Notice that $|\mathcal{B}_1 \cap \mathcal{B}_2| = S_1 + N_1 = |\mathcal{B}_1|$ (since $\mathcal{B}_1 \subset \mathcal{B}_2$); therefore, $P_{h_{(j)}}$ is not affected by the acquisition of the second filter $B_2$.

**$B_1$ and $B_2$ use different seeds:** In existing SPV clients, the random nonce $r$ used to instantiate the Bloom filter is stored in volatile memory. Therefore, each time the SPV client is restarted (e.g.,

smartphone reboots), a new filter will be created with a new seed chosen uniformly at random. If the adversary acquires two Bloom filters of the same user which are initialized with different seeds, then these filters are likely to exhibit different false positives. $B_1$ and $B_2$ will however comprise of a number of identical elements (which map to the Bitcoin addresses of the user). More specifically,

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + (|\mathcal{B}_1| - N_1)\frac{|\mathcal{B}_2|}{|\mathbb{B}| - N_1} \tag{7.11}$$

$$\approx N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| \tag{7.12}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| - k} \tag{7.13}$$

As we show in Section 7.5.2, the obtained $P_{h_{(j)}}$ is considerably large in this case, when compared to the case where the adversary has access to only one filter.

**$B_1$ and $B_2$ use the same seed, but have different sizes:** This is the case when users, e.g., create additional Bitcoin addresses beyond the capacity of their current Bloom filters. SPV clients therefore need to resize their Bloom filters. Note that filter resizing typically shuffles the bits of the Bloom filters; the resulting distribution of bits in the new resized filter is not necessarily pseudo-random (since the same seed is used) and depends on the sizes of the filters. As such, only the lower bound on $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be estimated using Equation 7.12 (which estimates the worst case where filter resizing causes a pseudo-random permutation of the bits of the filters); in Section 7.5.2, we confirm our analysis by means of experiments.

Recall that since there are only few tens of millions of addresses in Bitcoin, the adversary can brute-force search the entire list of Bitcoin addresses in order to acquire $\mathcal{B}_1$ and $\mathcal{B}_2$ and compute $\mathcal{B}_1 \cap \mathcal{B}_2$. Given any two Bloom filters $B_1$ and $B_2$, the adversary can easily guess whether these two Bloom filters contain Bitcoin addresses from the same wallet. Indeed, if $|\mathcal{B}_1 \cap \mathcal{B}_2|$ is small, then it is highly likely that $B_1$ and $B_2$ map to different elements (if $m_1$ and $m_2$ are not small), and therefore pertain to different users. On the other hand, when $|\mathcal{B}_1 \cap \mathcal{B}_2| \gg 0$, it is highly likely that all the Bitcoin addresses in the set $\mathcal{B}_1 \cap \mathcal{B}_2$ belong to the same SPV client.

**Multiple Bloom Filters** In the previous paragraphs, we discussed the case where the adversary is equipped with only two Bloom filters. Note that our analysis equally applies to the case where the adversary possesses any number $b > 2$ of Bloom filters pertaining to the same entity.

As mentioned earlier, by computing the intersection between each pair of filters, the adversary can find common elements to different filters; this also enables the adversary to guess with high confidence whether different filters have been generated by the same client. Given $b$ filters which belong to the same SPV client, the adversary can compute the number of elements inserted within each filter using Equation 7.4. In the sequel, we assume that filters $B_1, \ldots, B_b$ are sorted by increasing number of elements (i.e., $B_b$ contains the largest number of elements), and that filters are constructed using different seeds. Let $\mathcal{K}_j = \mathcal{B}_j \cap \cdots \cap \mathcal{B}_{(b-1)}, \forall j \in [1, b-1]$.

Note that $|\mathcal{K}_1| \leq |\mathcal{K}_2| \cdots \leq |\mathcal{K}_{(b-1)}|$. Here, the larger the number of Bloom filters at the disposal of the adversary, the smaller is the error of the adversary in correctly classifying the genuine addresses of the SPV client, and the larger is $P_{h_{(.)}}$. That is, the larger is $b$, the smaller are the number of common false positives that are exhibited by the different filters, and the higher is the confidence of the adversary in identifying the false positives of $B_j$. Following Equation 7.12,

$$E[|\mathcal{K}_1|] = \min(|\mathcal{B}_1|, |\mathcal{B}_2|, \ldots) \approx N_1 + |\mathbb{B}| \prod_{\forall j} P_f(m_j) \tag{7.14}$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_i - k}{N_i - k + |\mathbb{B}| \prod_{\forall j} P_f(m_j)} \tag{7.15}$$

Moreover, as $j$ increases, $\mathcal{K}_j$ will contain more false positives, and $P_{h_{(j)}}$ will decrease.

## 7.5.2 Experimental Evaluation

In what follows, we experimentally validate our analysis in Section 7.5.1. For that purpose, we rely on a similar setup to the one used in Section 7.4.2, and we perform four different experiments. We perform all four experiments by setting the target false positive rate $P_t$ to 0.05%, 0.1% and 0.5%, respectively.

| $N_i, N_j$ | $P_t$ (%) | $\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$ | $P_{h_{(1)}}$ $(b=2)$ | $P_{h_{(1)}}$ $(b=1)$ | $P_{h_{(N/2)}}$ $(b=2)$ | $P_{h_{(N)}}$ $(b=2)$ | $N_i, N_j$ | $P_t$ (%) | $\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$ | $P_{h_{(1)}}$ $(b=2)$ | $P_{h_{(1)}}$ $(b=1)$ | $P_{h_{(N/2)}}$ $(b=2)$ | $P_{h_{(N)}}$ $(b=2)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Experiment 1 (Same client, same seed, same size)** | | | | | | | **Experiment 2 (Same client, same seed, different size)** | | | | | | |
| 25,45 | 0.05 | 83.6(±10.88) | 0.2990 | 0.2910 | 0 | 0 | 70,270 | 0.05 | 70.3(±0.28) | 0.9957 | 0.0678 | 0.8145 | 0.2502 |
| 25,45 | 0.1 | 245.0(±15.36) | 0.1020 | 0.1070 | 0 | 0 | 70,270 | 0.1 | 71.70(±0.48) | 0.9762 | 0.0266 | 0.3177 | 0.0011 |
| 25,45 | 0.5 | 3192.90(±230.79) | 0.0078 | 0.0075 | 0 | 0 | 70,270 | 0.5 | 671.20(±46.54) | 0.1043 | 0.0031 | 0 | 0 |
| **Experiment 3 (Same client, different seed)** | | | | | | | | | | | | | |
| 10, 10 | 0.05 | 10.0(±0.0) | 1 | 0.9997 | 1 | 1 | 10, 10 | 0.5 | 10.0(±0.0) | 1 | 0.7448 | 1 1 | 1 |
| 100, 100 | 0.05 | 100.0(±0.0) | 1 | 0.1164 | 1 | 1 | 100, 100 | 0.5 | 110.90(±1.40) | 0.9017 | 0.0052 | 0.0009 | 0 |
| 1,000, 1,000 | 0.05 | 1004.70(±1.54) | 0.9953 | 0.0785 | 0.0390 | 0 | 1,000, 1,000 | 0.5 | 1499.70(±22.7) | 0.6668 | 0.0077 | 0 | 0 |
| 5,000, 5,000 | 0.05 | 5007.30(±1.96) | 0.9985 | 0.0003 | 0.0064 | 0 | 5,000, 5,000 | 0.5 | 5755(±15.05) | 0.8688 | 0.0308 | 0 | 0 |
| 10, 10 | 0.1 | 10.0(±0.0) | 1 | 0.9969 | 1 | 1 | 1,000, 1,000 | 0.1 | 1015.60(±2.37) | 0.9846 | 0.0395 | 0 | 0 |
| 100, 100 | 0.1 | 100.30(±0.28) | 0.9970 | 0.0464 | 0.8138 | 0.225 | 5,000, 5,000 | 0.1 | 5032.40(±3.15) | 0.9936 | 0.1376 | 0 | 0 |

**Table 7.3:** Measuring $|\mathcal{B}_i \cap \mathcal{B}_j|$ and $P_{h_{(.)}}$ in Experiments 1,2, and 3, using filters of the same SPV client with respect to $N$ and $P_t$. Here, $m_i \leq m_j$. Each data point is averaged over 10 independent runs; we also present the 95% confidence intervals.

In our first experiment (Experiment 1), we create 10 different user wallets, each generating five different Bloom filter $B_1$,$B_2$,...,$B_5$ with the same size and using the same random seed $r$, but each having a different number of elements $N = \{25, 30, 35, 40, 45\}$. This corresponds to the case where 10 different users constantly insert new Bitcoin addresses and update their outsourced Bloom filters. Here, we assume that our adversary captures all 50 Bloom filters and applies our analysis described in Section 7.5.1 to learn additional information about the user addresses. In this experiment, we compute $I_1 = \mathcal{B}_1 \cap \mathcal{B}_5$ for each SPV client; we then report the average intersection size for all 10 wallets in Table 7.3.

Our results in Table 7.3 show that Bloom filters pertaining to the same SPV client, and which share the same initial seed, are likely to exhibit the same false positives (in addition to the elements inserted in the Bloom filter). Indeed, our results show that $|I_1|$—measured experimentally—matches Equation 7.9, irrespective of the target $P_t$; moreover, in this case, $P_{h_{(.)}}$ obtained using $b = 2$ Bloom filters (cf. Equation 7.13) is similar to $P_{h_{(.)}}$ when $b = 1$ one Bloom filter.

In our second experiment (Experiment 2), we extend our evaluation to account for the case where Bloom filters originating from the same user have different sizes. Here, we also create 10 different user wallets, each generating five different Bloom filters $B_1$,$B_2$,...,$B_5$ using the same random seed, but each having a different number of elements (respectively $N = \{70, 120, 170, 220, 270\}$) and different sizes ($M$ ranges between 3224 and 9680 bits). Analogously to Experiment 1, we compute $I_2 = \mathcal{B}_1 \cap \mathcal{B}_5$ for each SPV client; we then report the average intersection size for all 10 wallets in Table 7.3. Additionally, we compute the intersection set $I_3$ shared by $B_1$ from the first wallet, with a randomly chosen Bloom filter from the remaining 9 wallets. In Table 7.5, we report the average intersection set size over the 9 wallets. Our results in Tables 7.3 and 7.5 confirm our aforementioned analysis. Indeed, $|I_2|$ matches the values derived using Equation 7.12 when the Bloom filters pertain to the same user. Otherwise, $|I_3|$ matches Equation 7.6.

In our third experiment (Experiment 3), we investigate the case where Bloom filters pertaining to the same user are initialized with different random seeds. As mentioned earlier, this, e.g., corresponds to the case when the user restarts the SPV client. In this experiment, we create 10 different user wallets, each generating 16 different Bloom filters constructed using different initial seeds as follows: 4 filters $B_1$,$B_2$,$B_3$,$B_4$

| $b$ | $P_{h_{(1)}}$ ($P_t = 0.05\%$) | | $P_{h_{(1)}}$ ($P_t = 0.1\%$) | | $P_{h_{(N/2)}}$ ($P_t = 0.05\%$) | | $P_{h_{(N/2)}}$ ($P_t = 0.1\%$) | | $P_{h_{(N)}}$ ($P_t = 0.05\%$) | | $P_{h_{(N)}}$ ($P_t = 0.1\%$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | (Analy.) | (Emp.) | (Analy.) | (Emp.) | (Analy.) | (Emp.) | (Analy.) | (Emp.) | (Analy.) | (Emp.) | (Analy.) | (Emp.) |
| 1 | 0.1713 | 0.1715 | 0.0926 | 0.0916 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0.9978 | 0.9977 | 0.9911 | 0.9908 | 0.0091 | 0.0079 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 7.4:** Experiment 4: $P_{h_{(.)}}$ w.r.t. the number $b$ of Bloom filters which pertain to same SPV client. Here, we assume that each filter is generated using a different seed.

with $N = 10$, $N = 100$, $N = 1,000$, and $N = 10,000$, respectively. For each wallet, and all filters of the same size, we compute $I_4 = \mathcal{B}_1 \cap \mathcal{B}_4$ and we report the average $|I_4|$ for each filter sizes in Table 7.3. Additionally, for each filter size, we also compute the intersection set $I_5$ using $B_1$ from the first wallet, with a randomly chosen Bloom filter from all other 9 wallets. In Table 7.5, we report the average intersection set size over the 9 wallets. Similar to Experiment 2, our results in Table 7.3 show that, irrespective of the filter size, and of the target false positive rate, $P_{h_{(.)}}$ significantly decreases when the adversary acquires $b = 2$ Bloom filters pertaining to the same SPV client.

Finally, in our final experiment (Experiment 4), we investigate the impact of having $b > 2$ Bloom filters pertaining to the same SPV client. For that purpose, we use 5 Bloom filters $B_1$,$B_2$,...,$B_5$ generated using different seeds with $N = \{3070, 3120, 3170, 3220, 3270\}$. We then compute $\mathcal{K}_j = \mathcal{B}_1 \cap \cdots \cap \mathcal{B}_{(j+1)}, \forall j \in [1, b-1]$, and the corresponding $P_{h_{(.)}}$ using Equation 7.15. Our findings are depicted in Table 7.4; our results validate our aforementioned analysis and show that the larger the number $b$ of acquired Bloom filters of the same SPV client, the larger is $P_{h_{(.)}}$, and the smaller is the privacy of the user's addresses.

**Summary**

- In Experiments 1,2, and 3, notice that $|I_3| \approx |I_5| \ll \frac{\min(m1,m2)}{2}$. This means that given any two Bloom filters $B_1$ and $B_2$, if $|B_1 \cap B_2| \ll \frac{\min(m1,m2)}{2}$ (estimated using Equation 7.1), then an adversary can easily tell whether any two Bloom filters pertain to the same SPV client.

- If the two Bloom filters acquired by the adversary belong to the same SPV client, the adversary can identify whether the SPV

| $N_i$ | $P_t$ (%) | $E[\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert]$ (Analytical) | $\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$ (Empirical) |
|---|---|---|---|
| **Experiment 2 (Different client, different seed, different size)** | | | |
| - | 0.05 | - | 0.0(±0) |
| - | 0.1 | - | 1.22(±0.60) |
| - | 0.5 | - | 31.60(±6.34) |
| **Experiment 3 (Different client, different seed, same size)** | | | |
| 10 | 0.05 | 0 | 0.0(±0.0) |
| 10 | 0.1 | 0 | 0.0(±0.0) |
| 10 | 0.5 | 0 | 0.0(±0.0) |
| 100 | 0.05 | 0 | 0.0(±0.0) |
| 100 | 0.1 | 0.13 | 0.0(±0.0) |
| 100 | 0.5 | 11.27 | 13.89(±1.64) |
| 1,000 | 0.05 | 4.21 | 5.0(±0.97) |
| 1,000 | 0.1 | 18.06 | 21.11(±2.35) |
| 1,000 | 0.5 | 523.20 | 512.89(±23.21) |
| 5,000 | 0.05 | 7.37 | 11.89(±1.73) |
| 5,000 | 0.1 | 30.01 | 40.11(±3.40) |
| 5,000 | 0.5 | 753.41 | 789.78(±18.92) |

**Table 7.5:** Measuring $|\mathcal{B}_1 \cap \mathcal{B}_2|$ in Experiments 1,2, and 3, using filters pertaining to different SPV clients with respect to $P_t$. Each data point is averaged over 10 independent runs.

client has restarted while generating his Bloom filters; here, notice that $\frac{\min(m1,m2)}{2} \leq |I_4| \ll |I_2| \ll |I_1|$.

- $P_{h_{(.)}}$ corresponding to $b = 2$ filters is considerably larger when compared to the case where the adversary has access to one Bloom filter. This means that an adversary which can acquire more than one Bloom filter pertaining to an SPV client can learn considerable information about the addresses of the node—irrespective of the size of the Bloom filter and $P_t$. In this case, our results show that $P_{h_{(N)}}$ approaches 1, which signals full leakage of the addresses of the SPV client. $P_{h_{(.)}}$ increases to 1 as the number $b$ of Bloom filters of the same SPV client captured by the adversary increases.

## 7.6 Our Proposed Solution

Given our findings, we propose in what follows a solution that enhances the privacy of SPV clients which rely on Bloom filters. Before

presenting our solution, we briefly summarize our observations from Sections 7.4 and 7.5.

**Observation 1:** The number of elements inserted within a Bloom filter significantly affects the resulting false positive rate of the filter. This is especially true when the filter's size is modest (e.g., $< 500$). Indeed, the number of elements inserted in the filter should match at all times the filter's size in order to achieve the target false positive rate (i.e., $P_f(m) = P_t$).

**Observation 2:** The acquisition of multiple Bloom filters considerably reduces the privacy of SPV clients. The construction of multiple Bloom filters per SPV client should be avoided. Otherwise, different Bloom filters should be constructed with different initial seeds, and should contain different elements. In this way, an adversary does not gain considerable advantage when acquiring two or more Bloom filters.

**Observation 3:** SPV clients should keep state about their outsourced Bloom filters (i.e., on persistent storage) to avoid the need to recompute a filter which contains the same elements using different parameters.

**Observation 4:** As mentioned earlier, inserting both the public key and the public key hash (the address) in the Bloom filter provides a sufficient distinguisher for the adversary in guessing whether an address is a true positive or not. Note that for the most common transaction type *Pubkey Hash* (P2PKH), inserting the hash of the public key is sufficient. However, there might be other transaction types where it is beneficial to also store the public key in the Bloom filter. In this case, the client can insert either a Bitcoin address or its corresponding public key (but not both) in the same Bloom filter. Indeed, sample experimental results that we conducted show that for almost 99% of all addresses in the network, it suffices to insert either the public key or the public key hash within the same Bloom filter in order to receive all the relevant transactions destined to the address.

**Our Solution** In what follows, we describe a solution which leverages Observations 1, 2, 3, and 4. In our solution, each SPV client generates

$N$ Bitcoin addresses at the start, and embeds them in a Bloom filter which can fit $M = m = N$. Here, the Bloom filter is constructed with a realistic target false positive rate $P_t$, which combined with $N$ and $M$, results in a target privacy level (cf. Equation 7.3). Note that since $M = m$, then we ensure that the Bloom filter's false positive rate matches $P_t$ (cf. Section 7.4). Here, we assume that only the address is inserted within each filter.

Clearly, since the user might not directly use all $N$ addresses, some of his Bitcoin addresses will not be revealed and will remain in his wallets. Whenever users run out of their $N$ addresses and need to get additional addresses, they repeat the aforementioned setup process. That is, users create an additional set of $N$ addresses and embed them in a new Bloom filter—constructed with a new initial seed— with $M = m = N$, and using the previously chosen $P_t$. Here, the advantage of an adversary which captures one or more Bloom filters pertaining to the same SPV client is negligible, since these filters do not have any element in common.

Additionally, our solution requires the SPV clients to keep state, e.g., about each Bloom filter, to avoid the need of re-computation of the same filter if the client restarts at any point in time. We point out that the required storage overhead to maintain information about the state of each Bloom filter is negligible. Indeed, to keep all the necessary state to reconstruct a Bloom filter, an SPV client needs to locally store: *(i)* the number of addresses embedded in the filer (4 bytes), *(ii)* the used target false positive rate $P_t$ (8 bytes), *(iii)* the used seed value (8 bytes), *(iv)* the *BloomUpdate* flag (2 bytes), and *(v)* the addresses inserted in the filter. The SPV client can add a pointer in the *ECKey* class of Bitcoinj in order to link each Bitcoin address to the Bloom filter that embeds it; the size of the pointer is roughly 2 bytes per address. This amounts to a total storage of $2N + 20$ bytes per Bloom filter. For example, when $N = 100$, the SPV client needs to store an additional 220 bytes per Bloom filter in order to reconstruct the filter upon restart; clearly, this overhead can be easily tolerated in existing SPV client implementations.

Clearly, our proposed solution can be directly integrated within existing SPV clients, and only incurs in small modifications to existing client implementations. Moreover, our solution does not incur additional overhead on the SPV clients—apart from the pre-generation of $N$ Bitcoin addresses (which is only done at setup time), and the storage space required for each generated Bloom filter. Note that our solution

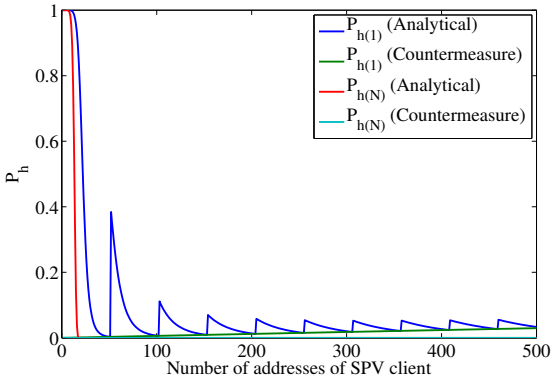



**Figure 7.6:** Evaluation of our countermeasure. Here, we measure $P_{h_{(1)}}$ and $P_{h_{(N)}}$ given $P_t = 0.05$ and assuming the current SPV client implementation. In computing $P_{h_{(1)}}$, we assume that the SPV client did not restart since initialization.

requires SPV clients to outsource all their filters to full Bitcoin nodes since different filters embed different sets of their addresses. Therefore, $N$ should be carefully chosen. If a user needs many addresses, choosing a larger $N$ would avoid the generation of a larger number of Bloom filters. We leave the task of finding the best parameters when constructing a Bloom filter as an interesting direction for future work. Since users typically have few hundred Bitcoin addresses, we argue that this overhead can be largely tolerated given the current usage patterns in Bitcoin. Notice that the larger is $N$, the smaller are the number of Bloom filters created by the SPV client. Note that, in our solution, an SPV client should not outsource more than one Bloom filter to each regular node that it connects to. Indeed, if the client outsources several filters to the same node, then the regular node can correlate these filters (even if they do not embed the same elements).

In Figure 7.6, we analytically compute $P_h$ given our countermeasure. Clearly, our countermeasure considerably decreases $P_h$, especially for clients that have a small number of addresses.

**Additional Insights** We point out that an adversary which has access to the Bitcoin blockchain can observe that Bitcoin addresses which recently were used by an SPV client do match the Bloom filter of that

client. This knowledge clearly reduces the privacy of newly used addresses. Here, we stress that every newly used address of an SPV client is likely to match the Bloom filters of a fraction $P_t$ of all other SPV clients. In our solution, $P_t$ therefore defines the minimum anonymity set size of each address.

One alternative would be to embed existing Bitcoin addresses that do not belong to the node, when constructing each Bloom filter. Subsequently, whenever an SPV client needs to generate a new Bitcoin address, it will choose a Bitcoin address which matches its existing Bloom filter[7]. However, this alternative also results in an anonymity set defined by $P_t$; it also comes at the expense of computational overhead incurred on the SPV clients. In an experiment that we conducted using a Bloom filter $B_t$ of size 1608 bits with $P_t = 0.05\%$ and $m = 102$, we were able to generate three new Bitcoin addresses which match $B_t$; these addresses were found after 240,351, 415,877, and 5,767,346 tries, respectively.

Moreover, we point out that our analysis and solution do not address the case where the adversary can link addresses by using side-channel information from the Bitcoin blockchain, namely:

**Filtering false positives by date:** SPV clients only appeared in the second half of 2011, which is 1.5 years after Bitcoin started. Therefore, if a Bitcoin address which was created before 2011 matches the Bloom filter of an SPV client, the adversary can label it, with high confidence, to be a false positive, and not a Bitcoin address of the node. However, we note that the number of addresses created from 2011 exceeds by far that corresponding to the period from 2009-2011, during which Bitcoin was still not widely used at the time.

**Clustering Bitcoin addresses:** The adversary can make use of techniques such as [1,78,104] to link/cluster certain Bitcoin addresses based on user behavior, transaction amounts, etc., and assess whether an address matching a Bloom filter is a false positive or a true positive. Here, we point out that our proposed solution can be used in conjunction with existing solutions such as [46,109] to prevent the linking/clustering of addresses using such techniques.

[7] That can be achieved, e.g., by constantly generating a new Bitcoin address until it matches its existing Bloom filter.

## 7.7 Conclusion

In this chapter, we explored the privacy provisions due to the integration of Bloom filters in SPV clients. Our results show that Bloom filters incur serious privacy leakage in existing SPV client implementations. More specifically, we show that a considerable number of the addresses of users of SPV clients which possess a modest number of Bitcoin addresses (e.g., $< 20$) are leaked by a single Bloom filter. Moreover, we show that a considerable number of the addresses of users is leaked if the adversary can collect two different Bloom filters issued by the same node, irrespective of the target false positive rate of the filter, and of the number of addresses owned by the user. Finally, we propose a lightweight solution that enhances the privacy offered by Bloom filters; our proposal can be integrated within existing SPV client implementations with minimum modifications.

# Chapter 8

# Quantifying Privacy of Transaction Prices

## 8.1 Introduction

Making data publicly available creates unexpected privacy risks. Recent examples include AOL's release of users' search keywords [120], which has led to the identification of users and their profiles [11]. Data released by Netflix was de-anonymized by leveraging IMDB and dates of user ratings [113], showing that the release of data cannot be analyzed in isolation. The privacy risks of combining different public records have led to several [131] de-anonymization attacks. Recent studies of anonymized mobility data showed that mobility traces can be de-anonymized by leveraging a few observations [74]. One source of consumer information involves their spending patterns. To date however, it was unclear to what extent consumer prices leak information about the respective purchase.

Consumer purchase histories are typically recorded by store chains with loyalty programs and are used to compute consumer spending profiles [25]. Banks, payment card issuers, and point-of-sale system providers collect this data at different levels of granularity. In a number of scenarios, it might be desirable to share this data within different departments of a company, across companies, or with the public [27]. Before disclosure, the data is sanitized so that it does not leak sensitive data, such as personally identifiable information and that it (partially

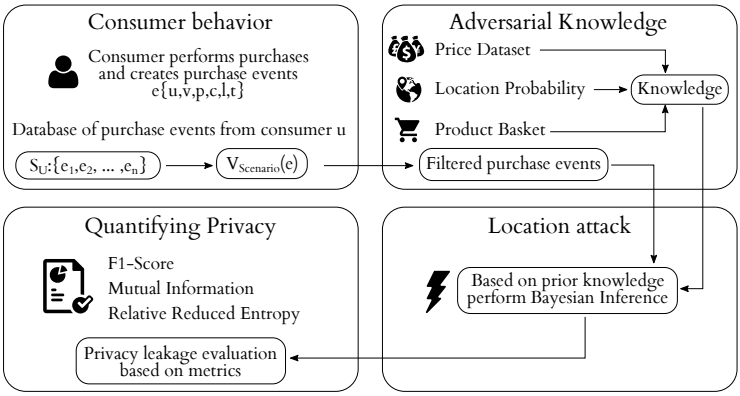



**Figure 8.1:** Framework overview for quantifying location privacy leakage from consumer price datasets.

or fully) hides location information. In new digital currency systems such as Bitcoin [125] and Ripple [30], transaction values are stored on a public ledger. Irrespective of whether transaction values are made available so that a system can fulfill its functions or are being disclosed for research purposes, it is important to understand the privacy implications of such disclosures.

In this chapter we focus on quantifying location disclosure resulting from the release of prices from consumer's purchase histories. Intuitively, the price distribution for a product differs from country to country, which allows us to identify possible purchase locations. We focus on consumer products which are generally inexpensive ($\leq$ 25 USD) and frequently-bought. More precisely, based on global prices (leveraging the Numbeo dataset [29]), we show that given access to a few consumer prices (and even without the product categories, precise times of purchase or currency), an adversary can determine the country in which the purchase occurred. Similarly, given the country, the city can be determined and within a city (leveraging the Chicago dataset [31]), the *local store* can be identified. We further demonstrate that it is possible to distinguish purchases among store chains (leveraging the Kaggle dataset [27]).

We present a generic framework (cf. Figure 8.1) that allows the modeling and quantitative evaluation of location leakage from consumer price datasets. In our framework we model the adversarial knowledge,

composed of a public dataset of consumer prices and location-specific information. We assume that the adversary has access to the individual product prices of a purchase (similar to the Kaggle dataset) and a coarse-grained value of the purchase time. In order to make the framework more flexible, our model supports different prior knowledge scenarios, e.g., the adversary additionally has access to the merchant category (e.g., knowledge that the product was bought in a market or a restaurant) or the product category (e.g., apples). Furthermore, we model the adversarial attack by detailing the corresponding probability functions. In particular, we point out how the adversary leverages multiple product prices in order to increase the probability of identifying the correct location.

Within our framework, we quantify the location privacy of consumer purchases in relation to different dimensions. For example, we measure how well the adversary estimates the location probability of the purchases with the $F_1$-score [128], capturing the test's accuracy. Furthermore, we use mutual information [70] to quantify the absolute location privacy loss of consumers, based on the considered price dataset. In addition, we capture the relative privacy loss by measuring the reduction in entropy. The proposed metrics are independent of the choice of adversarial strategy and therefore allow us to quantitatively measure the privacy loss induced from any price dataset known to the adversary.

We apply our framework to three real-world datasets: (i) the *Numbeo dataset* [29] contains, after outlier filtering, crowd-sourced real-world consumer prices from 112 countries and 23 US cities for 23 distinct product categories; (ii) the *Chicago dataset* [31] contains 24 million prices for 28 product categories capturing on average of 6304 products sold in Dominick's stores within the Chicago metropolitan area; finally, (iii) the *Kaggle dataset* [27] contains 350 million purchases from 311,541 consumer across 134 store chains.

Our evaluation shows that in order to infer the country based on a vector of purchases, an adversary often needs to observe less than 30 prices. Similarly, after having identified the country of the purchases and given roughly 30 prices, we show that we can reliably predict among 23 major cities within the United States. Finally, when the adversary narrowed down the coarse location, such as the Chicago metropolitan area, we show that based on a regional price dataset, and given a vector of purchases, an adversary can distinguish with high confidence *among local stores* using 100 purchases. For comparison, a weaker adversary with access only to coarse-grained time, i.e., the day of the purchase

and price information, requires 50 purchases to identify the country. Furthermore, to establish practical utility of our methodology, we evaluate it on a dataset of purchase records (Kaggle [27]) and show that an adversary requires approximately 250 purchases to distinguish with high confidence among 134 store chains.

The main contributions of this chapter are as follows:

- We propose a generic quantitative framework for evaluating attacks against the location privacy of consumer purchases. We validate our framework on three independent price datasets of real-world consumer prices and show that location information can be extracted reliably.

- We introduce three privacy metrics to capture the performance of the adversary in the attack as well as the extent to which location privacy of consumers is reduced when the adversary has access to a specific dataset of purchases.

To the best of our knowledge, this is the first work to infer the location of a purchase based on the price value in consumer purchases. The remainder of this chapter is organized as follows. In Section 8.2, we model purchase history and describe the adversarial model. In Section 8.3, we present the datasets selected for our evaluation in Section 8.4. We conclude the chapter in Section 8.5.

## 8.2 Model

In this section we introduce our system and adversarial model. We present the privacy metrics that quantify the probability of location disclosure based on the assumption that the adversary has access to a part of a consumer's purchase history.

### 8.2.1 System Model

A consumer interacts with merchants and performs purchases of one or more products. This interaction leaves a trace of purchase activity as a sequence of *purchase events*. We model each of the consumer's purchase events together with their contextual information as $e$: {consumer $u$, value $v$, product $p$, product category $c$, location $l$, time $t$}, where $v$ is the price value spent on product $p$ of product category $c$ at

location $l$ and time $t$. In our model, one purchase event is limited to one product, similar to the data contained in the Kaggle dataset. In addition, the price value is given in a global currency, which usually is different from the local currency of the purchase (e.g., the original price is SEK, but recorded in USD). The trace of purchases performed by the target consumer $U$, given as a series of purchase events, is denoted by $S_U$:$\{e_1, e_2, \ldots, e_n\}$. We define the following functions to represent the adversarial knowledge:

**Location Probability:** It describes the prior probability of a purchase event taking place in a specific location, e.g., $P(\text{USA})$ is the prior probability with which a random purchase event $e$ has $e.l = \text{USA}$. We define $\mathbb{L}$ as the set of all considered locations.

**Category Probability** Given location $l$, $P(c \mid l)$ describes the conditional probability of a purchase event to belong to a certain product category, e.g., $P(\text{Milk} \mid \text{USA})$ is the conditional probability with which a random event $e$ from the USA has $e.c = \text{milk}$. This conditional probability models the product category preferences in a location. We define $\mathbb{C}$ as the set of all considered product categories.

**Value Probability:** Given location $l$ and product category $c$, $P(v \mid l, c)$ describes the conditional probability of a purchase event at a given price value. It models the price distributions for different product categories in different locations, e.g., $P(1.5 \mid \text{USA}, \text{Milk})$ is the conditional probability with which milk can be bought in the USA for 1.5 worth of a global currency.

The adversary can now model the spending behavior and identify likely candidate locations. Specifically, the adversary computes the posterior probability that a single price value $v$ for a product category $c$ originated from a location $l$. The computation involves the prior and the conditional probabilities described above and the application of Bayes' theorem:

$$P(l \mid c, v) = \frac{P(l) \cdot P(c, v \mid l)}{P(c, v)} \tag{8.1}$$

In order to infer the location without knowing the product category, the adversary computes the probability that a price value $v$ originates

from location $l$:

$$P(l \mid v) = \frac{P(l) \cdot P(v \mid l)}{P(v)} \tag{8.2}$$

### 8.2.2 Adversarial Model

The adversary's goal is to identify the location of the events in $S_U$. In this section we present two different adversaries: (1) an adversary with complete knowledge and (2) an adversary with only public knowledge.

#### Adversary with Complete Knowledge

The ideal adversary represents a strong adversary with complete access to global purchase events. In particular, the adversary has access to the following prior knowledge:

**Global Purchase History:** The complete series of purchase events in the history of global purchases[1], denoted by $\mathcal{H}_G$. The adversary computes the posterior probability of a location based on $\mathcal{H}_G$.

**History for Target Consumer:** The adversary might have access to prior information about the target consumer's purchase history, denoted by $\mathcal{H}_U$. This could help the adversary to optimize the model for the target consumer[2].

Based on this knowledge, the ideal adversary computes the probabilities in Equations 8.1 and 8.2.[3]

#### Adversary with Public Knowledge

Our second adversarial model is a more realistic one, where the adversary only makes use of public information.

**Population:** Given the population at each location, the adversary estimates the location probability $P(l)$.

**Product Basket:** A product basket indicates which products an average consumer purchases during a year, both in terms of quantity

[1] The area of the attacker's interest can be restricted, e.g., when the adversary knows that its victim is somewhere in that restricted area.

[2] For example, by only considering the locations of previous purchases.

[3] The intermediate steps are given in the appendix A.

and monetary amount. We leverage the product basket in order to estimate the probability of a product category given the location ($P(c \mid l)$)[4].

**Price Dataset:** For each location and product category combination, a price value distribution $D$ is available, e.g., the Numbeo or the Chicago dataset. The adversary can use the distribution to estimate $P(v \mid l, c)$. We define $D(l, c, v)$ as the number of occurrences of price value $v$ for product category $c$ in location $l$ and $D(l, c)$ as the number of price values for product category $c$ and location $l$.

Since $D$ might be imperfect, the adversary can have incomplete or incorrect knowledge about the price value probabilities (i.e. unknown or rounded product prices). In this case the adversary should perform additive smoothing, which assigns a small probability $\alpha$ to each event [106]. On the contrary, if the adversary has or assumes complete knowledge of the price value probabilities, additive smoothing is not required.

The adversary with public knowledge computes the following probabilities:

$$P(l) = \frac{\text{Population}(l)}{\sum_{l' \in \mathbb{L}} \text{Population}(l')} \tag{8.3}$$

$$P(c \mid l) = \frac{\text{Basket}(l, c)}{\sum_{c' \in \mathbb{C}} \text{Basket}(l, c')} \tag{8.4}$$

$$P(v \mid l, c) = \frac{D(l, c, v) + \alpha}{D(l, c) + \alpha \cdot |S_U|} \tag{8.5}$$

In order to compute the probabilities defined earlier in Equations 8.1 and 8.2, the adversary requires access to either $P(l \mid c, v)$ or $P(l \mid v)$. Next, we describe how the adversary computes these probabilities and we define the adversary's knowledge.

### 8.2.3 Knowledge Scenarios

As mentioned, the adversary's objective is to identify the location of the events in $S_U$. The adversary is given a finite set of events $S_U$ on which

[4]We currently use a single product basket for all locations.

the attack is executed—the adversary is not allowed to choose or request new purchase events $e$. We consider an adversary with public knowledge and distinguish among three distinct adversarial knowledge scenarios, each consisting of a subset of the public knowledge. Depending on the knowledge scenario, the adversary might not have access to all information from a purchase event $e$. Therefore, we define a family of functions $V_{\text{scenario}}(e) = V(e)$ that filter, depending on the given scenario, the public knowledge accessible to the adversary.

**Price:** This scenario corresponds to an adversary that has access to multiple purchase events $e$, *only* the corresponding *price value* and a notion of the purchase time $e.t$. The adversary is not aware of the product $e.p$ or the product category $e.c$. The precision of the purchase time depends on further specifications of the scenario. More formally, $V_{\text{price}}(e) = \{e.v, e.t\}$. Given the public knowledge modeled by Equations 8.3, 8.4 and 8.5, the adversary computes the posterior probability $P(l \mid v)$ of a price value $v$ from location $l$. The intermediate steps for computing $P(v \mid l)$ and $P(v)$ are detailed in the appendix A in Equations 8.12 and 8.10.

**Price_Merchant:** Similar to the former knowledge scenario, the adversary here has access to $S_U$, a series of multiple purchase events. In this scenario, however, the adversary knows the price value $e.v$ of the event as well as which merchant category $m$ sold the product. Formally, for each purchase event $e$, $V_{\text{price\_merchant}}(e) = \{e.v, e.t, m\}$, where $V_{\text{price\_merchant}}$ requires a function $M(e) = m$. We consider three merchant categories: restaurant, market and local transportation. The $V_{\text{price\_merchant}}(e)$ function estimates the merchant category $m$ from the product category $e.c$ of the respective event[5]. Analogously, using Equation 8.1, the adversary computes the probability of a location, based on the merchant and the price value:

$$P(l \mid m, v) = \frac{P(l) \cdot P(m, v \mid l)}{P(m, v)} \tag{8.6}$$

where $P(m, v \mid l)$ is computed as follows:

$$P(m, v \mid l) = \sum_{c \in M^{-1}(m)} P(c, v \mid l) \tag{8.7}$$

[5]In the following we refer to the merchant category as merchant.

**Price_Product-Category:** This scenario corresponds to the most knowledgeable adversary with public knowledge. Similarly to the former scenarios, the adversary receives multiple purchase events $S_U$. In addition, the adversary has access to the product category $e.c$ as well as the price value $e.v$. Note that $e.c$ implicitly assumes knowledge of the merchant, resulting in more formally $V_{\text{price\_product-category}}(e) = \{e.v, e.t, e.c\}$.

Given the public knowledge described in Section 8.2.2, the adversary computes the probability $P(l \mid c, v)$ of a purchase event with product category $c$ and price value $v$ originating in location $l$. The intermediate steps for computing $P(c, v \mid l)$ and $P(c, v)$ are detailed in the appendix in Equations 8.13 and 8.11.

In the following section we provide an intuitive perspective on the probabilities $P(l \mid v)$ and $P(l \mid c, v)$.

### 8.2.4 Conditional probability intuition

$P(l \mid v)$ is the probability of a location, given a price value in a purchase event. An example plot based on our evaluation can be found in Figure 8.2. We have chosen the purchase event $e$ with a price value of $e.v = 1$ Euro and estimated the location of the price. The figure shows that the most likely location for 1 Euro is France, closely followed by Germany, Italy and Spain. The plot also shows $P(l \mid c, v)$ for a purchase event with $e.v = 1$ Euro and the product category is milk. The most likely country is again France, followed by Germany and Italy. Surprisingly, China ranks as $5^{th}$. This can be explained by the fact that (i) some prices from China in the dataset were erroneously reported in Euros and (ii) that the location probability $P(l)$ influences the overall outcome, and, since China's population is considerable, there is an increased probability of purchases occurring there. Overall we observed that the probability distribution changes when the product category is known, i.e., France is more likely to have a 1 Euro price for milk, than a 1 Euro price in general.

### 8.2.5 Multiple purchase events

Up to this point, the analysis has been based on a single purchase event. To naturally combine multiple purchase events, we assume that the purchase events are conditionally independent, given the location $l$.

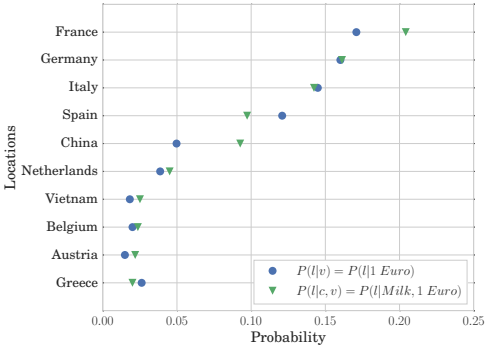



**Figure 8.2:** Probability distribution of $P(l \mid v)$ and $P(l \mid c, v)$, given 1 Euro and milk.

Therefore, the probability of a location $l$, given a set of purchase events $S_U$, is calculated as follows:

$$\begin{aligned} P(l \mid S_U) &= P(l \mid V(e_1), V(e_2), \ldots, V(e_n)) \\ &= \frac{P(l) \cdot \prod_{e \in S_U} P(V(e) \mid l)}{P(V(e_1), \ldots, V(e_n))} \end{aligned} \tag{8.8}$$

The intermediate steps for computing $P(l \mid S_U)$ can be found in the appendix in Equation 8.18. We experimentally verified the conditional independence of $V(e)$ given $l$ for the three knowledge scenarios and therefore Equation 8.8 applies equally to the different adversarial knowledge scenarios. Note that we effectively weaken the adversary by considering the products of different purchases independent from each other.

### 8.2.6 Privacy Metrics

We introduce three privacy metrics in order to capture the privacy of consumers revealing their purchase histories across different dimensions: We (i) measure the performance of the adversary in identifying the true location with the $F_1$-score. Then, (ii) using the notion of mutual information [70], we quantify the absolute privacy loss of the

consumer due to the adversary's knowledge of a price dataset. Finally, (iii) we use the relative reduced entropy as a relative privacy metric[6].

**$F_1$-score:** The objective of the adversary is to assign the purchase events to the correct location. In the worst case, the adversary is forced to randomly guess among all possible locations. If the adversary, however, can estimate location probabilities more accurately, location privacy is reduced. Our problem corresponds to a multi-class classification problem and we therefore quantify the adversarial performance by averaging the $F_1$-score [128] of each individual class. The $F_1$-score corresponds to the harmonic mean of recall and precision, measuring the test's accuracy.

**Mutual Information:** A purchase event dataset enables the adversary to infer the distribution of prices among locations. Therefore, we want to measure how much privacy consumers lose when their purchase events are revealed and when the adversary has access to a dataset of purchase events. We quantify this privacy objective by measuring the *absolute* reduced location entropy given the purchase events. To this extent, we use the Mutual Information [70], denoted by $I(l, V(e))$, which measures how much the entropy of the locations is reduced given the purchase events (cf. Equation 8.9).

$$I(l, V(e)) = \sum_{l \in \mathbb{L}, e \in S_U} P(l, V(e)) \cdot \log_2 \frac{P(l, V(e))}{P(l)P(V(e))} \tag{8.9}$$

**Relative Reduced Entropy:** Recall that the mutual information quantifies what we call the absolute privacy loss. In fact, there is an inherent randomness in the price distribution among locations. It is important to capture to what extent the original uncertainty about the locations can be reduced when a dataset of purchase events is given. The relative reduced entropy therefore captures the *relative* privacy, as the complement of the fraction of the conditional entropy over the location entropy. Given $H(l) = I(l, V(e)) + H(l \mid V(e))$, we compute the relative reduced entropy as $1 - \frac{H(l|V(e))}{H(l)}$ over all purchase events.

The proposed evaluation metrics are independent of a particular adversarial strategy. In return, the output of the privacy leakage quantification only depends upon the employed dataset of purchase events. In the next section we present the datasets utilized for our experimental evaluation.

[6]Defined as the complement of the fraction of conditional entropy over the location entropy.

## 8.3 Datasets

There are only a couple of datasets accurately accumulating the worldwide product price information. For individual products (e.g., a Big Mac [19] or Starbucks coffee [28]), the average price values per country are available. Because a product often appears multiple times with different price values in the same country or city, the average is not a good estimator for elaborate studies. In the following, we describe the three independent price datasets considered in our work.

The first dataset, Numbeo [29], is a crowd-sourced dataset containing worldwide price values per product category, city and country. It is the most complete dataset of worldwide harvested prices available to our knowledge. We restricted our analysis to 23 frequently bought product categories, and split the Numbeo dataset into two separate datasets: (i) two years of data as the Numbeo dataset and (ii) five months of data as the Numbeo test dataset (cf. Table 8.3). Numbeo performs sanity checks on the crowdsourced inputs, and we additionally filtered extreme outlier [16][7] from the data to account for possible mistakes from crowdsourced data. We identified 112 countries, with a total of 328,720 price values. Note that the provided data mostly contains prices from the US (18%) and India (14%).

The second dataset, referred to as the Chicago dataset [31], covers 84 stores in the Chicago metropolitan area over a period of five years. The data is sourced on a weekly basis from Dominick's supermarket stores. We sample 85 weeks with the most data, each containing on average 283,181 prices, spanning 28 product categories for an average of 6304 different products.

The third dataset originates from Kaggle [27], a Machine Learning competition platform. The dataset contains 350 million purchase events from 311,539 consumers across 134 store chains. The data is anonymized, but contains the individual product price, product category, date of purchase and purchase amount. Most purchase events cost less than 25 USD. The country of the dataset is not disclosed, but purchase prices are given in USD and purchase amounts are described in the imperial system.

In order to estimate the location probability, an adversary requires the knowledge of the population in each location. On the country

[7] $price < 25^{th}$ percentile $- 3 \cdot$ interquartile range, and
$price > 75^{th}$ percentile $+ 3 \cdot$ interquartile range

granularity, we use the data available from the World Bank [34] for the year 2013, while for the US city granularity we used the data from the US Census Bureau [134].

As described in Section 8.2.2, we increase the knowledge of the adversary with the product basket. A product basket details which and how many products an average person purchases, both in terms of quantity and monetary amount. We leverage a national product basket [33] from 2010 containing over 300 product categories in order to infer the ratio in which different products are bought over the year.

## 8.4 Experimental Evaluation

In this section we evaluate the adversarial models designed in Section 8.2.2. We start by presenting the assumptions and choices made for the evaluation.

### 8.4.1 Experimental Considerations

With respect to the value probability $P(v \mid l, c)$, we assume that the frequency of price values in the Numbeo dataset reflects the frequency of real-world purchase events with the corresponding price values. This is a natural assumption and is further motivated by the fact that e.g., Numbeo contributors likely entered the most popular price values for the considered product categories. Because our datasets contain a limited amount of products and product categories, our analysis is naturally confined to the available products. Note that, if the adversary knows the product categories of the purchases, e.g. milk, other categories such as apples can be ignored, which allows precise predictions with knowledge about few products. In order to compute the product category probability, $P(c \mid l)$, we only consider one national product basket and apply it to every country. Note that we do not use the product basket as an indicator of how much money is spent on average by a person, but rather as an indicator in which ratio products are bought.

**Sampling Price Values:** Given a location $l$, we generate *synthetic* consumer purchase events by sampling price values from the respective dataset. For the three datasets we consider adversaries with complete knowledge of the price values. In addition we instantiate an adversary with incomplete knowledge with the Numbeo test dataset. Given the product basket of the location $l$ we compute the probability of a prod-

uct category being sampled (cf. Equation 8.4). Thus, we sample each product category with the product category probability $P(c \mid l)$. For each location we repeat the sampling of the price values $n = 1000$ times and average the result.

**Additive smoothing parameter:** In the case of an adversary with incomplete knowledge, we make use of additive smoothing to avoid zero probabilities when aggregating the probabilities of multiple purchase events for locations (see Section 8.2.2). We choose a smoothing parameter $\alpha = 0.01$ which provides us with the best results on our data (cf. appendix Figure 8.6).

In the following, we evaluate up to three knowledge scenarios (cf. Section 8.2.3) for four location granularities: (i) across 112 countries worldwide; (ii) across 23 cities within the United States; (iii) across 84 stores within the Chicago metropolitan area; (iv) we distinguish among 134 store chains in a country.

### 8.4.2 Country Granularity

The adversary has to distinguish 112 candidate countries for each purchase event. We quantify the privacy given the three privacy metrics defined in Section 8.2.6. In particular, we performed our study in two settings. First, (i) we assumed that the adversary does not have complete knowledge. This means that the adversary receives purchase events from the Numbeo test dataset and estimate their location based on the Numbeo dataset. In the second case, (ii) the adversary assumes complete knowledge of price values, and therefore, the sampled prices are included in the price dataset which is adversarial knowledge.

Figure 8.3 shows the $F_1$-score for the first case based on the number of purchase events accessible to the adversary. Given one purchase event, the price, price_merchant and price_product-category knowledge scenario achieve an average of 0.38, 0.41 and 0.49 respectively. The high $F_1$-score after one purchase event shows, that even one event allows a decent prediction. We observe that the adversary is more likely to identify the correct location when it knows the product category of the purchase event. On the contrary, if the adversary has access to 10 purchase events, the respective $F_1$-scores are 0.80, 0.85 and 0.90. In other words, 10 purchase events significantly improve the ability of the adversary to identify the location of the purchase events. The reported values are averaged over $n = 1000$ iterations.

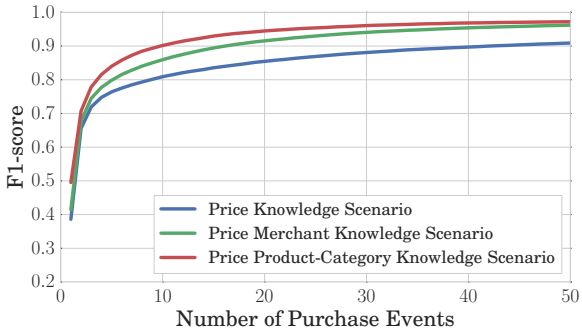



**Figure 8.3:** $F_1$-score for identifying the country given purchase events *sampled from the Numbeo test dataset*, corresponding to incomplete knowledge. We are not overfitting as we successfully classify new prices based on previously known prices.

Figure 8.4 corresponds to the second case, where the adversary assumes complete knowledge of the price values. We observe that the adversary can distinguish more accurately between the possible locations. The $F_1$-scores are averaged over all considered countries. For each considered country in the price knowledge scenario, we verify that averaging does not hide poorly performing countries (cf. Figure 8.7 in the appendix).

Table 8.1 presents the results of the mutual information and the relative reduced entropy for each knowledge scenario. We observe that the price_product-category knowledge scenario reduces the entropy more significantly than the other knowledge scenarios. Naturally, this is because the price_product-category knowledge scenario provides the adversary with more information than the price knowledge scenario, thus effectively reducing uncertainty when identifying the location.

### 8.4.3 US City Granularity

In this section we analyze an adversary that aims to distinguish among the purchase events of 23 US cities. As before, we quantify the privacy based on the three privacy metrics defined in Section 8.2.6. We sample and test purchase events on the Numbeo dataset only, since our test dataset does not contain sufficiently many purchase events per


**Figure 8.4:** $F_1$-score for identifying the country given purchase events *sampled from the Numbeo dataset*, corresponding to complete knowledge. Averaging does not hide poorly performing countries (cf. appendix).

| | 112 countries | | | 23 US cities | | | 84 stores | | 134 chains | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Knowledge Scenarios** | **P** | **PM** | **PPC** | **P** | **PM** | **PPC** | **P** | **PPC** | **P** | **PPC** |
| **Mutual Information** | 0.539 | 0.841 | 1.703 | 0.368 | 0.572 | 1.164 | 0.280 | 0.569 | 0.456 | 2.256 |
| **Relative Reduced E.** | 0.114 | 0.178 | 0.360 | 0.101 | 0.157 | 0.319 | 0.044 | 0.089 | 0.068 | 0.337 |

**Table 8.1:** Mutual information and relative reduced entropy for the three knowledge scenarios when estimating the country, city, store or chain of purchase events. The respective abbreviations P., PM., PPC. stand for Price, Price Merchant and Price Product-Category knowledge scenario respectively.

considered US city.

Figure 8.10 illustrates the $F_1$-score depending on the number of purchase events. We observe, that after 10 purchase events, the $F_1$-score is greater than 0.7. Therefore, our methodology also provides accurate estimations on a city granularity. Table 8.1 reports the mutual information and relative reduced entropy when estimating the US city. We observe that the relative reduced entropies of country and city granularity match across the knowledge scenarios. This exemplifies the usefulness of the relative reduced entropy to highlight similarities across different price datasets.

### 8.4.4 Chicago Metropolitan Granularity

In this section, we analyze an adversary that aims to distinguish among the purchase events of 84 Dominick's stores within the Chicago metropolitan area. We sample the price values from the Chicago dataset, and assume an adversary with complete knowledge; we therefore do not apply additive smoothing. We consider the location prior probability $P(l)$ to be uniform, because we do not have reliable store popularity information for the Chicago area.

In Figure 8.11 we can observe that the adversary can identify a local store given 100 purchase events with high confidence. We expected a weaker result, since all stores are operated by the same chain, implying relatively similar price structures. We ran our attack on each of the 85 weeks with most data, averaged the results and report the standard deviation as shown in the blue area of Figure 8.11.

Table 8.1 shows that the Chicago price dataset reveals less information about the considered locations than the Numbeo dataset. This observation holds for both knowledge scenarios, and is consistent with the result that more price points are required to localize purchase events within the Chicago area.

### 8.4.5 Store chain granularity

The large-scale Kaggle dataset does not provide precise location information of purchase events, but allows the adversary to distinguish among 134 store chains. Knowing the store chain of purchase events effectively reduces the possible locations of the purchases. Note, that the prices of Kaggle are distributed over a year and the adversary therefore does not know the precise time of the purchase events.

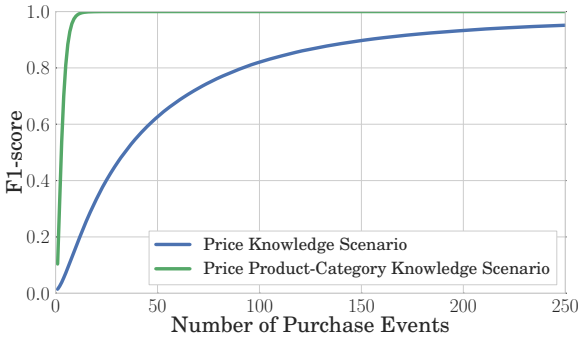



**Figure 8.5:** $F_1$-score for identifying the store chain. The purchase events are sampled from the Kaggle dataset.

We uniformly sample purchase events of different consumers and perform our attack on the Kaggle dataset. Figure 8.5 reveals that given approximately 250 price values we achieve an $F_1$-score of over 0.95 for the origin of the purchase events. Note, that the price_product-category knowledge scenario is particularly strong due to many product categories. This is reflected by the particularly high Mutual Information (cf. Table 8.1).

Given these results, we conclude, that our framework and methodology apply to a wide variety of different price datasets and allow us to quantitatively compare their respective privacy leakage. In the following, we extract further insights from our data to strengthen the attack.

### 8.4.6 Most Revealing Product Category

In this section we investigate which of the 23 considered product categories from the Numbeo dataset leak more information. This is a useful insight since an adversary would pick purchase events of this product category in order to increase the probability of correctly identifying their location. Therefore, with the mutual information we measure the extent to which the location entropy is reduced, given the purchase events of a particular product category. Contrary to the previous analysis, we evaluate the mutual information *per product category* based on the price_product-category knowledge scenario defined in Section 8.2.3. More specifically, we compute the mutual information using only pur-

chase events of a particular product category.

The results of the evaluation can be found in Figure 8.13. According to this metric, the most revealing product categories are milk, a one-way ticket for local transportation, and a loaf of white bread. On the contrary, the product categories that disclose less information about a location are oranges, chicken breasts and rice.

### 8.4.7 Required Time Precision

Previously, we assumed that knowledge of the exact currency conversion rates is required to compare non-localized purchase events. Exact currency conversion rates, however, require a precise knowledge of the purchase event times. In this section, we show that our attack does not require the exact currency conversion rates, but also works if the adversary knows only the date or even week of the purchase, i.e. it has an uncertainty of 24 hours or 7 days in relation to the conversion rates. We therefore relax the requirements on the time precision.

Due to the conversion rate differences, the adversarial estimation of $P(v \mid l, c)$ is inaccurate. To compensate for the conversion rate differences, the adversary can use a price tolerance. We study two options for the tolerance: a static tolerance and a dynamic tolerance. For the static tolerance, the adversary estimates $P(v \mid l, c)$ in the presence of uncertainty by considering price values in the interval $[v - tol_s, v + tol_s]$ where the static tolerance $tol_s$ is a small amount in global currency (e.g., 0.02 USD). The dynamic tolerance value $tol_d$ is a percentage-wise estimate of uncertainty (e.g., 2%). To estimate $P(v \mid l, c)$ the adversary considers price values from the interval $[v \cdot (1 - tol_d), v \cdot (1 + tol_d)]$.

We evaluated the attack to infer the country of purchase events with imprecise purchase times and compensated the time error with different tolerance values. To simulate imprecise purchase times, we converted the adversarial knowledge using conversion rates of 30 different days from the year 2014 and then converted the non-localized purchase events $S_U$ using the previous days' conversion rates. As before, we computed the $F_1$-score to evaluate the quality of the estimated $P(l \mid S_U)$.

For static and dynamic tolerance values, we found that the attack is still accurate, i.e. reaches an $F_1$-score above 95% with less than 50 purchase events. A higher tolerance value has two opposing effects: (i) it compensates for differences in currency conversion rates and increases the number of correctly considered price values; (ii) a higher tolerance, however, also increases the number of incorrectly considered

price values which fall into larger intervals. Therefore, the tolerance value presents a trade off between the true-positive and true-negative rate. Our experimental results reflect this trade off both for static and dynamic tolerance values (cf. appendix B). Based on our experimental results we propose a dynamic tolerance of 2% for a 24-hour time imprecision.

We also evaluated the uncertainty of one week on the currency conversion rates. We used real-world currency conversion rates that were seven days apart from each other. Figure 8.14 shows the result of this experiment for the different knowledge scenarios and a dynamic tolerance value of 2% on the Numbeo dataset. We conclude that our attack does not require precise purchase event times.

## 8.5 Conclusion

Having a systematic methodology to reason quantitatively about the privacy leakage from datasets containing price relevant information is a necessary step to avoid privacy leakages. While further tests with more datasets will help to generally claim that price values alone can reveal the location of a purchase, our empirical results provide evidence that with relatively few purchase events it is possible to identify a consumer's location. In this chapter, we have raised the following two questions: How much location information is leaked by consumer purchase datasets? How can it be quantified with the considered adversarial model and knowledge? In our proposed framework, we have modeled several adversaries and quantified the privacy leakage according to different dimensions. We make extensive use of Bayesian inference in our framework to model the different attack strategies. Our framework can be easily applied to any price dataset of consumer purchases and allows one to compare the privacy leakage of different datasets. We applied our methodology to three real-world datasets and achieve comparable results. The results presented in this chapter strongly motivate the need for careful consideration when sharing price datasets and should be considered when designing public ledger cryptocurrencies.

# Appendix 8.A: Probability calculations

In the following we clarify the individual steps for calculating the probabilities derived in Section 8.2.

$$\begin{aligned} P(v) &= \sum_{l \in \mathbb{L}} P(l, v) = \sum_{l \in \mathbb{L}} \sum_{c \in \mathbb{C}} P(l, c, v) \\ &= \sum_{l \in \mathbb{L}} \sum_{c \in \mathbb{C}} P(l) \cdot P(c \mid l) \cdot P(v \mid l, c) \end{aligned} \tag{8.10}$$

$$\begin{aligned} P(c, v) &= \sum_{l \in \mathbb{L}} P(l, c, v) \\ &= \sum_{l \in \mathbb{L}} P(l) \cdot P(c \mid l) \cdot P(v \mid l, c) \end{aligned} \tag{8.11}$$

$$\begin{aligned} P(v \mid l) &= \frac{P(l, v)}{P(l)} = \frac{\sum\limits_{c \in \mathbb{C}} P(l, c, v)}{P(l)} \\ &= \sum_{c \in \mathbb{C}} P(c \mid l) \cdot P(v \mid l, c) \end{aligned} \tag{8.12}$$

$$\begin{aligned} P(c, v \mid l) &= \frac{P(l, c, v)}{P(l)} = \frac{P(l) \cdot P(c \mid l) \cdot P(v \mid l, c)}{P(l)} \\ &= P(c \mid l) \cdot P(v \mid l, p) \end{aligned} \tag{8.13}$$

$$\begin{aligned} P(l \mid v) &= \frac{P(l) \cdot P(v \mid l)}{P(v)} \\ &= \frac{P(l) \cdot \sum\limits_{c \in \mathbb{C}} [P(c \mid l) \cdot P(v \mid l, c)]}{\sum\limits_{l' \in \mathbb{L}} \sum\limits_{c \in \mathbb{C}} [P(l') \cdot P(c \mid l') \cdot P(v \mid l', c)]} \\ &= \frac{P(l) \cdot \sum\limits_{c \in \mathbb{C}} [P(c \mid l) \cdot P(v \mid l, c)]}{\sum\limits_{l' \in \mathbb{L}} P(l') \cdot \sum\limits_{c \in \mathbb{C}} [P(c \mid l') \cdot P(v \mid l', c)]} \end{aligned} \tag{8.14}$$

$$= \frac{\frac{\text{Population}(l)}{\sum_{l' \in \mathbb{L}} \text{Population}(l')} \cdot \sum_{c \in \mathbb{C}} \left[ \frac{\text{Basket}(l,c)}{\sum_{c' \in \mathbb{C}} \text{Basket}(l,c')} \cdot \frac{D(l,c,v)}{D(l,c)} \right]}{\sum_{l' \in \mathbb{L}} \frac{\text{Population}(l')}{\sum_{l'' \in \mathbb{L}} \text{Population}(l'')} \cdot \sum_{c \in \mathbb{C}} \left[ \frac{\text{Basket}(l',c)}{\sum_{c' \in \mathbb{C}} \text{Basket}(l',c')} \cdot \frac{D(l',c,v)}{D(l',c)} \right]} \tag{8.15}$$

$$\begin{aligned} P(l \mid c, v) &= \frac{P(l) \cdot P(c, v \mid l)}{P(c, v)} \\ &= \frac{P(l) \cdot [P(c \mid l) \cdot P(v \mid l, p)]}{\sum_{l' \in \mathbb{L}} [P(l') \cdot P(c \mid l') \cdot P(v \mid l', c)]} \end{aligned} \tag{8.16}$$

$$= \frac{\frac{\text{Population}(l)}{\sum_{l' \in \mathbb{L}} \text{Population}(l')} \cdot \frac{\text{Basket}(l,c)}{\sum_{c' \in \mathbb{C}} \text{Basket}(l,c')} \cdot \frac{D(l,c,v)}{D(l,c)}}{\sum_{l' \in \mathbb{L}} \left[ \frac{\text{Population}(l')}{\sum_{l'' \in \mathbb{L}} \text{Population}(l'')} \cdot \frac{\text{Basket}(l',c)}{\sum_{c' \in \mathbb{C}} \text{Basket}(l',c')} \cdot \frac{D(l',c,v)}{D(l',c)} \right]} \tag{8.17}$$

$$\begin{aligned} P(l \mid S_U) &= P(l \mid V(e_1), V(e_2), \ldots, V(e_n)) \\ &= \frac{\prod_{i=1..n} P(V(e_i))}{P(V(e_1), V(e_2), \ldots, V(e_n))} \\ &\quad \frac{\prod_{i=1..n} P(l \mid V(e_i))}{P(l)^{n-1}} \\ &= \frac{P(l) \cdot \prod_{e \in S_U} P(V(e) \mid l)}{P(V(e_1), \ldots, V(e_n))} \end{aligned} \tag{8.18}$$

## Appendix 8.A.1: Probability calculations

Based on its knowledge, the ideal adversary computes the following probabilities by computing the fractions of events.

$$P(l) = \frac{|\{e | e \in \mathcal{H}_G : e.l = l\}|}{|\mathcal{H}_G|} \tag{8.19}$$

$$P(v) = \frac{|\{e | e \in \mathcal{H}_G : e.v = v\}|}{|\mathcal{H}_G|} \tag{8.20}$$

$$P(c, v) = \frac{|\{e | e \in \mathcal{H}_G : e.c = c \wedge e.v = v\}|}{|\mathcal{H}_G|} \tag{8.21}$$

$$P(v \mid l) = \frac{|\{e | e \in \mathcal{H}_G : e.l = l \wedge e.v = v\}|}{|\{e | e \in \mathcal{H}_G : e.l = l\}|} \tag{8.22}$$

$$P(c, v \mid l) = \frac{|\{e | e \in \mathcal{H}_G : e.l = l \wedge e.c = c \wedge e.v = v\}|}{|\{e | e \in \mathcal{H}_G : e.l = l\}|} \tag{8.23}$$

# Appendix 8.B: Further Experimental Results

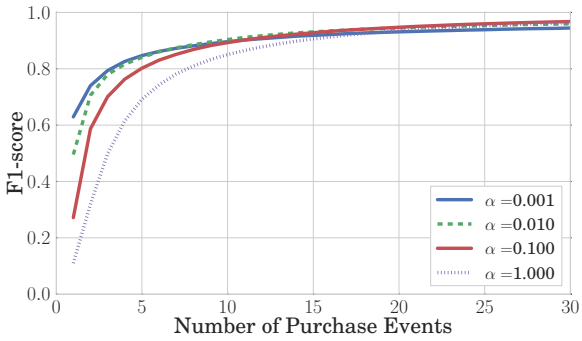



**Figure 8.6:** Comparison of different $\alpha$-parameters for additive smoothing based on the price_product-category knowledge scenario.

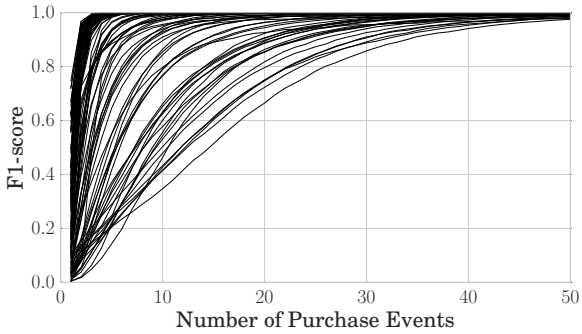



**Figure 8.7:** $F_1$-score of each individual country for the price knowledge scenario. The purchase events are sampled from Numbeo. We observe that no country performs poorly.

## Appendix 8.B.1: Required Time Precision

Figure 8.8 shows, that a larger $tol_s$ will improve the overall $F_1$-score, but more purchase events are needed to filter out the false positives. Similarly, for the dynamic tolerance in Figure 8.9, a higher value for $tol_d$ provides a better prediction for many purchase events, but a worse prediction for few purchase events. The figures show the experiments for the price_product-category knowledge scenario, however, we note that the results are analogous to the other scenarios. Based on these results we propose a dynamic tolerance of 2% in the case of a 24-hour time imprecision on the conversion rate.

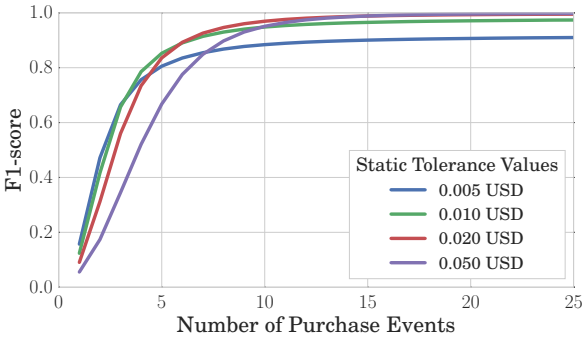



**Figure 8.8:** Using static tolerance values to compensate for imprecise time information (one day uncertainty) in the price_product-category knowledge scenario.

## Appendix 8.B.2: Motivating example

Since products appear in a multitude of price values, it is at first unclear how accurately price values can identify a location. To illustrate why purchases can be localized, we focus on an example of the product category *domestic beer (0.5L bottle)*, which can be bought in nearly every country. The price values are taken from the Numbeo dataset [29]. Figure 8.12 shows the distribution of price values of beer in USD for four countries. We observe that ranges of prices clearly differ for India and the other countries, while prices in Australia are more likely to be higher than in the US and Canada, where distributions of prices are

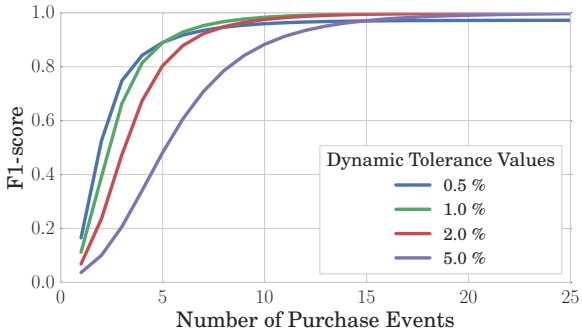



**Figure 8.9:** Using dynamic tolerance values to compensate for imprecise time information (one day uncertainty) in the price_product-category knowledge scenario.


**Figure 8.10:** $F_1$-score for identifying the US city given purchase events for different knowledge scenarios. The purchase events are sampled from the Numbeo dataset.

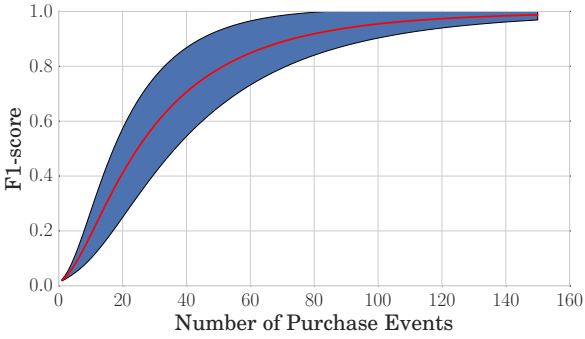



**Figure 8.11:** $F_1$-score and standard deviation over 85 weeks for identifying the store in the price knowledge scenario. Data sampled from the Chicago dataset among 84 stores.

similar. Given a beer price above 3 USD, in this case, it is highly likely that the purchase has not occurred in India.

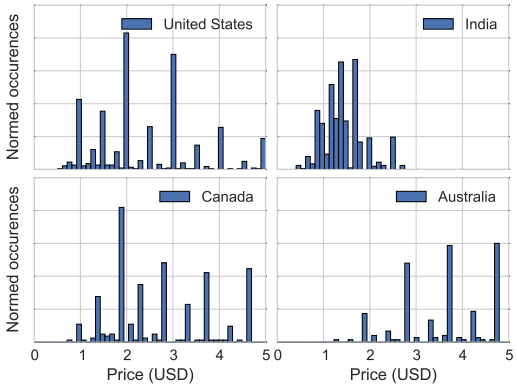



**Figure 8.12:** Distribution of domestic beer prices (0.5 Liter) in 4 countries from Numbeo in USD.

# Appendix 8.C


**Figure 8.13:** The higher the mutual information, the more revealing is the product category.


**Figure 8.14:** Dynamic tolerance of 2% with one week time uncertainty on the Numbeo dataset while estimating the country. Precise time allows an $F_1$-score of 0.95 after 10 purchase events whereas a one week time uncertainty achieves an $F_1$-score of 0.63.

| Category and Merchant | Unit | Prices |
|---|---|---|
| **Market Product Categories** | | |
| Apples | 1 kg | 11876 |
| Chicken Breasts | 1 kg | 11893 |
| Cigarettes (Marlboro) | 1 pack | 12712 |
| Domestic Beer | one 0.5 liter bottle | 10243 |
| Eggs | 12 units | 14617 |
| Imported Beer | one 0.33 liter bottle | 9484 |
| Lettuce | 1 head | 8966 |
| Loaf of White Bread | 0.5 kg | 14633 |
| Local Cheese | 1 kg | 10975 |
| Milk (regular) | 1 liter | 17197 |
| Oranges | 1 kg | 10289 |
| Potato | 1 kg | 10891 |
| Rice (white) | 1 kg | 10924 |
| Tomato | 1 kg | 10539 |
| Water | 1.5 liter bottle | 12762 |
| Wine (Mid-Range) | 1 bottle | 11893 |
| **Restaurant Product Categories** | | |
| Cappuccino (regular) | 1 unit | 21539 |
| Coke/Pepsi | one 0.33 liter bottle | 21351 |
| Fast Food Combo Meal | 1 unit | 21794 |
| Domestic Beer | one 0.5 liter bottle | 19128 |
| Imported Beer | one 0.33 liter bottle | 18048 |
| Water | one 0.33 liter bottle | 21691 |
| **Local Transportation Categories** | | |
| One-way Ticket | 1 unit | 15275 |

**Table 8.2:** Product categories of the Numbeo dataset.

| **Numbeo Dataset (2 years)** | |
|---|---|
| Number of countries | 112 |
| Number of prices | 328,720 |
| Number of cities in the US | 23 |
| Number of prices in the US cities | 11,686 |
| Number of distinct product categories | 23 |
| **Numbeo Test Dataset (5 months)** | |
| Number of countries | 47 |
| Number of prices | 40,968 |
| Number of distinct product categories | 23 |
| **Chicago Dataset (5 years)** | |
| Number of stores | 84 |
| Number of total prices in top 85 weeks | 24,070,437 |
| Average number of prices per week | 283,181 $\pm$ 6790 |
| Number of distinct product categories | 28 |
| Average number of products per week | 6304 $\pm$ 461 |
| **Kaggle Dataset (1 year)** | |
| Number of store chains | 134 |
| Number of purchase events | 349,655,789 |
| Number of consumers | 311,539 |
| Number of distinct product categories | 836 |

**Table 8.3:** Statistics about the three price datasets

# Part V

# Conclusions and Future Work

# Chapter 9

# Conclusions and Future Work

With this final chapter, we conclude this thesis. We first summarize our contributions and then outline future work. We finish with concluding remarks.

## 9.1 Summary of Contributions

In our first contribution (cf. Chapter 4), we have observed that the Bitcoin ecosystem is not decentralized, but a few individuals typically control the decision-making process. This contrasts the original purpose of Bitcoin, to decentralize the process of how monetary values is transmitted and stored. Due to the ability of third parties to exert pressure on traceable Bitcoin transactions, we show that its fungibility is moreover limited.

In Chapter 5 we provide the first realistic quantitative framework to objectively compare the security and performance provisions of Proof of Work (PoW) based blockchains. We consider the strongest possible adversaries by determining optimal strategies for selfish mining and double-spending. We provide an open source blockchain simulator which enabled us to simulate the space of possible reparametrizations of Bitcoin. Our work, therefore, allows us to increase Bitcoin's transaction throughput by a factor of ten, only by changing the block interval

time to 1 minute. According to our results, this change would not deteriorate the security of Bitcoin.

We show in Chapter 6, the previously unconsidered impact of PoW blockchain's scalability optimizations on its security. Our proposed design modifications to improve Bitcoin's security have been implemented in the main Bitcoin client.

In Chapter 7, we investigate the privacy provisions of Bloom Filters in lightweight clients. Lightweight clients typically run on low-performant hardware, such as smartphones, and are those that are the most critical to the blockchain's mainstream adoption. Our proposed design modifications are being implemented to protect the user's privacy.

We analyze in Chapter 8 the implications of public transaction prices on the location privacy. Our results show that an adversary can accurately locate a position given only a few consumer prices.

## 9.2 Future Work

In this section we highlight potential avenues for future research.

### 9.2.1 Smart Contract Security

Our work does not cover smart contract or Script security. Recall, that smart contracts allow writing Turing-complete software run on top of the blockchain. Bitcoin's Script language is not Turing-complete, but similarly permits the execution of custom software. Recent events have highlighted, that their security is fragile and multiple new classes of smart contract specific vulnerabilities emerged [72, 140]. Future work could, for instance, provide insights on smart contract security through static or dynamic code analysis. Formal verification moreover might allow making sure that the smart contract's specifications and its corresponding code are in agreement with each other.

### 9.2.2 Ring Signature Privacy

Due to its public and open log of transactions, the privacy provisions of the blockchain is a major issue. Cryptographic solutions have emerged as well as transaction mixes where participants can mix their crypto

coins. Ring signatures might provide the option to implement a decentralized mixing service.

### 9.2.3 Combining Selfish Mining and Double-Spending

In Chapter 5 we have shown how to quantify the security of PoW blockchains by studying selfish mining and double-spending independently. Our motivation to consider these attacks separately is because selfish mining is not always a rational strategy, i.e. the adversary does not earn as many block rewards with selfish mining as he would with honest mining. This observation holds as long as the difficulty of the blockchain is not adjusted. Difficulty adjustments happen every 2016 blocks in Bitcoin, while in Ethereum, the difficulty is adjusted for every block.

For a more realistic security analysis, future work should take into account the difficulty changes of the blockchain, analyze selfish mining and double-spending together with the objective to optimize the absolute revenue of the adversary.

## 9.3 Concluding Remarks

This thesis provides a thorough examination of PoW blockchain security, lightweight client privacy provisions, a view on the (de-)centralized aspects of Bitcoin as well as on the location privacy of transaction prices.

For PoW blockchain security, our work is the first that allows to objectively and realistically compare the security and performance of different PoW blockchain instantiations (e.g. Litecoin, Dogecoin, Ethereum) while accounting for their offered security. Our contributions make it possible to alter the blockchain parameters to increase the performance characteristics, without deteriorating the provided security guarantees. We moreover show the practical feasibility of affordable network wide attacks on existing blockchain networks, their impact beyond trivial denial of service attacks and propose appropriate countermeasures to mitigate the blockchain's vulnerability. Our proposed design changes that reduce these vulnerabilities are implemented in the current Bitcoin codebase.

For blockchain privacy, our work focusses on the privacy provisions of lightweight Bitcoin clients that outsource computation to full

blockchain clients and only receive transactions relevant to their operation. We show that the current design violates the lightweight client's privacy and propose appropriate countermeasures that are being implemented by the respective developers.

Throughout this thesis, we provide new ways of thinking about security and privacy of PoW blockchain technology: is Bitcoin truly decentralized, network partitioning attacks that exploit scalability measures, bloom filter for privacy in lightweight clients, modeling of the PoW blockchain as Markov Decision Processes with separation of selfish mining and double-spending.

Although blockchain technology is still in its infancy, we have highlighted and quantified many tension points and tradeoffs to build more secure, performant and privacy preserving PoW blockchains. We consider our contributions therefore as a necessary first step towards achieving a high throughput, scalable, decentralized and privacy preserving blockchain, whether based on PoW, or other consensus mechanisms.

# Bibliography


[1] Evaluating User Privacy in Bitcoin, In Proceedings of Financial Cryptography and Data Security Conference (FC), 2013. Available from http://eprint.iacr.org/2012/596.pdf.

[2] Casascius Bitcoin POS system, Available from https://en.bitcoin.it/wiki/Casascius_Bitcoin_POS_system.

[3] Contracts - Bitcoin, Available from https://en.bitcoin.it/wiki/Contracts.

[4] The Finney Attack, Available from https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack.

[5] Bitcoin Wiki, Available from https://en.bitcoin.it/wiki/.

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

[7] Bitcointalk Forum - Available from https://bitcointalk.org/.

[8] [Bitcoin-development] Revisiting the BIPS process, Available from https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg02982.html.

[9] Core Development Status Report # 1 - Bitcoin, Available from https://bitcoinfoundation.org/2012/11/01/core-development-status-report-1/.

[10] BitcoinJ, Available from http://bitcoinj.github.io/.

[11] A Face Is Exposed for AOL Searcher No. 4417749, 2006. Available from: http://www.nytimes.com/2006/08/09/technology/09aol.html.

[12] Ikea Billy Bookshelf Index, Bloomberg, 2009. Available from: http://www.bloomberg.com/apps/news?pid=newsarchive&sid=a.K4T4ypP9ko.

[13] Bitcoin Gateway, A Peer-to-peer Bitcoin Vault and Payment Network, 2011. Available from http://arimaa.com/bitcoin/.

[14] Bitcoin: Tempering the Digital Ring of Gyges or Implausible Pecuniary Privacy, 2011. Available from http://ssrn.com/abstract=1937769ordoi:10.2139/ssrn.1937769.

[15] Bitcoin Blockchain parser, 2013. Available from: https://github.com/znort987/blockparser.

[16] NIST/SEMATECH e-Handbook of Statistical Methods, 2013. Available from: http://www.itl.nist.gov/div898/handbook/.

[17] Bitcoin Wallet, Android, 2014. Available from: https://play.google.com/store/apps/details?id=de.schildbach.wallet.

[18] BitcoinJ, privacy assumptions, 2014. Available from: https://github.com/bitcoinj/bitcoinj/blob/ee2a91010e5cf66299684160d6a48a108ff2299b/core/src/main/java/com/google/bitcoin/core/PeerGroup.java#L250.

[19] Big Mac Index, The Economist, 2015. Available from: http://www.economist.com/content/big-mac-index.

[20] Bitcoin dev commit 4547: prevent peer flooding request queue for an inv, 2015. Available from: https://github.com/bitcoin/bitcoin/pull/4547.

[21] Bitcoin dev commit 4831: net: Better askfor request management, 2015. Available from: https://github.com/bitcoin/bitcoin/pull/4831.

[22] Bitcoin Mailing List - Peter Todd about how many pools use an additional relay network, 2015. Available from: http://sourceforge.net/p/bitcoin/mailman/message/34152876/.

[23] Bitcoin Relay Network, 2015. Available from: http://bitcoinrelaynetwork.org/.

[24] Bitcoin XT, 2015. Available from: https://github.com/bitcoinxt/bitcoinxt.

[25] Consumer panel data and retail scanner data across the United States., 2015. Available from: http://research.chicagobooth.edu/nielsen/.

[26] Double spending in Bitcoin, 2015. Available from: https://medium.com/@octskyward/double-spending-in-bitcoin-be0f1d1e8008.

[27] Kaggle, Acquire Valued Shoppers Challenge, 2015. Available from: https://www.kaggle.com/c/acquire-valued-shoppers-challenge.

[28] More (or Less) Brew for your Buck, Starbucks coffee price, 2015. Available from: http://online.wsj.com/news/articles/SB10001424127887324048904578319783080709860.

[29] Numbeo, database of user contributed data about cities and countries worldwide., 2015. Available from: http://www.numbeo.com.

[30] Ripple, cryptocurrency, 2015. Available from: https://ripple.com/.

[31] Store-level scanner data collected at Dominick's Finer Foods., 2015. Available from: http://research.chicagobooth.edu/kilts/marketing-databases/dominicks/dataset.

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

[33] Warenkorbstruktur Schweiz, 2015. Available from: http://www.bfs.admin.ch/.

[34] World Population, The world bank, 2015. Available from: http://data.worldbank.org/indicator/SP.POP.TOTL?order=wbapi_data_value_2009+wbapi_data_value+wbapi_data_value-first&sort=asc.

[35] Bitcoin block size limit controversy, 2016. Available from: https://en.bitcoin.it/wiki/Block_size_limit_controversy.

[36] Cryptocurrencies Market Capitalization, 2016. Available from: http://coinmarketcap.com/.

[37] CryptoLocker, 2016. Available from: https://en.wikipedia.org/wiki/CryptoLocker.

[38] Financial Audit, Wikipedia, 2016. Available from: https://en.wikipedia.org/wiki/Financial_audit.

[39] Hyperledger, 2016. Available from: https://www.hyperledger.org/.

[40] Lighthouse, 2016. Available from: https://github.com/vinumeris/lighthouse.

[41] Litecoin, 2016. Available from: https://litecoin.org/.

[42] Onward from the Hard Fork, 2016. Available from: https://blog.ethereum.org/2016/07/26/onward_from_the_hard_fork/.

[43] Satoshidice, 2016. Available from: https://satoshidice.com.

[44] Silkroad, 2016. Available from: https://en.wikipedia.org/wiki/Silk_Road_(marketplace).

[45] Thinking About Smart Contract Security, 2016. Available from: https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/.

[46] Elli Androulaki and Ghassan O Karame. Hiding transaction amounts and balances in bitcoin. In *Trust and Trustworthy Computing*, pages 161–178. Springer, 2014.

[47] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, Zongren Liu, and Christian A Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 831–843. ACM, 2014.

[48] Vasileios Glykantzis Arthur Gervais. Bitcoin simulator. Available from: `http://arthurgervais.github.io/Bitcoin-Simulator/`.

[49] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On Bitcoin and Red Balloons. In Proceedings of the ACM Conference on Electronic Commerce (EC'12), 2012.

[50] T. Bamert, C.Decker, L. Elsen, R. Wattenhofer, and S. Welten. Have a Snack, Pay with Bitcoins. 13-th IEEE International Conference on Peer-to-Peer Computing, 2013.

[51] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to Better - How to Make Bitcoin a Better Currency. In *Proceedings of Financial Cryptography and Data Security*, 2012.

[52] Steven M Bellovin and William R Cheswick. Privacy-enhanced searches using encrypted bloom filters. *IACR Cryptology EPrint Archive*, 2004:22, 2004.

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

[54] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. Better than nothing privacy with bloom filters: To what extent? In *Privacy in Statistical Databases*, pages 348–363. Springer, 2012.

[55] Bitnodes. Bitnodes ip crawler. Available from: `https://github.com/ayeowch/bitnodes`.

[56] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[57] Andrew J Blumberg and Peter Eckersley. On locational privacy, and how to avoid losing it forever. *EEF*, 2009.

[58] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, May 2015.

[59] Stefan Brands. Electronic cash on the internet. In *Network and Distributed System Security, 1995., Proceedings of the Symposium on*, pages 64–84. IEEE, 1995.

[60] D Brown. Sec 1: Elliptic curve cryptography, 2010. Available from: `http://www.secg.org/sec1-v2.pdf`.

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

[62] V. Buterin. A next-generation smart contract and decentralized application platform, 2014.

[63] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–321. Springer, 2005.

[64] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[65] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.

[66] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.

[67] J. Clark and A. Essex. (Short Paper) CommitCoin: Carbon Dating Commitments with Bitcoin. In *Proceedings of Financial Cryptography and Data Security*, 2012.

[68] Matt Corallo. Bitcoin relay network. Available from: `http://bitcoinrelaynetwork.org/`.

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

[70] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[71] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[72] Phil Daian. The dao exploit. Available from: `http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/`.

[73] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895*, 2015.

[74] Yves-Alexandre de Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3, 2013.

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

[76] Bitcoin Developers. Bitcoin address generation. Available from: `https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses`.

[77] Shantanu Dutta, Mark Bergen, and Daniel Levy. Price flexibility in channels of distribution: Evidence from scanner data. *Journal of Economic Dynamics and Control*, 26(11):1845 – 1900, 2002.

[78] Meiklejohn et al. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, 2013. ACM.

[79] Ethereum. Ethereum tie breaking. Available from: `https://github.com/ethereum/go-ethereum/commit/bcf565730b1816304947021080981245d084a930`.

[80] Ethereum. ethernodes. Available from: `https://www.ethernodes.org/network/1`.

[81] Ethereum. ethstats. Available from: `https://ethstats.net/`.

[82] P. Everaere, I. Simplot-Ryl, and I. Traore. Double Spending Protection for E-Cash Based on Risk Management. In *Proceedings of Information Security Conference*, 2010.

[83] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[84] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[85] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[86] Arthur Gervais and Ghassan Karame. Bitcoin protocol specification. In *Bitcoin and Blockchain Security (Chapter 3), ISBN: 978-1-63081-013-9, 2016 (Invited Chapter)*, 2016.

[87] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? *IEEE Security and Privacy Magazine, 2014*.

[88] Arthur Gervais, Ghassan O. Karame, Damian Gruber, and Srdjan Capkun. 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*, 2014.

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

[90] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42. ACM, 2003.

[91] Mike Hearn. Connection bloom filtering, 2012. Available from: `https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki`.

[92] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security*, pages 129–144, 2015.

[93] Roland Herrmann and Anke Möser. Price variability or rigidity in the food-retailing sector? theoretical analysis and evidence from german scanner data. Technical report, 2003.

[94] Daniel Hosken and David Reiffen. Patterns of retail price variation. *RAND Journal of Economics*, pages 128–146, 2004.

[95] Ronald A Howard. *Dynamic Probabilistic Systems, Volume I: Markov Models*, volume 1. Courier Corporation, 2012.

[96] IBM. Ibm openblockchain. Available from: `http://www.ibm.com/blockchain/`.

[97] Intel. Proof of elapsed time (poet). Available from: `http://intelledger.github.io/`.

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

[99] Ghassan O Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in bitcoin: a study of double-spending and accountability. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):2, 2015.

[100] John G Kemeny, J Laurie Snell, and Gerald L Thompson. Finite mathematics. *DC Murdoch, Linear Algebra for Undergraduates*, 1974.

[101] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 85–86. ACM, 2012.

[102] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[103] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *University of Maryland and Cornell University*, 2015.

[104] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014.

[105] KU Leuven. Ripemd160. Available from: `https://homes.esat.kuleuven.be/~bosselae/ripemd160.html`.

[106] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[107] D. Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. Available from: `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`.

[108] R.C. Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[109] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[110] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes.

[111] Tyler Moore and Nicolas Christin. Beware the middleman: Empirical analysis of bitcoin-exchange risk. Proceedings of Financial Cryptography, 2013.

[112] James K Mullin. A second look at bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[113] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008.

[114] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. Location privacy via private proximity testing. In *NDSS*, volume 11, 2011.

[115] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 305–320, March 2016.

[116] NIST. Ecdsa. Available from: `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.

[117] NIST. Sha 256. Available from: `https://web.archive.org/web/20130526224224/http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf`.

[118] Ryo Nojima and Youki Kadobayashi. Cryptographically secure bloom-filters. *Transactions on Data Privacy*, 2(2):131–139, 2009.

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

[120] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[121] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In Yaniv Altshuler, Yuval Elovici, Armin B. Cremers, Nadav Aharony, and Alex Pentland, editors, *Security and Privacy in Social Networks*, pages 197–223. Springer New York, 2013.

[122] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In Proceedings of Financial Cryptography, 2013.

[123] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[124] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*. 2016.

[125] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[126] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41, 2009.

[127] Reza Shokri, George Theodorakopoulos, George Danezis, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Quantifying location privacy: the case of sporadic location exposure. In *Privacy Enhancing Technologies*, pages 57–76. Springer, 2011.

[128] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[129] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[130] S Joshua Swamidass and Pierre Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling*, 47(3):952–964, 2007.

[131] Latanya Sweeney. Simple demographics often identify people uniquely. *Health (San Francisco)*, 671:1–34, 2000.

[132] testmy.net. testmy.net. Available from: `http://testmy.net/country`.

[133] Jonathan Toomim. blocktorrent. Available from: `http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011176.html`.

[134] U.S. Census Bureau, Population Division. Annual Estimates of the Resident Population for Incorporated Places of 50,000 or More, Ranked by July 1, 2013, 2014.

[135] Verizon. Verizon latency. Available from: `http://www.verizonenterprise.com/about/network/latency/`.

[136] Athanasios S Voulodimos and Charalampos Z Patrikakis. Quantifying privacy in terms of entropy for context aware services. *Identity in the Information Society*, 2(2):155–169, 2009.

[137] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *Proceedings of the IFIP WG 11.4 Workshop iNetSec 2015*. 2015.

[138] Chiemi Watanabe and Yuko Arai. Privacy-preserving queries for a das model using encrypted bloom filter. In *Database systems for advanced applications*, pages 491–495. Springer, 2009.

[139] Wikipedia. Ecdsa. Available from: `https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm`.

[140] www.slock.it. The dao. Available from: `https://daohub.org/`.

[141] Vivek Kumar Singh Alex Sandy Pentland Yves-Alexandre de Montjoye, Laura Radaelli. Unique in the shopping mall: on the reidentifiability of credit card metadata. *Science, 2015*.