Applications of the Blockchain using Cryptography

Patrick McCorry

School of Computing Science
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

February 2018

This work is dedicated to my family Anne Haddad (Mother), James McCorry (Father), James McCorry (Brother) and Brian McCorry (Uncle).

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Patrick McCorry
February 2018

Acknowledgements

This research would not have been possible without my advisors Feng Hao and Siamak F. Shahandashti who were willing to jump aboard this emotional roller-coaster with me. I thank them for supporting me while I pursued researching Bitcoin which at the onset could have easily failed, and alongside it my PhD. I also thank my colleagues Syed-Taha Ali and Peter Hyun-Jeen Lee who listened to me ramble about Bitcoin when few others would.

Outside of Newcastle University, I am eternally grateful to Andrew Miller for inviting me to visit University of Illinois at Urbana-Champaign to experience American-style research. I'll never forget how I felt when he acknowledged me as an *Independent Researcher* which respectfully suggested that I should no longer consider myself a graduate student. I'm grateful to Malte Möser for my first (and hopefully not last) collaborative experience. In the Bitcoin community, I thank the participants of `#bitcoin-dev` and `#bitcoin-wizards` as without their insightful discussions I would not comprehend or fully appreciate the technology as I do today. I'd like to acknowledge some participants by their pseudonyms (keeping with the community's tradition). This includes sipa, gmaxwell, wumpus, hearn, gavinandresen, andytoshi, waxwing, kanzure and many others. On the other hand, in the Ethereum community, I want to thank Nick Johnson (and the Ethereum Foundation as a whole) for patiently (and quickly) answering questions I had about Ethereum/Solidity.

Outside of the research community I thank my close friends Jack MacKenzie and Sam Finnigan for the past seven years. I thank Melanie Smith, Gina Smith, Rebecca Smith and Bill Robson for their friendship, and most importantly accommodating my precious puppies, booshy boo and summer. I am grateful for the support of Dr Nigel Thompson and Elaine Stoker (NHS Freeman Hospital) for diagnosing me with Crohn's disease at the start of this PhD and helping me towards remission. I would also like to acknowledge two teachers from Christian Brothers School (Belfast), Paul Gault and Lawrence Watson, who mentored and pushed me (e.g. provided me an office so I had solitude to study) towards university. It looks like we achieved it - the first "sixth form" graduate with a PhD.

Finally I am grateful to have truly lived this topic and witness first-hand the rise of cryptocurrency research from less than ten academic papers at the start of this PhD to more than a thousand at the time of submission.

Abstract

We have witnessed the rise of cryptocurrencies in the past eight years. Bitcoin and Ethereum are the world's most successful cryptocurrencies with market capitalisations of $37bn and $21bn respectively in June 2017. The innovation behind these cryptocurrencies is the blockchain which is an immutable and censorship resistant public ledger. Bitcoin introduced the blockchain to trade a single asset (i.e. bitcoins), whereas Ethereum adopted the blockchain to store and execute expressive smart contracts. In this thesis, we consider cryptographic protocols that bootstrap trust from the blockchain. This includes secure end-to-end communication between two pseudonymous users, payment protocols, payment networks and decentralised internet voting. The first three applications rely on Bitcoin, whereas the final e-voting application is realised using Ethereum.

First, it is important to highlight that Bitcoin was designed to protect the anonymity (or pseudonymity) for financial transactions. Nakamoto proposed that financial privacy is achievable by storing each party's pseudonym (and not their real-world identity) in a transaction. We highlight that this approach for privacy has led to real-world authentication issues as merchants are failing to re-authenticate customers in post-transaction correspondence. To alleviate these issues, we propose an end-to-end secure communication protocol for Bitcoin users that does not require any trusted third party or public-key infrastructure. Instead, our protocol leverages the Blockchain as an additional layer of authentication. Furthermore, this insight led to the discovery of two attacks in BIP70: Payment Protocol which is a community-accepted standard used by more than 100,000 merchants. Our attacks were acknowledged by the leading payment processors including Coinbase, BitPay and Bitt. As well, we have proposed a revised Payment Protocol that prevents both attacks.

Second, Bitcoin as deployed today does not scale. Scalability research has focused on two directions: 1) redesigning the Blockchain protocol, and 2) facilitating 'off-chain transactions' and only consulting the Blockchain if an adjudicator is required. We focus on the latter and provide an overview of Bitcoin payment networks. These consist of two components: payment channels to facilitate off-chain transactions between two parties, and the capability to fairly exchange bitcoins across multiple channels. We compare Duplex Micropayment Channels and Lightning Channels, before discussing Hashed Time Locked Contracts which

enable Bitcoin-based payment networks. Furthermore, we highlight challenges in routing and path-finding that need to be overcome before payment networks are practically feasible.

Finally, we study the feasibility of executing cryptographic protocols on Ethereum. We provide the first implementation of a decentralised and self-tallying internet voting protocol with maximum voter privacy as a smart contract. The Open Vote Network is suitable for boardroom elections and is written as a smart contract for Ethereum. Unlike previously proposed Blockchain e-voting protocols, this is the first implementation that does not rely on any trusted authority to compute the tally or to protect the voter's privacy. Instead, the Open Vote Network is a self-tallying protocol, and each voter is in control of the privacy of their own vote such that it can only be breached by a full collusion involving all other voters. The execution of the protocol is enforced using the consensus mechanism that also secures the Ethereum blockchain. We tested the implementation on Ethereum's official test network to demonstrate its feasibility. Also, we provide a financial and computational breakdown of its execution cost.

Table of contents

Chapter 1

Introduction

Nakamoto posted the Bitcoin whitepaper [90] on The Cryptography Mailing list at metzdowd.com on 1st November 2008 [91]. Shortly afterwards on 9th January 2009, the source code was published alongside connection instructions for the Bitcoin network [92]. The motivation behind Bitcoin is famously captured in the *genesis block* of its public ledger:

The Times 03/Jan/2009 Chancellor on brink of second bailout for banks

After all, Bitcoin was born in the aftermath of the worst financial crisis since the Great Depression in the 1930s that led to subsequent bailouts for both banks[1] and governments[2]. Thus, it is no surprise that Bitcoin is a global transaction system that provides its users with full autonomy over their money. However, even in this financial climate, it is still remarkable that Bitcoin has succeeded with a $37bn market capitalisation and generated over $1bn of venture capitalist investment in Bitcoin/Blockchain FinTech startups [60], whereas most e-cash protocols proposed in the past thirty-years failed to garner practical use.

With hindsight, these e-cash protocols failed for several reasons, including the need for wide-spread acceptance from both customers and merchants[3], reliance on banks (and not private enterprise) to issue coins [13], and ultimately the banking infrastructure is increasingly growing to support micropayments of $5 or less [96]. Remarkably, Nakamoto's ingenuity overcame these issues by introducing the *Blockchain* that stores every transaction in a

[1] Lehman Brothers filed for banksruptcy [80]. Other banks required a bailout including Goldman Sachs, Morgan Stanley, Citigroup, Bank of America [87] and the Royal Bank of Scotland [121].

[2] Eurozone countries Greece, Portugal, Ireland, Spain and Cyprus were unable to finance their government debt and required a bailout from other eurozone countires, the European Central Bank and the International Monetary Fund [116].

[3] Chaum described this problem as the following: *"It's a chicken-and-egg sort of proposition. You can't sign merchants if you don't have enough users. You can't sign users if you don't have enough merchants. It takes efforts to build"* [46].

public ledger and the consensus mechanism, now known as *Nakamoto consensus*, provides permissionless governance of the Blockchain. Both innovations facilitate *peer-to-peer transactions* that are authorised by a *decentralised network of peers* and there is no barrier to entry as users independently compute their *pseudonymous credentials* for sending and receiving bitcoins.

Clearly, the envisioned application of Bitcoin is to provide a medium of exchange between two or more parties over the Internet. Today, merchants rely on e-mail addresses to re-authenticate messages from customers once a payment is confirmed. Unfortunately, there is no community-accepted standard to associate an e-mail address with the customer's *pseudonymous credentials* that authorised the payment. As such, the motivation for Chapters 3 and 4 is to overcome the inability that real-world merchants face when attempting to re-authenticate messages from pseudonymous customers. The former proposes two authenticated key exchange protocols for *post-payment communication*, whereas the latter proposes new attacks on the community-accepted BIP70: Payment Protocol that is used by over 100,000 merchants alongside a proposed revision of the standard.

Chapter 5 focuses on the scalability issues that limit Bitcoin as a medium of exchange due to the underlying Blockchain protocol. The Blockchain's current parameters limit its throughput to 3.3-7 transactions per second and simply re-parameterising can inadvertently weaken its robustness and security [30]. This chapter provides an extensive survey for *off-chain transactions* that facilitiates two or more parties to privately exchange thousands of transactions and in the best case only two transactions are stored in the Blockchain. Unfortunately, most of the state-of-the-art research is scattered across message boards, chatrooms, mailing lists and private discussions. Thus, the motivation is to summarise this emerging field to encourage other researchers to pursue further research.

Interestingly, the above scalability approach is an example of a smart contract that allows the Bitcoin network to enforce the conditions of an agreement between two or more parties. These smart contracts are heralded to revolutionise finance [98], asset management [68], and government interaction with citizens [126]. Unfortunately, Bitcoin does not easily support the creation of smart contracts due to its limited scripting facility.

This has led to the rise of Ethereum which is the second most popular cryptocurrency with a market capitalisation of $21bn. Similarly to Bitcoin, Ethereum has a Blockchain that is governed by a *decentralised network of peers*. Yet, it supplies an expressive programming language for developers to write smart contracts and both the code and its execution transcript are stored in the Blockchain. Most importantly, it introduces *consensus computing* as all peers perform the same computation to enforce the contract's correct execution and verify

the Blockchain's contents. Unfortunately, the public verifiability of these contracts requires all information to be publicly available.

The lack of privacy for smart contracts motivates the research in Chapter 5 that experimentally tests the feasibility of implementing and executing cryptographic protocols on the Ethereum network. It is important to note that cryptography has the capability to preserve the confidentality of data while maintaining the contract's public verifiability. This chapter considers the application of e-voting on the Blockchain and provides an implementation of the Open Vote Network which is a self-tallying internet voting protocol that guarantees maximum voter privacy. The computation and financial breakdown derived from experiments on the network highlights that Ethereum as deployed today can support cryptography, but it is not yet ready for wide-spread use.

1.1 Thesis Outline

This thesis is outlined as follows:

- Chapter 2 presents background information and the inner workings of both Bitcoin and Ethereum. We highlight how users compute their pseudonymous credentials, the structure of a transaction, the underlying blockchain protocol and Nakamoto Consenseus that permits the decentralised network of peers to achieve a consistent view of the blockchain.

- Chapter 3 proposes the first two protocols to re-authenticate two pseudonymous parties that share a transaction history for *post-payment communication*. This is not covered in the community-accepted BIP70: Payment Protocol or IEEE, ISO/IEC security standards. Furthermore, there is currently no PKI-based or password-based authenticated key exchange protocols that are suitable for this purpose.

- Chapter 4 highlights a flaw in BIP70: The Payment Protocol that introduces the *Silkroad Trader* and *Marketplace Trader* attacks which are acknowledged by Coinbase, BitPay and Bitt. These attacks rely on an underlying flaw in the protocol that does not provide the merchant with publicly verifiable evidence that the refund address he received during the protocol exchange was endorsed by the pseudonymous customer. We amend the payment protocol to fix this flaw and prevent both attacks.

- Chapter 5 provides a survey for payment networks that facilitate *off-chain transactions* between two or more parties. The survey presents technical descriptions of *payment channels*, *Duplex Micropayment Channels* and *Lightning Channels* before discussing

future problems that need to be solved before payment networks can become a reality. Unfortunately, the only academic publication in this survey is *Duplex Micropayment Channels*. Our contribution is gathering the information from private discussions, mailing lists and public chatrooms to help other researchers in this field understand the current progress for this scaling approach.

- Chapter 6 presents the first smart contract implementation of a boardroom voting protocol. It investigates the feasibility of executing cryptographic protocols on the Ethereum network and the benefits of *consensus computing* to self-enforce the correct execution of the voting protocol. At the time of writing the voting protocol costs $0.73 per voter, and $31.98 for forty voters to participate in an election. We highlight the technical difficulties of developing cryptographic protocols using the Solidity language and subtle (and not immediately obvious) attack vectors that are unique to smart contracts. Finally, we can conclude from the computational and financial breakdown of our experiments that Ethereum can support cryptography, but it is not ready for wide-spread use.

1.2 Collaborators and Publications

.

Chapters 3, 4, 5 and 6 reflect papers that we published in international conferences. A list of these publlications with the co-authors and acknowledgements is provided below:

- **Authenticated Key Exchange over Bitcoin**, Patrick McCorry, Siamak F. Shahandashti, Dylan Clarke, and Feng Hao, accepted by the 2nd Security Standardisation Research Conference in Tokyo, Japan [77].
- **Refund Attacks on Bitcoins Payment Protocol**, Patrick McCorry, Siamak F. Shahandashti, and Feng Hao, accepted at the 20th Financial Cryptography and Data Security conference, Bridgetown, Barbados [78].
- **Towards Bitcoin Payment Networks**, Patrick McCorry, Malte Möser, Siamak F. Shahandashti, and Feng Hao, invited paper for 21st Australasian Conference on Information Security and Privacy , Melbourne, Australia [76].
- **A Smart Contract for Boardroom Voting with Maximum Voter Privacy**, Patrick McCorry, Siamak F. Shahandashti, and Feng Hao, accepted at the 21st Financial Cryptography and Data Security conference, Sliema, Malta [79].

Chapter 2

Cryptocurrency Background

2.1 Bitcoin

Bitcoin [90] is the world's first cryptocurrency whose value is not endorsed by any central bank, but is based on the perception of its users [70]. This cryptocurrency was created by Satoshi Nakamoto whose real-world identity remains unknown. His vision was to introduce a peer-to-peer electronic cash system that does not rely on a central authority to issue currency or authorise transactions. Instead, he proposed that a decentralised network of peers can provide the infrastructure to maintain an immutable, censorship-resistant and public ledger that stores all transactions on the network.

In this section, we outline the inner-workings of Bitcoin. We focus on the concept of a Bitcoin address which is effectively the user's pseudonymous identity and transactions that let users authorise the transfer of bitcoins to others. Afterwards we present the Blockchain that stores all transactions on the network and Nakamoto consensus which is the first leader election protocol that requires a distributed set of peers to compete in a probabilistic election. The winner of this competition is elected as a one-time leader to authorise which transactions should be accepted into the Blockchain. In fact, it is this probabilistic election that differs from all previous consensus protocols as users are appointed based on their computational power as opposed by an authority. Finally we highlight that the entire protocol is publicly verifiable and this allows a peer-to-peer network operated by altruistic users to verify the Blockchain's correctness (and also propagate new transactions to all other peers). To finish we discuss how Bitcoin has inspired the community to re-think how the Blockchain can be applied to solve problems in other domains.

A **Bitcoin address** is a pseudonymous identity that is used to send and receive bitcoins. An address can be described as the hash of an EC (Elliptic Curve) public key and the accompanying private key is used to produce ECDSA (Elliptic Curve Digital Signature

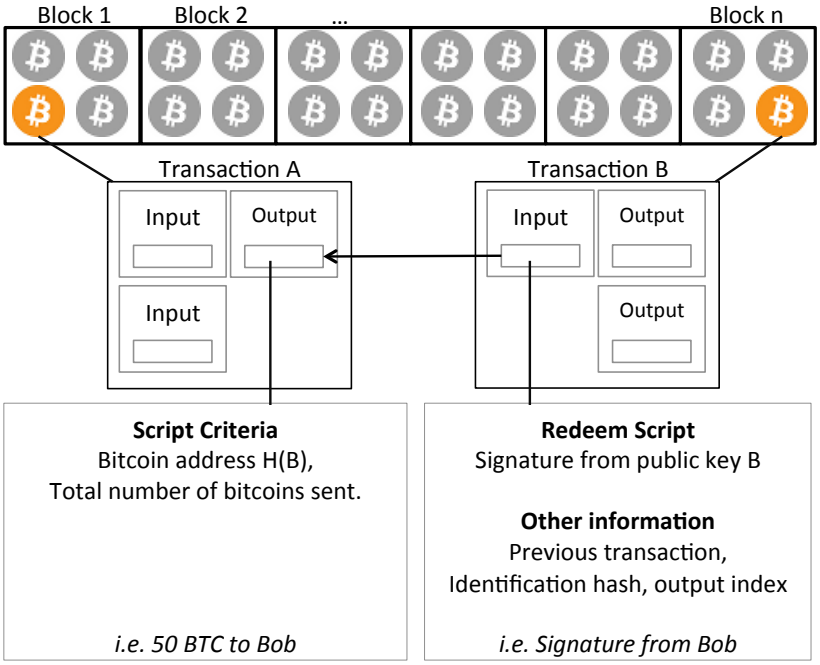


Figure 2.1 Transactions stored in the Blockchain

Algorithm) signatures to authorise payments. This approach is considered appropriate as the probability that two users generate the same public key is negligible due to the high number of possible ECDSA public keys. The corresponding private key for a new Bitcoin address is computed using a random number generator or deterministically derived from a pseudo-random seed [129]. In order to obfuscate the ownership of coins, the community recommends using a new Bitcoin address when receiving coins on the network. However this obfuscation only provides limited financial privacy if the user's wallet software is not careful when selecting which coins to spend [104, 9, 101, 82].

A **Transaction** consists of one or more inputs and one or more outputs as seen in Figure 2.1. Briefly, an input specifies the source of bitcoins being spent (the previous transaction's identification hash and an index to one of its output) and is accompanied with signature(s) and public key(s) of the sender to authorise the payment. An output specifies the new owner's Bitcoin address and the number of bitcoins being sent. Strictly, these inputs and outputs are controlled using a Forth-like scripting language to dictate the conditions required to claim the bitcoins. The dominant script today is the *pay-to-pubkey-hash* which requires a single signature from a Bitcoin address to authorise the payment. On the other hand, the *pay-to-script-hash* approach enables a variety of transaction types and was introduced as a soft-fork in BIP16 [5]. In practice, this pay-to-script-hash script is widely used[1] to enable escrow services and multi-signature authorisation (k of n keys required to claim bitcoins).

[1]Currently 10.8% of all bitcoins are stored using the pay-to-script-hash approach [2].

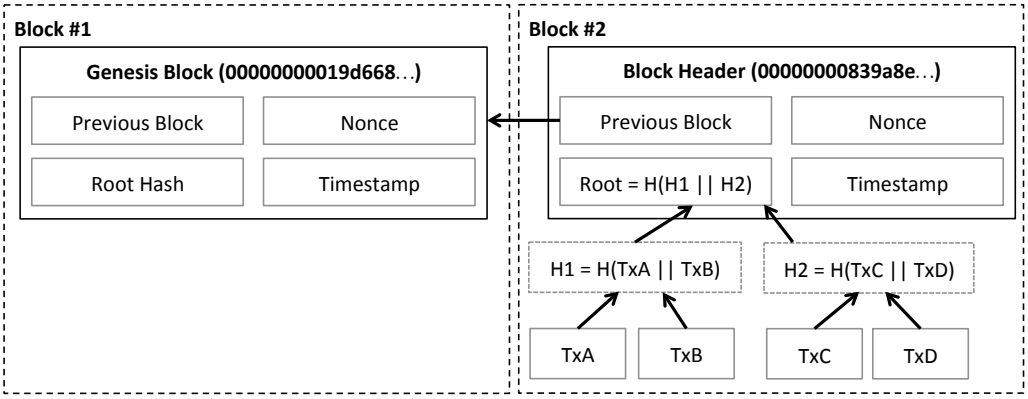


Figure 2.2 Two blocks in the Blockchain

The **Blockchain** is a mechanism to synchronise a set of replicated databases and provide a consistent view to all peers on the network. In the case of Bitcoin, it allows each peer to compute an up to date list of spendable coins and the associated scripts that must be satisfied in order to spend them. The network's maintainers are responsible for creating a new block of transactions approximately every ten minutes and once created the block is appended to the Blockchain's tip as seen in Figure 2.1. This append-only feature effectively creates a chain of blocks resulting in the name *Blockchain* and all transactions are stored in chronological order. The Blockchain is publicly verifiable which is reflected in the community's motto, *Trust, but verify*. This allows a peer who is receiving coins on the network to independently verify whether the sender is authorised to spend the coins and crucially whether the network's maintainers are attempting to censor transactions or include invalid transactions.

Technically, there is no real concept of a coin in Bitcoin. Instead, the Bitcoin client maintains a database of unspent transaction outputs (UTXO set) which states the bitcoins that can be redeemed if the corresponding script is satisfied. Blocks of transactions are responsible for updating this UTXO database and a block has two components. The first component is a list of recent transactions and a coinbase transaction[2] that sends the winning miner their bitcoins. The second component is a block header as seen in Figure 2.2 which contains the identification hash of the previous block and a merkle tree root that provides a computationally binding commitment to the included set of transactions.[3] Most importantly, the identification hash of the block is also a proof of work that provides the computational commitment to the entire block's content. The purpose of this proof of work and how the network reaches consensus on the list of unspent transaction outputs is discussed next.

[2]This transaction has no inputs and is the first item in the list.

[3]A merkle tree is constructed using the list of transactions in this block. The root of this tree is stored in the block header.

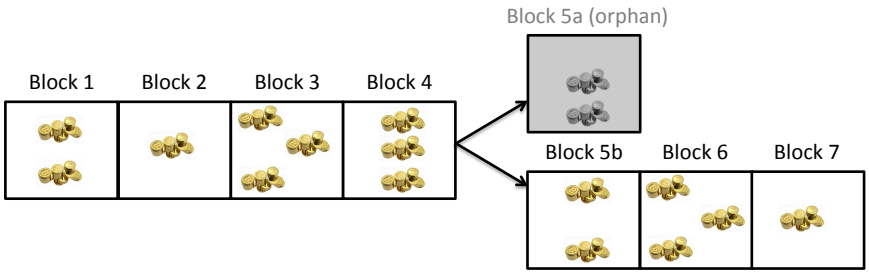


Figure 2.3 Block 5a and 5b are competing to be accepted into the Blockchain.

Nakamoto consensus provides governance of the Blockchain such that it is possible to synchronise a set of replicated databases that *mutually distrust each other* without a trusted authority to appoint the maintainers. This is achieved by creating a competition amongst pseudonymous peers to solve a computationally (and probabilistically) difficult puzzle. Competing peers are called *miners* and anyone with significant computational resources can participate. The first miner to solve the puzzle is elected as a one-time leader to create and append a new block to the Blockchain. In return for solving this puzzle, the miner is rewarded the sum of all transaction fees and a subsidy (i.e. 12.5 bitcoins in December 2016). A puzzle solution is both a proof of work and a computational commitment to the block's content. All peers verify whether the proof of work is a valid solution to the network's puzzle before appending this block to the Blockchain and propagating the block to other peers. Finally the miner's reward is spendable once this block has achieved a sufficient depth in the blockchain (i.e. it is the 100th block from the blockchain's tip).

Figure 2.3 highlights how the probabilistic nature of the computational puzzle allows two or more miners to simultaneously create blocks that compete for the same position (i.e. block height) in the Blockchain. Only one block is appended to the Blockchain and all other competing blocks are simply discarded (and become known as *stale blocks*). The miner of a stale block receives no reward. To resolve a fork requires miners to select one of the competing blocks before attempting to solve the next puzzle which can potentially split the network's computational power until a fork associated with the most *proof of work* emerges and becomes the credible Blockchain. As such, it is recommend practice to wait for a transaction to achieve a depth of six confirmations in the Blockchain[4] before considering it irreversible due to the risk of a fork. Fundamentally, the security of Nakamoto consensus assumes that 51% or more of the network's computation is honest [5] as this guarantees that one chain will eventually emerge as the longest/heaviest.

[4]The sixth most recent block at the Blockchain's tip.

[5]Rational miners with less than 51% of the network's computational power can follow different strategies to gain more bitcoins than they deserve [40][95][109].

Both the concept of *governance by a computational competition* and *proof of work* provides *censorship resistance* and *immutability* properties for the Blockchain. First, miner's can be located anywhere in the world and if less than 51% of the network's miners are under duress, then other miners can include censored transactions.[6] Second, the computational *proof of work* associated with each block increases the difficulty overtime for an attacker to reverse transactions in the Blockchain. To perform a history-revision attack [14] requires re-solving puzzles starting with the block that contains this transaction to the Blockchain's current tip. For example, if the transaction is in block 500, and the Blockchain's tip is block 1000, then the attacker must re-solve all puzzles from block 500 to block 1000. A successful attack requires more than 51% of the network's computational power in order to create blocks faster than all other honest miners.

Finally, the consensus protocol dictates the monetary policy of Bitcoin. The creation of new bitcoins in each block is programmed to halve every four years[7] until 21m bitcoins exist. As such, the financial incentive for miners to continue maintaining the Blockchain is expected to shift from the block reward to transaction fees.[8] It is anticipated that a *fee market* will emerge to cover the cost of mining and this will require users to bid for their transactions to be accepted into the Blockchain [88]. However, selecting which transactions to include in a block is an instance of the bin-packing problem and miners must also be cautious as poorly packed blocks can directly impact the number of forks that occur on the network [30]. Therefore, there is a trade-off between the sum of fees a miner can collect and which transactions are accepted into the Blockchain.

The **peer-to-peer network** is *open-membership* as peers can participate at any time. There are three roles as illustrated in Figure 2.1 which include miners who maintain the Blockchain, volunteers who verify that new transactions (and blocks) satisfy the network's consensus rules before propagating the transactions/blocks to other peers, and users who only store sufficient information to verify that transactions relevant to them are confirmed in the Blockchain. To fulfil these roles, each peer operates a different type of node:

- **Full node.** Stores a full copy of the Blockchain and verifies all transactions.
- **Pruned node.** Stores a pruned copy of the Blockchain which is a list of transaction outputs that are available to spend and verifies all transactions.

[6] We note that there is a feather-forking mining strategy to encourage other miners to enforce censorship by attempting to discard their blocks [17].

[7] So far the reward has halved twice on the 28th November 2012 from 50 to 25 bitcoins and the 9th July 2016 from 25 to 12.5 bitcoins.

[8] There is potential instability without a block reward as miners may be incentivised to fork "wealthy" blocks to steal their reward [23].

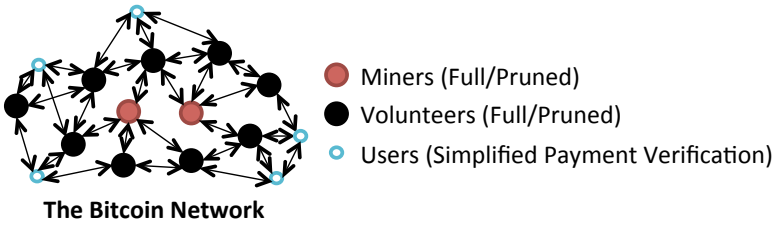


Figure 2.4 Three type of nodes in the peer-to-peer network.

- **Simplified Payment Verification (SPV) node.** Stores the list of block headers, a list of relevant transactions and a list of merkle tree branches that proves each relevant transaction is stored in the Blockchain.

Both full and pruned nodes download and verify the full blockchain which at the time of submission is over 100GB. On the other hand, pruned nodes only maintain a list of transaction outputs that are available to spend and in June 2017 there was approximately 48 million spendable outputs. This type of node can independently verify blocks received from other peers on the network, but there remains a subtle trust assumption in terms of connectivity as this node must be connected to a single and honest peer in order to receive all new blocks with the most proof of work. Otherwise, the full/pruned node may suffer an eclipse attack [54] which allows an adversary to delay which blocks are received.

On the other hand, SPV is designed for embedded devices with low storage and bandwidth capacity (i.e. mobile phones). An SPV node is not required to download or store a full copy of the Blockchain, but will only download and store a chain of block headers (i.e. each header is approximately 80 bytes). The block header allows an SPV node to verify the proof of work associated with each block, verify if a transaction is confirmed in this block (i.e. a merkle tree branch can be used as a proof of inclusion) and whether this block is associated heaviest/longest chain. Similarly to full/pruned nodes, an SPV node must have multiple connections to avoid eclipse attacks.

Applications of Bitcoin. Bitcoin was designed to provide a medium of exchange between two or more parties. It has inspired the community to find new applications for both Bitcoin and the Blockchain which includes exchanging assets [59], carbon dating [27] and authenticating devices in the Internet of Things [58]. These applications leverage the Blockchain as a global database to store data (and cryptographic commitments to data). We highlight that the Bitcoin network cannot independently validate transactions that contain application-specific data using the external application's policy and instead must rely on a third party to ensure the rules are enforced. As a result (and not surprisingly), it is not an ideal platform for many of the desired applications in the community. In the next section, we

present Ethereum that is conceptually a global computer and unlike Bitcoin can enforce both the validation of data and execution of external protocols.

2.2 Ethereum

Ethereum was created by Vitalik Buterin [20], formalised by Gavin Wood [128] and crowd-funded to kick-start development [11]. Their vision was to introduce a global computer that can store and execute programs. These programs are called *smart contracts* as the conditions of an agreement between two or more parties is enforced using the same consensus protocol that secures the Blockchain. Similarly to Bitcoin, it relies upon a *decentralised network of peers* and *governance by a computational competition* to provide an immutable and censorship-resistant Blockchain.[9]

In this section, we briefly outline the inner-workings of Ethereum. We summarise the concept of smart contracts before discussing Ethereum accounts that represent the user's identity on the network. We focus on how a contract's code and its execution instructions are propagated throughout the network using transactions. We highlight that the consensus protocol is claimed to be a variant of the GHOST protocol which is a tree-based blockchain. Finally, we discuss the type of network nodes available on the peer-to-peer network.

Smart contracts are computer programs that are stored on the Blockchain and executed by Ethereum's peer-to-peer network. This concept was first envisioned by Szabo back in 1994 and he provides the following definition [118]:

> *A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.*

Instead of the above definition, an easier mental model for understanding a smart contract is to visualise it as a trusted third party that can only maintain public state. The code of a contract is immutable once deployed, all state transitions are atomic and all function calls are honestly executed. We call this paradigm *consensus computing* as to achieve the above guarantees relies on all network nodes deterministically executing the contract using their

[9] In practice, the Ethereum Foundation has significant influence over Ethereum's governance. For example, a hard-fork recently changed the network's consensus rules to reverse the theft of $40m worth of ether from theDAO hack [24].

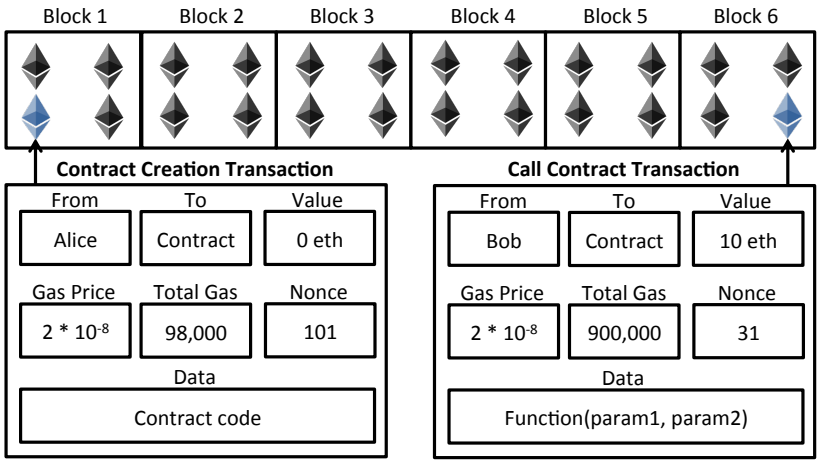


Figure 2.5 Alice creates a smart contract on the Blockchain and the contract code is sent in the transaction's *Data* field. The contract's address is in the *To* field. Bob can also invoke a function of the contract using a second transaction.

copy of the EVM (Ethereum Virtual Machine) in order to reach the same final state. This repetition of computation permits the network to directly enforce the correct execution of a contract and results in publicly verifiable contracts (i.e. any observer can verify a contract's final state by repeating this computation). Unfortunately, this public verifiability is also Ethereum's weakness as a contract's computation must be deterministic and all data used in a contract must be stored publicly [47].

Solidity is a popular high-level programming language for writing smart contracts. This code is compiled into bytecode which is composed of Ethereum-specific opcodes (i.e. operation codes) before it is stored in the Blockchain. The initial state of a contract is set at the time of creation and all future updates to the contract's state is stored in a persistent memory area called storage.[10] Finally contracts are event-driven programs and updating its state requires a user to explicitly invoke one of its functions.[11]

An **Ethereum account** is the user's pseudonymous identity on the network and there are two types of accounts available:

- An **externally owned account (user-controlled)** is a public-private key pair controlled by the user.
- A **contract account** defines an access-control policy using code that is enforced by the network.

Unlike Bitcoin, all Ethereum accounts are directly associated with the in-built ether currency and every payment simply increments the account's balance (as opposed to generat-

[10] A keystore that maps 256 bits to 256 bits.
[11] An Application Binary Interface (ABI) defines how to call a contract's function.

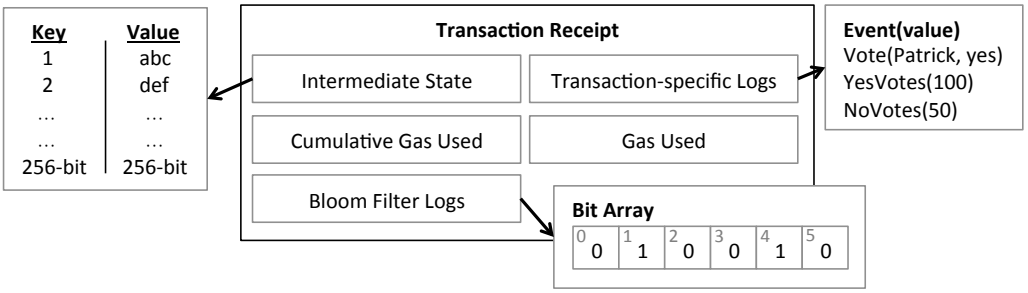


Figure 2.6 A list of fields in the transaction's receipt.

ing a new unspent transaction output). Interestingly, a contract account allows the user to explicitly define access control over their coins which is self-enforced by the network. For example, the contract can establish an escrow service that requires two or more signatures from externally controlled accounts, and require an authenticated data feed[12] to prove an external condition has been met before the ether is accessible.

An **Ethereum Transaction** facilitates sending payments and storing/executing a smart contract on the network. Like Bitcoin, all miners are rewarded via a transaction fee. Figure 2.5 highlights the transaction structure and each field is described below:

- **From:** A signature from an user-controlled account to authorise the transaction.
- **To:** The receiver is either a user-controlled or contract address.
- **Data:** Contains the contract code or its execution instructions.
- **Gas Price:** The conversion rate of purchasing gas using the ether currency.
- **Total Gas:** The maximum amount of gas that can be consumed by the transaction.
- **Nonce:** A counter that is incremented for each new transaction from an account.

The above transaction format facilitates three types of transactions:

- A **payment transaction** allows two parties to send and receive coins.
- A **creation transaction** is responsible for creating the contract's address[13] and storing the contract's bytecode in the Blockchain.
- An **execution transaction** is used to invoke functions and execute the contract.

[12]Zhang et al have demonstrated that trusted hardware can provide trustworthy and authenticated data feeds for smart contracts [132].

[13]This address is a hash of the sender's Ethereum account and the creation transaction's nonce [38].

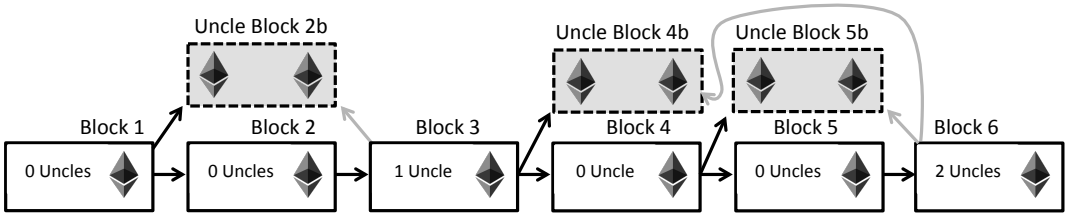


Figure 2.7 Ethereum's blockchain is tree-based as *uncle blocks* that were previously orphaned are now included in the Blockchain.

All transactions depend on a *gas* metric which is self-enforced by the network's consensus rules and determines the exact cost of computation and storage. This gas can be purchased using the network's ether currency and the conversion rate for ether to gas is set by the user in the *gas price* field. It is important the user selects a reasonable conversion rate as the purchased gas is a transaction fee that rewards the miner for including this transaction in their block. There is a subtle issue with Ethereum's execution model (and the estimated gas of execution) as there is no guarantee what computation will occur until the transaction is accepted into the Blockchain. This is possible as the smart contract's state may change after the transaction has already been signed and published for inclusion in the Blockchain. As a result, the exact gas that will be purchased is unknown in advance and the user must update the *total gas* field with the maximum quantity of gas that the user is willing to purchase.

Figure 2.6 highlights that all transactions have a *receipt* that includes the contract's intermediate state, transaction-specific logs, a corresponding bloom filter for the logs and gas statistics. A receipt can be used to convince a low-resourced client that a state transition has occurred, but this assumes the verifier has access to the longest/heaviest chain of block headers. Also, the receipt can be used to notify external protocols about meaningful events that have occurred within the contract. For example, a voting event can trigger the front-end interface to inform the user that their vote was recorded in the contract. Finally the *cumulative gas used* field measures gas used throughout the contract's history and the *gas used* field states the quantity of gas used in this transaction.

Ethereum's blockchain is an orderly-transaction based state machine. If multiple transactions call the same contract, then the contract's final state is determined by the order of transactions that are stored in the block. Strictly, the consensus protocol is claimed to be a variant of the GHOST protocol [112] which is a tree-based blockchain. Miners are rewarded 5 ether for new blocks that are appended to the main branch of the Blockchain and blocks are created approximately every twelve seconds. Unlike Bitcoin and as seen in Figure 2.7, a stale block can later be included in a future block as an uncle block (i.e. it is appended as

a leaf to the Blockchain's main branch).[14] An uncle block's list of transactions is simply discarded and have no effect on a contract/user account's final state. Its sole purpose is to provide a partial reward to miners and the final reward depends on when the uncle block is accepted into the blockchain.

Figure 2.8 highlights that a block has a block header, a list of transactions and a list of uncle block headers. This block header contains the block's position in the Blockchain and statistics on the quantity of gas consumed which must be less than the block's gas limit.[15] The header is also responsible for storing cryptographically binding commitments to the list of transactions, each transaction's receipt and the new global state of all contracts/accounts. As well, here is a bloom filter in the block header to help low-resourced nodes detect if this block contains a specific transaction log. Unlike Bitcoin, the proof of work is based on a memory-hard problem[16] and it is stored in the block header. The Ethereum Foundation plan to eventually change the consensus protocol to a proof of stake protocol called Casper [56]. If successful, then Casper will remove governance by a computational competition and instead allow participants to compete based on their financial stake (i.e. share of the network's coins).

The **peer-to-peer network** is similar to Bitcoin with its open-membership policy as anyone can join and leave the network. Unlike Bitcoin, there are multiple competing implementations of the Ethereum protocol that validates the network as full nodes[17] including *cpp-ethereum*, *Geth* and *Parity*. This thesis focuses on the Geth client as it is used in the proof of concept implementation for Chapter 6. This client is a daemon that runs in the background that can support both full nodes and Light Ethereum Subprotocol (LES) nodes.[18]

Full nodes store a full copy of the Blockchain. Unlike Bitcoin Core, Geth does not offer a *pruned* flag that discards historical blocks and instead it has a *fast* flag that bypasses verifying historical blocks. This involves downloading a full copy of Blockchain, verifying each block's proof of work, downloading a global state of all contracts/accounts for a specified block

[14]Each block can reference up to two uncle blocks that are within six generations. i.e. Block 500 can accept up to two uncle blocks that were created after Block 404.

[15]Each block's gas limit is computed based on the previous block's gas consumption which is defined as Equation 45-47 in [128].

[16]A seed based on each block header is used to compute a 16MB pseudorandom cache. A larger dataset (1GB+) is computed using the cache. Mining involves hashing random slices of the data set. Similarly to Bitcoin, the final solution must be below a desired target value (i.e. the solution will begin with leading zeros.).

[17]Interestingly, Nakamoto's warning about the dangers of multiple implementations [93] came to pass as the Blockchain recently forked due to Geth and Parity failing to identically implement consensus-critical code.

[18]Included in Geth 1.5 on 17th November 2016 [117]

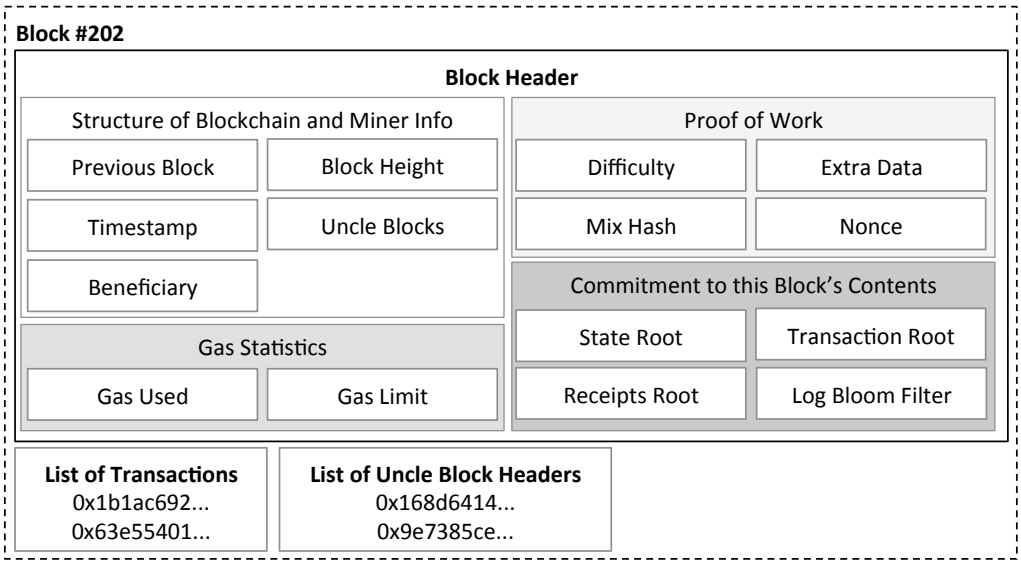


Figure 2.8 An Ethereum block has a list of transactions, a list of uncle block headers, and its own block header.

height and verifying that the global state's hash matches the block header's corresponding state root as seen in Figure 2.8. The downside of this flag is that it assumes the majority of miners honestly enforced the consensus rules for historical blocks. Of course, once the node is fully synchronised it will validate and store all newly created blocks.

LES nodes is designed for low-resourced devices to participate in the network. It involves downloading and verifying only block headers before requesting relevant transactions and receipts from the network. The node can verify that a transaction (or its receipt) is included by checking the block header's cryptographic commitments. It is claimed that the client only requires 10MB (or more) storage, 1 MB/hour bandwidth while idling and 2-3KB bandwidth for each state/storage request from the network [41]. This mode is still in its infancy, but opens the possibility of mobile devices connecting to the peer-to-peer network and in fact packages are readily available for both Android and iOS.

2.3 Conclusion

In this chapter, we presented background information for Bitcoin which is relevant for Chapters 3, 4 and 5, and Ethereum for Chapter 6. We discussed their inner-workings including how users independently compute their pseudonymous credentials, how transactions are used to carry execution instructions to update each peer's replicated database and how their different Blockchain protocols reach consensus. In the next chapter, we study how pseudonymous users in Bitcoin can re-authenticate using a shared transaction history.

Chapter 3

Authenticated Key Exchange over Bitcoin

3.1 Introduction

Bitcoin is increasingly being accepted by many e-commerce websites as a form of payment. For example, Dell, one of the largest computer retailers in the world, now allows customers to use Bitcoin to pay for online purchases on the Dell website [113]. Recently, Paypal [15] and Expedia [103] have also endorsed support for using Bitcoin. Similarly, many community-driven organisations allow anonymous donations using Bitcoin. Examples include the TOR project [120], Mozilla Foundation [89] and the Calyx Institute [83],

While Bitcoin is designed to support anonymity (or pseudonymity) in a transaction, little attention has been paid to the anonymity in the post-payment scenario. As with any on-line payment system, the payer and the payee may need to engage in follow-up correspondence after the payment has been made, e.g., to acknowledge the receipt, to confirm billing information, to amend discrepancies in the order if there are any and to agree on the product delivery or pick-up. Such correspondence can involve privacy-sensitive information, which, if leaked to a third party, may trivially reveal the identity of the user involved in the earlier transaction (e.g., information about product delivery may contain the home address).

Currently, the primary mechanism to support follow-up correspondence after a Bitcoin transaction is through email. The Dell website requires shoppers to provide their email address when making a Bitcoin payment to facilitate post-payment correspondence. The Calyx Institute, a non-profit research organization dedicated to providing "privacy by design for everyone", also recommends using e-mails for follow-up correspondence after a donation is made in Bitcoin. On its website, the instruction is given as the following [83]:

> *"Note that if you make a donation by Bitcoin, we have no way to connect the donation with your email address. If you would like us to confirm receipt of the donation (and send a thank you email!), you'll need to send an email with the details of the transaction. Otherwise, you have our thanks for your support in advance."*

However, emails are merely a communication medium and have no built-in guarantees of security. First of all, there is no guarantee that the sender of the email must be the same person who made the Bitcoin payment. The details of the transaction cannot serve as a means of authentication, since they are publicly available on the Bitcoin network. Furthermore, today's emails are usually not encrypted. The content of an email can be easily read by third parties (e.g., ISPs) during the transit over the Internet. The leakage of privacy-sensitive information in email can seriously threaten the anonymity of the user who has made an "anonymous" payment in Bitcoin previously.

So far the importance of protecting post-payment communication has been largely neglected in both the Bitcoin and the security research communities. To the best of our knowledge, no solution is available to address this practical problem in the real world. This is a gap in the field, which we aim to bridge in our work.

One trivial solution is to apply existing Authenticated Key Exchange (AKE) protocols to establish a secure end-to-end (E2E) communication channel between Bitcoin users. Two general approaches for realising secure E2E communication in cryptography include using 1) PKI-based AKE (e.g., HMQV), and 2) Password-based AKE (e.g., EKE and SPEKE). The former approach would require Bitcoin users to be part of a global PKI system, with each user holding a public key certificate. This is not realistic in current Bitcoin applications. The second approach requires Bitcoin users to have a pre-shared secret password. However, securely distributing pairwise shared passwords over the internet is not an easy task. Furthermore, passwords are a weak form of authentication and they may be easily guessed or stolen (e.g. by shoulder-surfing). A solution that can provide a stronger form of authentication without involving any passwords will be desirable.

Following the decentralised and anonymity-driven nature of the Bitcoin network [74], we propose new AKE protocols to support secure post-payment communication between Bitcoin users, without requiring any PKI or pre-shared passwords. Our solutions leverage the transaction-specific secrets in the confirmed Bitcoin payments published on the public blockchain to bootstrap trust in establishing an end-to-end secure communication channel. Given each party's transaction history and our AKE protocols, both parties are guaranteed to be speaking to the other party who was involved in the transactions, without revealing their real identities.

Algorithm 1 ECDSA Signature Generation algorithm [49]

Input: Domain parameters $D = (q, P, n, \text{Curve})$, private key d, message m.
Output: Signature (r, s).

1: Select $k \in_R [1, n-1]$.
2: Compute $kP = (x_1, y_1)$ where $x_1 \in_R [0, q-1]$
3: Compute $r = x_1 \bmod n$. If $r = 0$, then go to Step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$, then go to Step 1.
6: Return (r, s).

Contributions. Our contributions are summarised below.

- We propose two authenticated key exchange protocols – one interactive and the other non-interactive – using transaction-specific secrets and without the support of a trusted third party to establish end-to-end secure communication. These are new types of AKE protocols, since they bootstrap trust from Bitcoin's public ledger instead of a PKI or shared passwords.
- We provide proof-of-concept implementations for both protocols in the Bitcoin Core client with performance measurements. Our experiments suggest that these protocols are feasible for practical use in real-world Bitcoin applications.

Organization. The rest of the chapter is organised as follows. Section 3.2 explains ECDSA signatures that are used for authenticating Bitcoin transactions. Section 3.3 proposes two protocols to allow post-payment secure communication between users based on their transaction history. One protocol is non-interactive with no forward secrecy, while the other is interactive with the additional guarantee of forward secrecy. Security proofs for both protocols are provided in Section 3.4. Section 3.5 describes the proof-of-concept implementations for both protocols and reports the performance measurements. Finally, Section 3.6 concludes the paper.

3.2 Background

In this section, we provide brief background information about how transactions are signed in Bitcoin, the underlying Elliptic Curve Digital Signature Algorithm (ECDSA) and how implicit authentication for key exchange protocols allows each party to plausibly deny their involvement. This information is required to understand the two key exchange protocols presented in this chapter.

3.2.1 Transaction Signature

Figure 2.1 demonstrates that the signature is stored in the input of a transaction. This signature must be from the Bitcoin address mentioned in the previous transaction's output. Briefly, it is important to highlight that the user will create the transaction, specify the inputs and outputs, hash this transaction and then sign it using their private key. This prevents an adversary from modifying the contents of a transaction or claiming ownership of the bitcoins before it is accepted into the Blockchain.

Bitcoin incorporates the OpenSSL suite to execute the ECDSA algorithm. The NIST-P256 curve is used and all domain parameters over the finite field including group order n, generator P and modulus q can be found in [25]. An outline of the signature generation algorithm is presented in Algorithm 1 to highlight the usage of k as this will be required for the authenticated key exchange protocols. The verification algorithm follows what is defined in [64]. The notations and symbols used in this chapter are summarised in Table 3.1.

3.2.2 Authentication in Key Exchange Protocols

All key exchange protocols either explicitly or implicitly authenticate the long-term public keys which are associated with each party's identity. The subtle distinction between explicit and implicit authentication will determine whether each party can deny their involvement in the key exchange protocol if the transcript is leaked. For example, the Station-to-Station is a popular key exchange protocol which provides explicit authentication. Each party must digitally sign the ephemeral keys used to derive the shared secret κ using their long-term public key. If this transcript is leaked, then the digital signature from their long-term keys provides undeniable proof of involvement. Other key exchange protocols such as MVQ, HMVQ and YAK do not require the parties to explicitly prove knowledge of their long-term public key's corresponding private key during the key establishment (i.e. no messages are digitally signed). Again if the transcript is leaked, then each party can deny their involvement as one party could have simulated the entire protocol without the counter-party's assistance. This plausible deniability property is why we rely on implicit authentication key exchange protocols in the remainder of this chapter.

3.3 Key exchange protocols

Key exchange protocols allow two or more participants to derive a shared cryptographic key, often used for authenticated encryption. In this section we will present two authenticated key exchange protocols: Diffie-Hellman-over-Bitcoin and YAK-over-Bitcoin. These protocols

$ZKP\{w\}$	Zero knowledge proof of knowledge of w
(V,z)	Schnorr zero knowledge proof values
$KDF(.)$	Key derivation function
$Uncompress(x,sign)$	Uncompresses public key using x co-ordinate and $sign \in \{+,-\}$
(x,y)	Represents a point on the elliptic curve
P	Generator for the elliptic curve
(r,s)	Signature pair that is stored in a transaction
A, B	Alice and Bob's bitcoin addresses: $H(dP)$
d_A, d_B	Alice and Bob's private key for their Bitcoin address
k_A, k_B	Alice and Bob's transaction-specific private key
$\widehat{k}_A, \widehat{k}_B$	Alice and Bob's estimated transaction-specific private key
Q_A, Q_B	Alice and Bob's transaction-specific public key
$\widehat{Q}_A, \widehat{Q}_B$	Alice and Bob's estimated transaction-specific public key
w_A, w_B	Alice and Bob's ephemeral private keys used for YAK
κ_{AB}	Shared key for Alice and Bob

Table 3.1 Summary of notations and symbols

will take advantage of a random nonce k from an ECDSA signature. Our aim is to achieve transaction-level authentication by taking advantage of a secret that only exists due to the creation of a transaction that is stored on the Blockchain.

Both of these protocols will use k as a transaction-specific private key and $Q = kP$ as a transaction-specific public key. Diffie-Hellman-over-Bitcoin will be a non-interactive protocol without forward secrecy and YAK-over-Bitcoin will be an interactive protocol with forward secrecy. All domain parameters D for both protocols are the same as the ECDSA algorithm.

3.3.1 Setting the stage

We will have two actors, Alice and Bob. A single transaction T_A is used by Alice to send her payment (anonymously or not) to Bob. For our protocols, we will assume that Bob has created a second transaction T_b using his ECDSA private key, so the Blockchain contains both Alice and Bob's ECDSA signature. This is a realistic assumption as Bob naturally needs to spend the money or re-organise his bitcoins to protect against theft. In one possible implementation, upon receiving Alice's payment, Bob can send back to Alice a tiny portion of the received amount as acknowledgement, so his ECDSA signature is published on the blockchain (the signature serves to prove that Bob knows the ECDSA private key). This is just one way to ensure that the Blockchain contains both actors' signatures, and there may be many other methods to achieve the same.

The owner of a transaction will be required to derive the transaction-specific private key (random nonce) k from their signature before taking part in the key exchange protocols. For

both protocols, we assume the transactions T_A, T_B between Alice and Bob have been sent to the network and accepted to the Blockchain with a depth of at least six blocks, which is considered the standard depth to rule-out the possibility of a double-spend attack.

In both protocols, each user will need to extract their partner's signature (r,s) and attempt to derive their partner's transaction-specific public key $Q = (x,y)$. Algorithm 1 demonstrates that the r value from the signature is equal to the x co-ordinate modulo n (note that there is a subtle difference in the data range, since $r \in Z_n$ and $x \in Z_q$, but this has an almost negligible effect on the working of the protocols as we will explain in detail in Section 3.5.2). However, the y co-ordinate of Q is not stored in the transaction, and it can be either of the two values (above/below the x axis).

We define the uncompression function as $Uncompress(x, sign)$ by using the x co-ordinate from their partner's signature and the y co-ordinate's $sign \in \{+,-\}$. Using point uncompression and assuming one of the two possible signs for the y co-ordinate, Alice or Bob will be able to derive a value $\widehat{Q}$ which we call the estimated transaction-specific public key for their partner. This $\widehat{Q}$ could be either $Q = (x,y)$ or its additive inverse $-Q = (x,-y)$. This $\widehat{Q}$ will correspond to the estimated transaction-specific private key $\widehat{k}$, which could be either k or $-k$.

3.3.2 Authentication

Our definition of authentication will refer to data origin authentication and we will use the Blockchain as a trusted platform for storing digital signatures. Knowledge of the private key d for a bitcoin address or the random nonce k in a signature will prove the identities of pseudonymous parties. We will define two concepts for authentication using Bitcoin:

1. **Bitcoin address authentication**. Knowledge of the discrete log d for a Bitcoin address.

2. **Transaction authentication**. Knowledge of the discrete log k from a single digital signature in a transaction.

Bitcoin address authentication is well-known in the community and has been used for other protocols. However, transaction authentication is a special case that our protocols will exploit. Although k and d are equivalent in proving ownership of a Bitcoin address or transaction, k is randomly generated for every ECDSA signature and is unique for each new transaction.

We will show that Alice and Bob can authenticate each other based on the knowledge of the k. This relies on participants trusting the integrity of the Blockchain as the cornerstone for authentication. For an adversary to mount a man-in-the-middle attack in this scene, he

	Blockchain contains (r_A, s_A) *and* (r_B, s_B) *from* T_A *and* T_B	
	Alice (A, d_A)	**Bob** (B, d_B)
1.	$k_A = (H(T_A) + d_A r_A) s_A^{-1}$	$k_B = (H(T_B) + d_B r_B) s_B^{-1}$
2.	$\widehat{Q}_B = Uncompress(r_B, +)$	$\widehat{Q}_A = Uncompress(r_A, +)$
3.	$k_A \widehat{Q}_B = (x_{AB}, \pm y_{AB})$	$k_B \widehat{Q}_A = (x_{AB}, \pm y_{AB})$
	$\kappa = \mathrm{KDF}(x_{AB})$	$\kappa = \mathrm{KDF}(x_{AB})$

Figure 3.1 The Diffie-Hellman-over-Bitcoin Protocol

would need to perform a history-revision attack to modify the ECDSA signatures stored in the Blockchain.

3.3.3 Diffie-Hellman-over-Bitcoin Protocol

Based on the concept of transaction authentication, the first protocol that we present is 'Diffie-Hellman-over-Bitcoin'. The protocol is non-interactive; the shared secret is generated using the signatures from two transactions and no additional information from the participants is required. However, forward secrecy is not provided, as we will illustrate in the security analysis.

Figure 3.1 presents an outline of the protocol. Initially, each user will derive the random nonce k from their own signatures and fetch their partner's transaction from the Blockchain. Each user will gain an estimation of their partner's public key $\widehat{Q}$ before using their own transaction-specific private key k to derive the shared secret $(x_{AB}, \pm y_{AB})$. Regardless of whether $\widehat{Q}_A = \pm Q_A$ (or $\widehat{Q}_B = \pm Q_B$), the x co-ordinate of $k_B \widehat{Q}_A$ will be the same as that of $k_A \widehat{Q}_B$. Following the Elliptic Curve Diffie Hellman (ECDH) [85] approach, the x_{AB} co-ordinate will be used to derive the key $KDF(x_{AB}) = \kappa$.

3.3.4 YAK-over-Bitcoin Protocol

The second protocol we present is 'YAK-over-Bitcoin'. This is based on adapting a PKI-based YAK key exchange protocol [50] to the Bitcoin application by removing the dependence on a PKI and instead relying on the integrity of the Blockchain. We chose YAK instead of others (e.g., station-to-station, MQV, HMQV, etc), as YAK is the only PKI-based AKE protocol that requires each sender to demonstrate the proof of knowledge of both the static and ephemeral private keys. This requirement is important for the security proofs of our system as we will detail in Section 3.4.1. As well, we will show in the security analysis that the protocol allows the participants to have full forward secrecy.

	Blockchain contains (r_A, s_A) *and* (r_B, s_B) *from* T_A *and* T_B		
	Alice (A, d_A)		**Bob** (B, d_B)
1.	$k_A = (H(T_A) + d_A r_A) s_A^{-1}$		$k_B = (H(T_B) + d_B r_B) s_B^{-1}$
2.	$Q_A = (r_A, y_A) = k_A P$		$Q_B = (r_B, y_B) = k_B P$
3.	$\widehat{Q}_A = Uncompress(r_A, +)$		$\widehat{Q}_B = Uncompress(r_B, +)$
	If $Q_A = \widehat{Q}_A$ then $\widehat{k}_A = k_A$		If $Q_B = \widehat{Q}_B$ then $\widehat{k}_B = k_B$
	else $\widehat{k}_A = -k_A$		else $\widehat{k}_B = -k_B$
4.	$\widehat{Q}_B = Uncompress(r_B, +)$		$\widehat{Q}_A = Uncompress(r_A, +)$
5.	$w_A \in_R [1, n-1]$, $W_A = w_A P$	$\xrightarrow{W_A, \text{ZKP}\{w_A\}}$	$w_B \in_R [1, n-1]$, $W_B = w_B P$
		$\xleftarrow{W_B, \text{ZKP}\{w_B\}}$	
6.	Verify ZKP$\{w_B\}$		Verify ZKP$\{w_A\}$
7.	$(x_{AB}, y_{AB}) =$ $(\widehat{k}_A + w_A)(\widehat{Q}_B + W_B)$ $\kappa = \text{KDF}(x_{AB})$		$(x_{AB}, y_{AB}) =$ $(\widehat{k}_B + w_B)(\widehat{Q}_A + W_A)$ $\kappa = \text{KDF}(x_{AB})$

Figure 3.2 YAK-over-Bitcoin Protocol

An outline of our protocol is presented in Figure 3.2. Initially, each user will follow the same steps as seen in the previous 'Diffie-Hellman-over-Bitcoin' protocol to derive their secret k and their partner's estimated public key $\widehat{Q}$. However, a subtle difference requires each user to compare their real public key Q with the estimation of their own key $\widehat{Q}$ to determine if they are equal. If these public keys are different, then the user will use the additive inverse of k as their estimated transaction-specific private key and we will denote this choice between the two keys as $\widehat{k}$. This subtle change will allow both parties to derive the same shared secret (x_{AB}, y_{AB}) which would be expected in an interactive protocol without exchanging their real y co-ordinates.

Each user generates an ephemeral private key w and computes the corresponding public key $W = wP$. As required in the original YAK paper [50], each user must also construct a zero knowledge proof to prove possession of the ephemeral private key w. These zero knowledge proofs can be sent over an insecure communication channel to their partners. Here, we will use the same Schnorr signature as in [50] to realise the ZKP. Details of the Schnorr ZKP are summarised in Algorithm 2 and 3. The definition of the Schnorr ZKP includes a unique signer identity ID, which prevents an attacker replaying the ZKP back to the signer herself [50]. In our case, we can simply use the unique r value from the user's ECDSA signature (r, s) in the associated Bitcoin transaction T as the user's identity.

Once the ZKPs have been verified, each user will derive (x_{AB}, y_{AB}) using their secret $w, \widehat{k}$, public value W and their partners' estimated transaction-specific public key $\widehat{Q}$. It should be easy to verify that although the shared secret has four different combinations

Algorithm 2 Schnorr Zero Knowledge Proof Generation Algorithm

Input: Domain parameters $D = (q, P, n, \text{Curve})$, signer identity ID, secret value w and public value W.
Output: (V, z)

1: Select $v \in_R [1, n-1]$, Compute $V = vP$
2: Compute $h = H(D, W, V, ID)$
3: Compute $z = v - wh \mod n$
4: Return (V, z)

Algorithm 3 Schnorr Zero Knowledge Proof Verification Algorithm

Input: Domain parameters $D = (q, P, n, \text{Curve})$, signer identity ID, public value W, Schnorr zero knowledge proof values (V, z)
Output: Valid or invalid

1: Perform public key validation for W [64]
2: Compute partners $h = H(D, W, V, ID)$
3: Return $V \stackrel{?}{=} zP + hW \mod n$

$(\pm\widehat{k}_A + w_A)(\pm\widehat{k}_B + w_B)P$, the secret key derived between Alice and Bob will always be identical (due to each participant predicting the estimated public key $\widehat{Q}$ that their partner will choose).

3.4 Security Analysis

Our protocols are based on reusing the signature-specific random value k in ECDSA as the transaction-specific secret on which the authenticated key exchange protocol is based. Hence, the security of both the ECDSA signature and the key exchange protocols needs to be analysed to make sure the reusing of k is sound in terms of security.

For the AKE protocols, following the security analysis of YAK [50], we consider three security requirements, informally defined in the following:

- **Private key security:** The adversary is unable to gain any *extra*[1] information about the private key of an honest party by eavesdropping her communication with other parties, changing messages sent to her, or even participating in an AKE protocol with her.

[1] By "extra" information, we mean information other than what is derivable from the honest party's already available public key.

- **Full forward secrecy:** The adversary is unable to determine the shared secret of an eavesdropped AKE session in the past between a pair of honest parties, even if their private keys are leaked subsequently.
- **Session key security:** The adversary is unable to determine the shared secret between two honest parties by eavesdropping their communication or changing their messages.

Note that in our security arguments we consider the security of *shared secrets* (x_{AB} in Fig's 3.1 and 3.2), as opposed to that of the subsequently calculated shared *session keys* (κ in the same figures). We henceforth denote the shared secret by K, i.e., $\kappa = \mathrm{KDF}(K)$. We require the shared secret to be hard to determine for the adversary in the full forward secrecy and session key security requirements. A good key derivation function (KDF) derives from such a shared secret a session key which is indistinguishable from random. Our security proofs can be easily adapted to prove indistinguishability based on the decisional rather than computational Diffie-Hellman assumption.

For ECDSA signature, we require that it remains unforgeable against chosen-message attacks despite the randomness k being reused in subsequent AKE protocols. Although ECDSA has withstood major cryptanalysis, the security of ECDSA has only been proven under non-standard assumptions or assuming modifications (see [125] for a survey of these results). In our analysis, we consider extra information available to an attacker as a result of k being reused in AKE protocols, and show that it does not degrade the security of ECDSA.

We assume ECDSA to be a (non-interactive honest-verifier) zero-knowledge proof of knowledge of the private key d. This is a reasonable assumption in the random oracle model which follows the work of Malone-Lee and Smart [73][2]. In practice, people accept bitcoin transactions only when the ECDSA signatures are verified successfully. Verifying the ECDSA signature is tantamount to verifying the knowledge of the ECDSA private key d that should only be held by the legitimate bitcoin user.

We also note that given an ECDSA message-signature pair, $m,(r,s)$, knowledge of the private key d is equivalent to knowledge of the randomness k since given either the other can be calculated from $sk = H(m) + dr \mod n$.

[2]Note that the results apply to a slightly modified version of ECDSA in which $e = H(r|m)$ where $|$ denotes concatenation. Although the Bitcoin Core implementation is based on the original ECDSA standard, the above modification is included in more recent standards of ECDSA such as ISO/IEC 14888 [63]. Furthermore, as another option for signing, the Bitcoin community is considering including Schnorr signature [6], which is proven to be a zero-knowledge proof of knowledge of the private key.

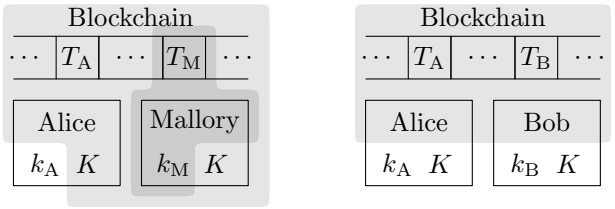


Figure 3.3 Security of Diffie-Hellman-over-Bitcoin. Light grey denotes what the adversary (Mallory in (i), Eve in (ii)) *knows*. Dark grey denotes what the adversary (Mallory) *chooses*.

3.4.1 Security of Diffie-Hellman-over-Bitcoin

This protocol is an Elliptic Curve Diffie-Hellman key exchange and the public values are bound to two transactions in the Blockchain. Private key security considers a malicious active adversary "Mallory", and session key security considers an eavesdropper adversary "Eve". The protocol does not provide full forward secrecy. We will provide a sketch of the proof of security for each property in the following. In each proof, we follow the same approach as in [50] to assume an extreme adversary, who has all the powers except those that would allow the attacker to trivially break any key exchange protocol.

Theorem 1 (Private Key Security)**.** *Diffie-Hellman-over-Bitcoin provides private key security under the assumption that ECDSA signature is a zero knowledge proof of knowledge of the ECDSA secret key.*

sketch. The goal of an adversary Mallory is to be able to gain some *extra* information on Alice's transaction-specific private key k_A through the following attack. Mallory is given the public parameters of the system and access to the Blockchain which includes Alice's transaction T_A, then she provides a transaction T_T which is included in the Blockchain, then she carries out a Diffie-Hellman-over-Bitcoin protocol with Alice (which is non-interactive), and eventually is able to calculate the shared secret K. The attack is depicted in Fig. 3.3(i). Alice's ECDSA signature in T_A is assumed to be zero knowledge and hence does not reveal any information about her private key. Furthermore, since Mallory's transaction T_T includes an ECDSA signature by her, and ECDSA signature is a proof of knowledge of Mallory's ECDSA secret key d_T, Mallory must know d_T, and hence k_T. Hence, Mallory does not gain any extra knowledge from calculating K, since knowledge of k_T and Alice's public key enables her to simulate K on her own. □

Theorem 2 (Session Key Security)**.** *Diffie-Hellman-over-Bitcoin provides session key security based on the computational Diffie-Hellman assumption under the assumption that ECDSA signature is a zero knowledge proof of knowledge of the ECDSA secret key.*

sketch. Assume there is a successful adversary Eve that is able to calculate the shared secret K for a key exchange between two honest parties Alice and Bob, without knowing either Alice or Bob's transaction-specific secret keys, k_A or k_B. The attack is depicted in Fig. 3.3(ii). Note that since the protocol is non-interactive, the adversary is reduced to a passive adversary. A successful attack would contradict the computational Diffie-Hellman (CDH) assumption since given an instance of the CDH problem $(P, \alpha P, \beta P)$, one is able to leverage Eve and solve the CDH problem by setting up Alice and Bob's transaction-specific secrets as $k_A = \alpha$ and $k_B = \beta$, which results in $K = \alpha\beta P$. A successful Eve implies that CDH can be solved efficiently. □

3.4.2 Security of YAK-over-Bitcoin

This protocol is an Elliptic Curve YAK key exchange and the public values are bound to two transactions in the Blockchain. Private key security and session key security consider a malicious active adversary "Mallory", and full forward secrecy considers an eavesdropper adversary "Eve". Similar as before, we assume an extreme adversary who has all the powers except those that would trivially allow the attacker to break any key exchange protocol. Under this assumption, we provide a sketch of the proof of security for each property in the following.

Theorem 3 (Private Key Security)**.** *YAK-over-Bitcoin provides private key security under the assumption that ECDSA signature is a zero knowledge proof of knowledge of the ECDSA secret key.*

Proof (sketch). The goal of an adversary Mallory is to be able to gain some *extra* information on Alice's transaction-specific private key k_A through the following attack. Mallory is given the public parameters of the system and access to the Blockchain which includes Alice's transaction T_A, then she provides a transaction T_T which is included in the Blockchain, then she carries out a YAK-over-Bitcoin protocol with Alice, in which Alice sends the message $(w_A P, \text{ZKP}\{w_A\})$ and Mallory sends the message $(w_T P, \text{ZKP}\{w_T\})$. Alice's ephemeral secret w_A is also assumed to be leaked to Mallory. The attack is depicted in Fig. 3.4(i). Alice's ECDSA signature in T_A is assumed to be zero knowledge and hence does not reveal any information about her private key. Furthermore, since the ECDSA signature in Mallory's transaction and her message in the protocol are proofs of knowledge of d_T (equivalently k_T) and w_T, respectively, Mallory must know both k_T and w_T. Note that she receives $(w_A P, \text{ZKP}\{w_A\})$ and w_A and hence will be able to calculate $K = (k_T + w_T)(k_A P + w_A P)$. Hence, Mallory does not gain any extra knowledge from the values she receives, since

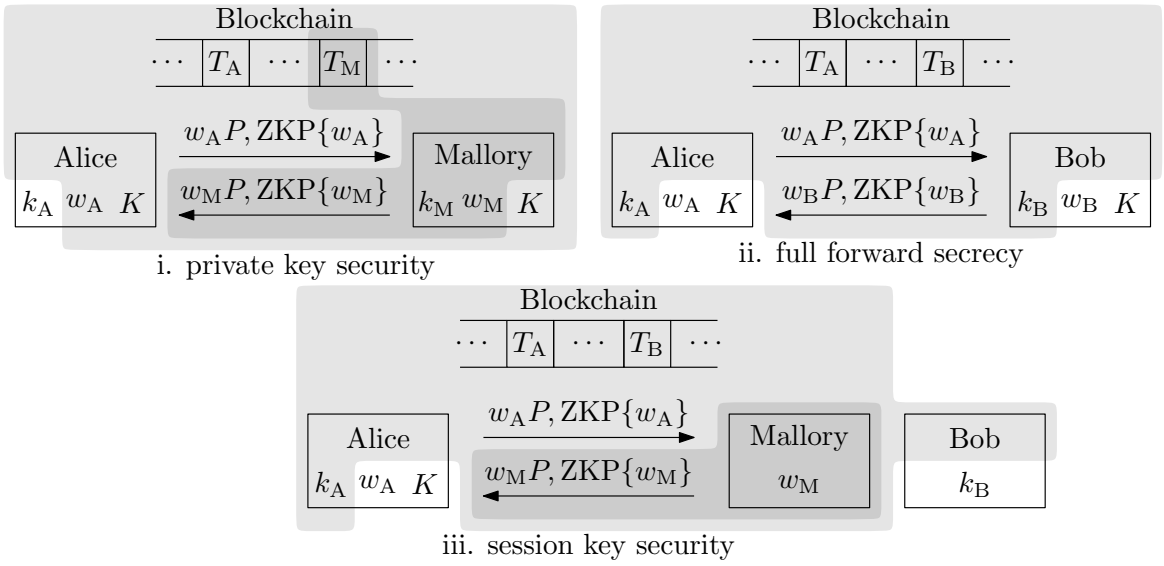


Figure 3.4 Security of YAK-over-Bitcoin. Light grey denotes what the adversary (Mallory in (i) and (iii), Eve in (ii)) *knows*. Dark grey denotes what the adversary (Mallory) *chooses*.

w_A is independent of k_A and knowledge of w_A, k_T, and w_T enables Mallory to simulate all the values she receives. □

Theorem 4 (Full Forward Secrecy)**.** *YAK-over-Bitcoin provides full forward secrecy based on the computational Diffie-Hellman assumption.*

Proof (sketch). Assume there is a successful adversary Eve that is able to calculate the shared secret K for a previous key exchange between two honest parties Alice and Bob through the following attack. Both Alice and Bob's transaction-specific secret keys k_A and k_B are assumed to be leaked to Eve. Eve is also assumed to have access to all the protocol messages exchanged between Alice and Bob, as well as the Blockchain of course. The attack is depicted in Fig. 3.4(ii). Given an instance of the CDH problem $(P, \alpha P, \beta P)$ one is able to leverage Eve and solve the problem as follows. The protocol is set up with the ephemeral secret values $w_A = \alpha$ and $w_\mathrm{B} = \beta$ and all other parameters as per the protocol description. When Eve calculates K, the value $S = K - k_A k_\mathrm{B} P - k_A(\beta P) - k_\mathrm{B}(\alpha P)$ is calculated and returned as the solution to the CDH problem. Note that since $K = (k_A + w_A)(k_\mathrm{B} + w_\mathrm{B})P$, we have $S = \alpha\beta P$. A successful Eve implies that CDH can be solved efficiently. □

Theorem 5 (Session Key Security)**.** *YAK-over-Bitcoin provides session key security based on the computational Diffie-Hellman assumption under the assumption that ECDSA signature is a zero knowledge proof of knowledge of the ECDSA secret key.*

Proof (sketch). Assume there is a successful adversary Mallory that is able to calculate the shared secret K for a key exchange between two honest parties Alice and Bob through

the following attack by impersonating Bob to Alice. Alice believes she is interacting with Bob, whereas in reality she is interacting with an impersonator Mallory who replaces Bob's message in the protocol with her own $(w_{\mathrm{T}}P, \mathrm{ZKP}\{w_{\mathrm{T}}\})$. Alice's transaction-specific secret key k_A is assumed to be leaked to Mallory as well. However, Mallory does not know Bob's transaction-specific secret key k_{B}. The attack is depicted in Fig. 3.4(iii). Given an instance of the CDH problem $(P, \alpha P, \beta P)$ one is able to leverage Mallory and solve the problem as follows. The protocol is set up with Alice's ephemeral secret $w_A = \alpha$ and Bob's transaction-specific secret $k_{\mathrm{B}} = \beta$ and all other parameters as per the protocol description. When Mallory calculates K, the value $S = K - k_A w_{\mathrm{B}} P - w_A(\beta P) - w_{\mathrm{B}}(\alpha P)$ is calculated and returned as the solution to the CDH problem. Note that since $K = (k_A + w_A)(k_{\mathrm{B}} + w_{\mathrm{B}})P$, we have $S = \alpha\beta P$. A successful Mallory implies that CDH can be solved efficiently. □

3.4.3 Security of ECDSA Signatures

Diffie-Hellman-over-Bitcoin is a non-interactive protocol and the protocol participants do not send any messages to each other that would potentially have an impact on the security of ECDSA signatures.

In 'YAK-over-Bitcoin', the messages that the protocol participants send each other include information about their ephemeral keys w_A and w_{B} only, which are chosen independently of all the secret values related to the ECDSA signatures in T_A and T_{B}. As shown in Theorem 3 in Section 3.4.2, the protocol does not reveal any information about the static private key (i.e., k), and hence not any information about the ECDSA private key (i.e., d) since the two values are linearly related. One can compute d from k, or vice versa. The key element in the proof of Theorem 3 is that each party is required to demonstrate knowledge of both the static and ephemeral keys. This also explains our choice of the YAK protocol, as YAK is the only PKI-based AKE protocol that has the requirement that each party must demonstrate the proof of knowledge for both the static and ephemeral keys (the former is realized by the Proof of Possession at the Certificate Authority registration while the later is achieved by Schnorr Non-interactive ZKP).

3.5 Implementation

Our implementation is a modification of the Bitcoin Core client and is considered a proof of concept. We have included three new remote procedure commands (RPC) that will allow the client to perform a non-interactive Diffie-Hellman key exchange, generate a zero knowledge proof to be shared with their partner and verify a partner's zero knowledge proof before

Step	Description	Time
	Diffie-Hellman-over-Bitcoin	
1-2	Compute k_A and $\widehat{Q}_{\mathrm{B}}$	0.08 ms
3	Compute shared secret K_{AB} and key κ_{AB}	0.51 ms
	Total:	0.59 ms
	YAK-over-Bitcoin	
1-4	Compute $k_A, Q_A, \widehat{Q}_A$ and $\widehat{Q}_{\mathrm{B}}$	0.53 ms
5	Compute w_A, W_A and ZKP$\{w_A\}$	0.90 ms
6	Verify Bob's ZKP$\{w_{\mathrm{B}}\}$	0.69 ms
7	Compute shared secret K_{AB} and key κ_{AB}	0.43 ms
	Total:	2.55 ms

Table 3.2 Alice performing YAK-over-Bitcoin

revealing the shared secret. Our modified implementation was executed using the -txindex parameter which allows us to query the Blockchain and retrieve the raw transaction data.

Two transactions were created using a non-modified implementation on the 10th December, 2013 to allow us to test our key exchange on the real network. All tests were carried out a MacBook Pro mid-2012 running OS X 10.9.1 with 2.3GHz Intel Core i7 and 4 cores and 16 GB DDR3 RAM. Each protocol is executed 100 times from Alice's perspective and the average times are reported.

3.5.1 Time analysis

Preliminary steps for both protocols involve fetching the transactions from the Blockchain 0.04 ms and retrieving the signatures (r,s) stored in the transaction 0.08 ms. Overall, these steps on average require 0.12 ms.

This 'Diffie-Hellman-over-Bitcoin' protocol is non-interactive as participants are not required to exchange information before deriving the shared secret. Table 4.1 demonstrates an average time of 0.08 ms to derive Alice's transaction-specific private key k_A and Bob's estimated public key $\widehat{Q_{\mathrm{B}}}$ and 0.051 ms to compute the shared key κ_{AB}. Overall, on average the protocol executes in 0.59 ms which is reasonable for real-life use.

The 'YAK-over-Bitcoin' protocol is interactive as it requires each party to send an ephemeral public key together with a non-interactive Schnorr ZKP to prove the knowledge of the ephemeral private key. Table 4.1 shows that computing and verifying zero knowledge proofs is the most time-consuming operation. However, a total execution time of 2.55 ms is still reasonable for practical applications.

3.5.2 Note about domain parameters

Our investigation highlighted that $q > n$ as seen in [25] which could obscure the relationship between k and r as the x co-ordinate can wrap around n. However, the probability that this may occur can be calculated as $(q-n)/q \approx 4 \times 10^{-39}$ and is unlikely to occur in practice. However, in the rare chance that this does happen then it is easily resolved by $r' = r+n$. This does not require any modification to the underlying signature code as it is simply an addition of the publicly available r with the modulus n. Once resolved, both parties can continue with the protocol. For reference, q and n are defined below:

```
q=FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
n=FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
```

3.6 Conclusion

We proposed two protocols to allow for interactive and non-interactive key exchange, the latter offering an additional property of forward-secrecy. We encourage the community to try our proof-of-concept implementation and to take advantage of this new form of authentication to enable end-to-end secure communication between Bitcoin users. In the next chapter, we explore how the community accepted standard BIP70: Payment Protocol does not rely on Bitcoin address authentication. This results in two new attacks as the merchant cannot verify if the pseuodnymous customer endorsed the refund address received during the protocol.

Chapter 4

Refund attacks on Bitcoin's Payment Protocol

4.1 Introduction

In the previous chapter, we considered the scenario where both parties (e.g. a customer and merchant) require an authenticated and end-to-end communication channel to support post-payment correspondence. This led us to evaluate how merchants were handling the pre-payment correspondence to allow customers to authenticate the merchant's payment address and how the merchant authenticates the customers refund address in the event of a dispute. We discovered in 2014 that more than 100,000 merchants used the infrastructure provided by Payment Processors such as BitPay and Coinbase to receive bitcoin payments.

Both Payment Processors and all merchants implement the community accepted BIP70: Payment Protocol standard that was proposed by Andresen and Hearn [8]. The motivation for this protocol is to reduce the complexity of Bitcoin payments as customers are no longer required to handle Bitcoin addresses[1]. Instead, the customer can verify the merchant's identity using a human-readable name before authorising a payment. At the time of a payment authorisation, the customer's wallet will also send a refund Bitcoin address to the merchant that should be used in the event of a future refund.

The Payment Protocol provides two pieces of evidence that can be used in case of a dispute with an arbitrator. The customer has evidence that they were requested to authorise a payment if they keep a copy of the signed *Payment Request* message. This can be considered evidence as the customer could not have produced the signature without the co-operation of

[1] A form of identity (26–35 alphanumeric characters) that is related to a public-private key pair and is used to send/receive bitcoins.

the merchant. The second piece of evidence for both the customer and the merchant is the payment transaction as the payment is signed by the customer and time-stamped on Bitcoin's Blockchain, that is stored by most users of the network. In this chapter, we argue that a third piece of evidence is required to authenticate the refund address sent from the customer as the protocol recommends the customer's payment and refund addresses not be the same. While the authenticated key exchange protocols in the previous chapter could be used to receive a refund address during post-payment correspondence, we propose that the third piece of evidence can be incorporated in the pre-payment correspondence as desired by the Payment Protocol. Without this evidence the following attacks are possible:

- The *Silkroad Trader* attack relies on a vulnerability in the Payment Protocol as the customer can authenticate that messages originate from the merchant, but not vice-versa. This allows a customer to route payments to an illicit trader via a merchant and then plausibly deny their own involvement.
- The *Marketplace Trader* attack focuses on the current refund policies of Coinbase and BitPay who both accept the refund address over e-mail [123][28]. This allows a rogue trader to use the reputation of a trusted merchant to entice customers to fall victim to a phishing-style attack.

Without the knowledge of our attacks, Schildbach asked the Bitcoin-Development mailing list why the refund address in the Payment Protocol was currently unprotected [110] and one of the original authors responded:

> *We talked about signing it with one of the keys that's signing the Bitcoin transaction as well. But it seems like a bit overkill. Usually it'll be submitted over HTTPS or a (secured!) Bluetooth channel though so tampering with it should not be possible. - Mike Hearn [53]*

As seen above, a solution may involve the customer endorsing a refund address using one of the public keys that authorised the transaction. However, the author stated this solution was an overkill as the refund address is currently protected in an HTTPS communication channel. We demonstrate that the HTTPS communication channel cannot protect the refund address as it only provides one-way authentication, the customer can authenticate messages originated from the merchant, but not vice-versa. At first glance, the 'overkill' solution suggested by Hearn could provide the evidence required for the merchant. Unfortunately, this solution opens the door to another attack which allows a malicious co-signer the sole authority to endorse the refund address used by the merchant and thus steal the bitcoins of other co-signers. We will discuss this in detail in Section 4.5.

Contributions. Our contributions in this chapter are summarised below:

- We present new attacks on Bitcoin's Payment Protocol and the current practice of both the Payment Processors,
- We present real-world experiments that demonstrate how merchants today are vulnerable to both attacks using a modified Bitcoin wallet.
- We propose a solution that removes the incentive to perform both attacks as the merchant is provided with publicly verifiable evidence whose origin can be verified by an arbitrator.

4.2 Background

Background information on Bitcoin can be found in Chapter 2.1. In this section we present the community accepted Payment Protocol standard.

4.2.1 Payment Protocol

Andresen and Hearn proposed the Payment Protocol which has been accepted as a standard in BIP70 [8] and is supported by several prominent wallets. The goal of this protocol is described in the standard as the following:

> *"This BIP describes a protocol for communication between a merchant and their customer, enabling both a better customer experience and better security against man-in-the-middle attacks on the payment process."*

Communication between the customer and merchant is sent over HTTPS[2] and importantly, the customer is also responsible for broadcasting the payment transaction to the Bitcoin network. In this HTTPS setting, the merchant must have an X.509 certificate issued by a trusted Certificate Authority. This is necessary to let the customer authenticate messages from the merchant.

Figure 4.1 outlines the messages exchanged and actions performed for the protocol. To initiate, the customer clicks the 'Pay Now' button on the merchant's website to generate a Bitcoin URI. This URI opens the customer's Bitcoin wallet and downloads the *Payment Request* message from the merchant's website. The wallet verifies the digital signature for

[2]The protocol specification allows messages to be sent over HTTP and for the merchant not to have an X.509 certificate, but this is not considered secure.

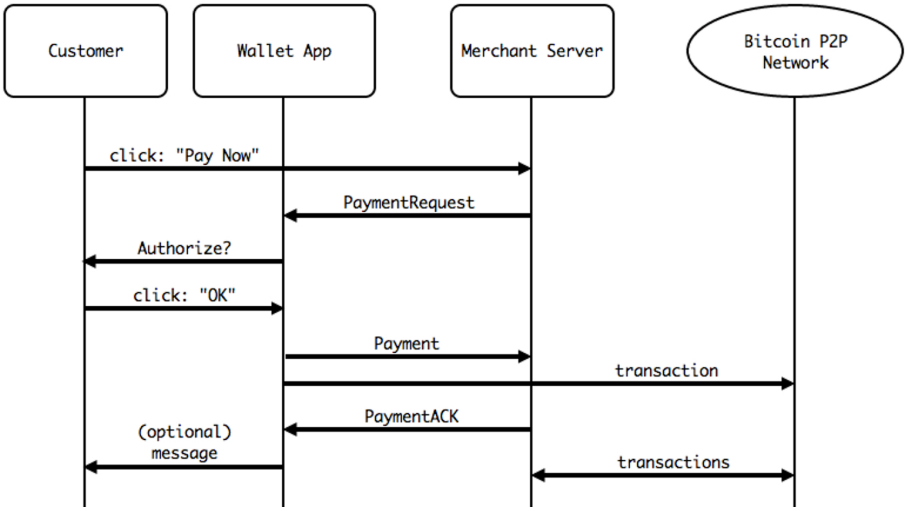


Figure 4.1 Overview of the Payment Protocol [8]

the message using the public key found in the merchant's X.509 certificate (and checks the merchant's certificate for authenticity using the operating system's list of root certificate authorities). A human-readable name for the merchant[3] and the number of requested bitcoins is displayed on-screen and the customer must check this information before clicking 'Send'. Upon authorisation, the wallet performs two actions:

1. The customer's wallet sends one or more payment transactions to the Bitcoin network.
2. The *Payment* message which includes the payment transactions and refund addresses is sent to the merchant's website.

The merchant responds to the customer's *Payment* message with a *Payment Acknowledgement* message which notifies the customer's wallet to display a confirmatory 'Thank you' message. Furthermore, once the merchant has detected the payment transaction on the Bitcoin network, the customer's web browser is refreshed to display a confirmation page. For the rest of this chapter, we focus on messages sent over the HTTPS communication channel as seen in Figure 4.1 and for simplicity we assume the customer only sends a single payment transaction. The content for each message is the following:

- The **Payment Request** message contains a unique payment address B, requested number of bitcoins ฿, creation time for request t_1, expiry time for request t_2, a memo message m_B, a payment URL u_B and some merchant-specific data to link any future payments z_B. The contents of this message is signed using the private

[3]URL from the the X.509 certificate's 'common name' field.

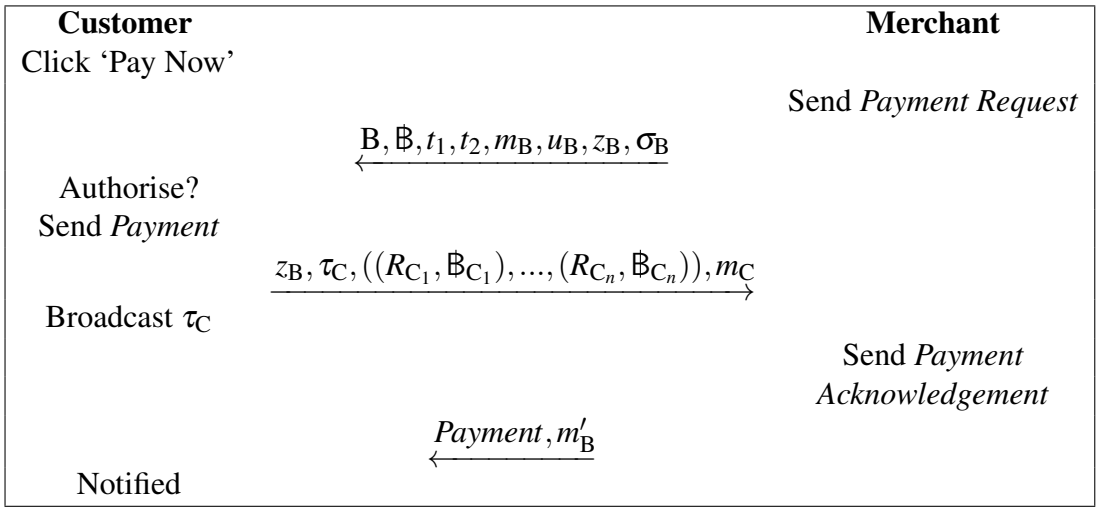


Figure 4.2 Message contents for the Payment Protocol

key xsk_{B} that corresponds to the merchant's X.509 certificate public key such that $\sigma_{\mathrm{B}} = S_{xsk_{\mathrm{B}}}(\mathrm{B}, ฿, t_1, t_2, m_{\mathrm{B}}, u_{\mathrm{B}}, z_{\mathrm{B}})$ where S is the signature algorithm.

- The **Payment** message contains a repeat of the merchant-specific data z_{B}, a payment transactions τ_{C}[4], a list of refund addresses $(R_{\mathrm{C}_1}, ..., R_{\mathrm{C}_n})$ and the number of bitcoins ฿ that should be refunded to each address such that $((R_{\mathrm{C}_1}, ฿_1), ..., (R_{\mathrm{C}_n}, ฿_n))$ and a memo from the customer m_{C}. There is no restriction to the number of refund addresses sent to the merchant and the customer is responsible for deciding how the bitcoins are refunded amongst the refund addresses provided. Merchants expect one or more *Payment* messages until all requested bitcoins have been received.

- The **Payment Acknowledgement** message is a repeat of the customer's *Payment* message and includes an optional memo m'_{B} from the merchant.

As seen in Figure 4.1, the refund address sent in the *Payment* message is not digitally signed by the customer and its integrity relies on the HTTPS communication channel established between the customer and the merchant to prevent man-in-the middle attacks. This lack of mutual authentication in the Payment Protocol and the refund policy of both Payment Processors to accept refund addresses over e-mail enables the attacks outlined in the next section.

[4] A single payment transaction τ_{C} is considered for simplicity. The protocol supports one or more payment transactions, and our results still apply in this case.

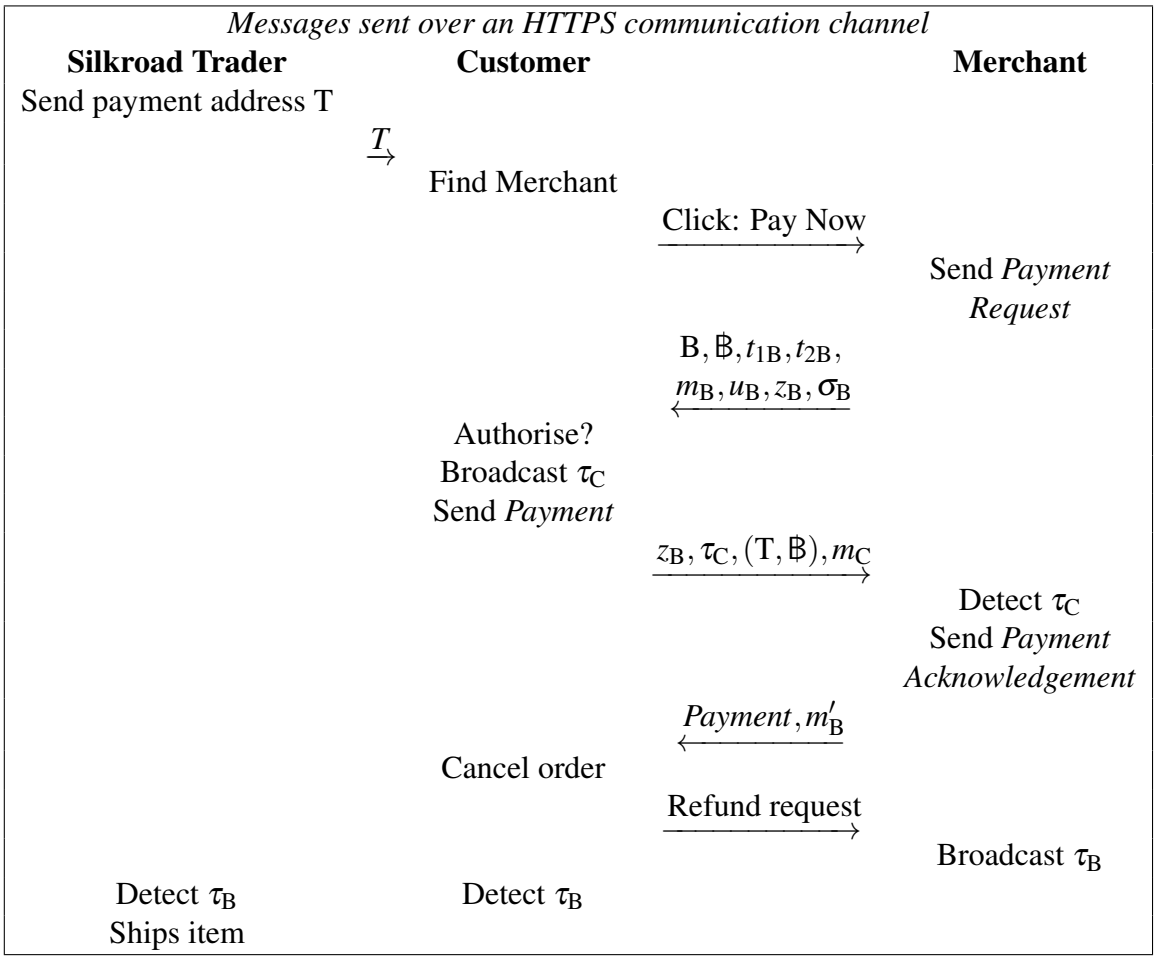


Figure 4.3 *Silkroad Trader* attack allows a customer to route bitcoins to an illicit trader via an honest merchant and then plausibly deny their involvement.

4.3 Attacking the Payment Protocol

In this section, we outline attacks which are feasible due to an authentication vulnerability in the Payment Protocol and the refund policy of both Payment Processors. Fundamentally, our attacks rely on the merchant's inability to distinguish if the refund address originated from the same pseudonymous customer that authorised the payment. As well, these attacks are successful even when all messages are sent over an HTTPS communication channel.

4.3.1 Silkroad Trader Attack

This attack allows a customer to route payments to an illicit trader via an honest merchant and then plausibly deny their own involvement in the refund transaction. The main idea behind this attack relies on the customer's ability to swap the refund address in their *Payment*

message with a Bitcoin address under the control of an illicit trader. Most importantly, the customer is not required to endorse the illicit trader's Bitcoin address with a digital signature.

Figure 4.3 is an outline of the Silkroad Trader attack. It begins with the customer finding an 'illicit good' to purchase from a merchant on Silkroad and receiving the illicit trader's payment address T. Next the customer finds a merchant who supports the Payment Protocol and has an item listed with approximately equal (or greater) in price. Once a merchant and item is found, the customer clicks 'Pay now' to start the payment process and downloads a *Payment Request* message from the merchant's website. To commence the attack, the customer's wallet authorises the payment transaction τ_C and inserts the illicit trader's payment address T in the *Payment* message as the refund address (instead of their own refund address) and then sends the message to the merchant.

The customer must request a refund to finish the attack once their Bitcoin wallet has received the *Payment Acknowledgement* message alongside a confirmation e-mail from the merchant. Assuming the merchant follows the Payment Protocol faithfully, the refunded bitcoins in τ_B are sent to the illicit trader's payment address T. Also, the customer can detect the refund transaction (merchant sending bitcoins to the illicit trader) τ_B on the Bitcoin network before contacting the illicit trader for an acknowledgement that the 'illicit goods' have been dispatched.

Ideally, if this attack happened in practice, the merchant could provide the *Payment* message as publicly verifiable evidence that the bitcoins were sent to the refund address provided by the pseudonymous customer. Unfortunately, the customer may plausibly deny having supplied the illicit trader's payment address due to their lack of endorsement, and hence claim that the merchant has forged the message single-handedly.

4.3.2 Marketplace Trader attack

In practice, the policy of Coinbase and BitPay encourages customers to provide refund addresses using an external method of communication such as e-mail [28][123] which ignores the refund address sent in the Payment Protocol. This deviation from the protocol is the basis of a new phishing style attack as a 'rogue trader' can use the reputation of a 'trusted' merchant to encourage potential customers to purchase an item from their website.

Figure 4.4 outlines this attack. It begins with the rogue trader establishing a website that sells the latest products well below the market rate to attract customers to their store. Most customers may be suspicious that the rogue trader can offer these prices and may wisely think it is a scam. To encourage customers to proceed with a purchase, the rogue trader can advertise that all payments are sent to a trusted merchant such as CeX and there is little reason not to trust them. When a customer proceeds to checkout on the rogue

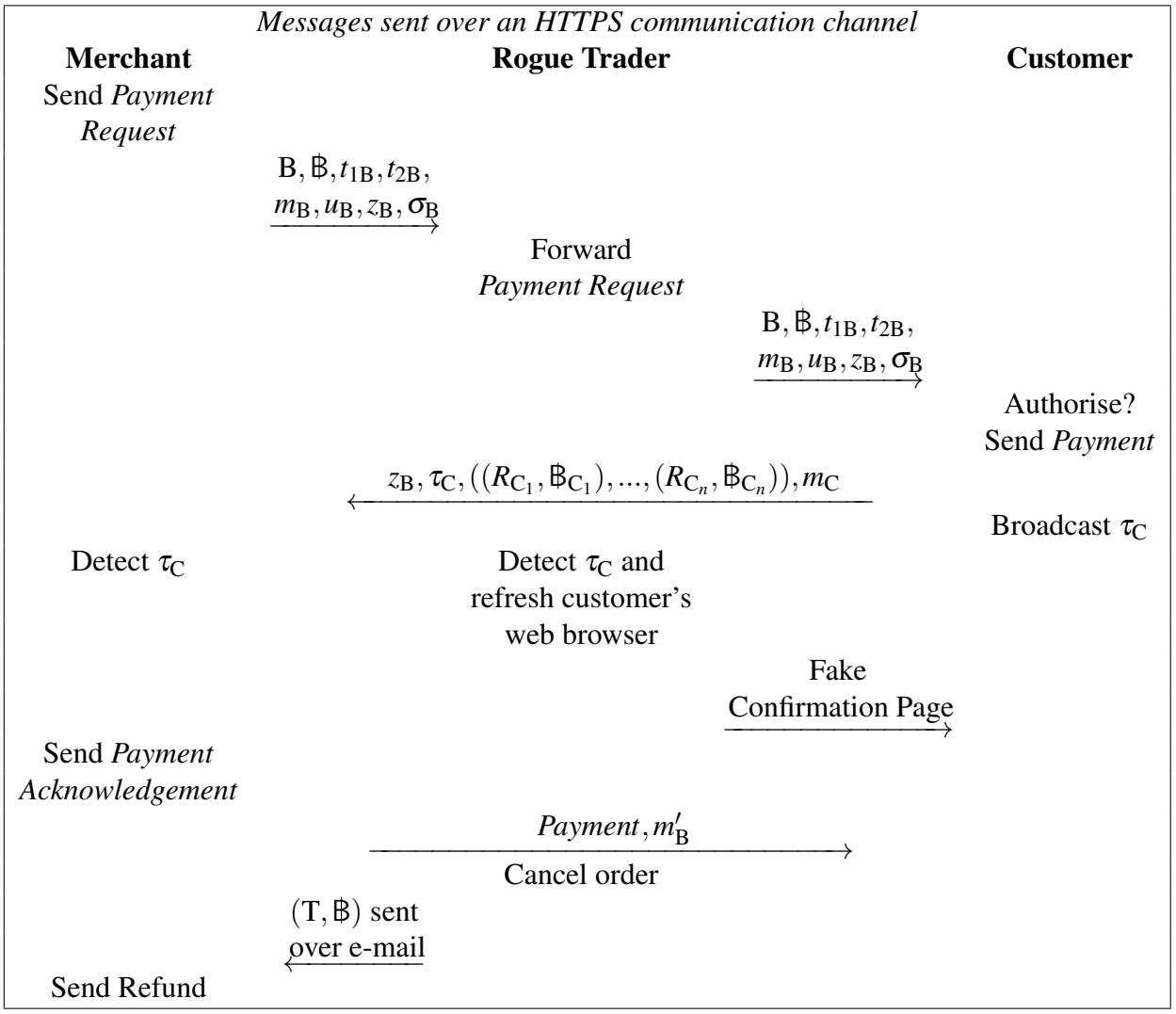


Figure 4.4 *Marketplace Trader* attack involves a rogue trader using the reputation of a 'trusted' merchant to encourage customers to fall victim to a phishing-style attack.

trader's website and clicks 'Pay now', the rogue trader's website can automatically fetch a *Payment Request* message from the trusted merchant's website and forward this to the customer. The customer's wallet opens the genuine *Payment Request* message and displays a human-readable name for the trusted merchant alongside the number of requested bitcoins. This can boost the customer's confidence that the rogue trader is legitimate as the payment is sent to the 'trusted' merchant.

Unfortunately, the customer falls victim to the attack upon authorising the payment as they are unwittingly paying for a purchase on behalf of the rogue trader to the trusted

merchant. The rogue trader detects the payment[5] transaction on the Bitcoin network and refreshes the victim's web browser to display a fake confirmation page (remember, the customer's web browser is connected to the rogue trader's website). The rogue trader can proceed to cancel the order and send a new refund address over e-mail to the trusted merchant. As the merchant's policy is to use an external method of communication to authenticate customers and deviate from the Payment Protocol standard - then the refund address sent by the rogue trader over e-mail should receive the bitcoins.

Furthermore, the customer cannot be aware this attack has occurred as they lack enough information to identify the refund transaction on the Bitcoin network. More importantly, this attack is deployable single-handedly by a rogue trader and does not require the co-operation of a 'trusted' merchant. In fact, the trusted merchant may only become aware of this scam if contacted in the future by the customer.

4.4 Real-world experiments

Our experiments aim to verify the current practice of processing refunds by merchants, and assess the feasibility of the attacks. We purchased items from real-life merchants using a modified Bitcoin wallet before requesting for the order to be cancelled and a refund processed. The attack is considered successful if the refunded bitcoins are received by the adversary's wallet. The merchants used during these experiments are based in the UK and are supported by BitPay or Coinbase. The bitcoins used for the experiments are owned by the authors and no money is sent to any illicit trader. All experiments have been ethnically approved by Newcastle University's ethical committee.

4.4.1 Proof of concept wallet

We have developed a wallet which supports the Payment Protocol and automates the *Silkroad Trader* attack. We explain how our wallet works step-by-step:

1. The customer inserts the illicit trader's *Payment Request* URI into the wallet which stores both the request and Bitcoin address for later use.

2. The customer finds an item equal (or greater) in value as the 'illicit goods' and inserts the merchant's *Payment Request* URI into their wallet.

3. The wallet provides a list of refund addresses that can be chosen for the *Payment* message that is sent to the merchant and the customer can choose the illicit trader's Bitcoin address.

[5]Currently 50% of nodes on the network receive a new transaction within 5 seconds [3].

4. Assuming a refund has been authorised by the merchant, the wallet can detect the merchant's refund transaction on the network and include it in a *Payment* message that is sent to the illicit trader.

5. The wallet is notified by a *Payment Acknowledgement* message from the illicit trader that the payment has been received.

4.4.2 Simulation of attacks

We discuss our experience carrying out a simulation of both attacks against real world merchants using arbitrary identities (i.e., random name, e-mail address, telephone number, delivery/billing addresses created for experiments only). Only e-mail is used to communicate with each merchant. Our results for the *Silkroad Trader* attack are as follows:

Cex refunded the bitcoins within 3 hours of cancelling the order and used the refund address from the Payment Protocol.

Pimoroni Ltd refunded the bitcoins within a single business day and used the refund address from the Payment Protocol.

Scan refunded the bitcoins after 26 days and used the refund address from the Payment Protocol. The delay was due to Scan initially requesting us to provide a refund address over e-mail, but we insisted using the one specified in the original payment message.

Dell were unable to process the refund due to 'technical difficulties' and requested our bank details. We informed them that we did not own a bank account and Dell suggested sending the refund as a cheque. While not the experiment's aim, this potentially opens Dell as an exchange for laundering tainted bitcoins.

To simulate the *Marketplace Trader* attacks we sent the refund address in an e-mail to the merchants. Assuming the merchants accepted e-mail as a good form of authentication and ignored the refund address sent in the Payment Protocol, then the phishing-style attack we described earlier could happen in practice. Our results were the following:

Something Geeky refunded the bitcoins within a single business day to a refund address sent over e-mail.

Girl meets dress refunded the bitcoins within 11 business days to a refund address sent over e-mail. The delay was due to the merchant initially thinking we paid using a bank transfer.

BitRoad refunded the bitcoins within a single business day to a refund address sent over e-mail. In this experiment, we registered using a non-existing e-mail address and requested for the order to be cancelled using a variant of the e-mail address. This demonstrates that even the registered e-mail address to initiate the purchase is not being used to authenticate the customer.

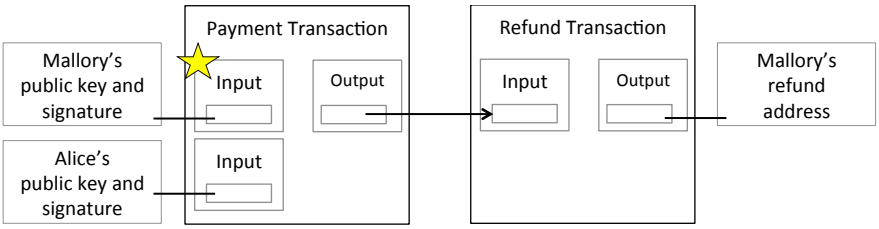


Figure 4.5 The malicious co-signer attack allows a co-signer the sole authority to endorse the refund address used by the merchant and thus steal the bitcoins of other co-signers

4.5 Solution

We propose providing the merchant with publicly verifiable evidence that can cryptographically prove the refund address received during the protocol was endorsed by the same pseudonymous customer who authorised the payment.

A solution proposed by Hearn [53] assumes the payment transaction is authorised by a single customer and recommends endorsing the refund address using any key which authorised the transaction. However, it is not valid to assume that a transaction has been authorised by a single customer due to the nature of a Bitcoin transaction. If the adversary is responsible for sending the *Payment* message to the merchant, then they have the sole authority to endorse the refund address used by the merchant as seen in Figure 4.5.

Our proposed solution prevents this attack by requiring each key that authorised the transaction to also endorse its own refund address. In the event of a refund the merchant sends the same (or less) number of bitcoins received[6] from each transaction input to an associated refund address.

4.5.1 Proposed Solution

To achieve a signature solution requires changes to each message sent as part of the protocol. We outline these changes in Figure 4.6 and explain each message separately before discussing their implications.

The **Payment Request** message considers the memo m_B as a mandatory parameter and should contain enough information for the customer(s) to be aware that this payment request is only for them, e.g. the registered e-mail address, delivery address, product information, etc. This memo field should also include customer-specified instructions to provide evidence that the merchant followed any instructions provided by the customer. The payment address B should be unique for each *Payment Request* and like before, there should be no restriction on

[6] A transaction input does not record the number of bitcoins 'sent' and instead references an output from a previous transaction which specifies the bitcoins.

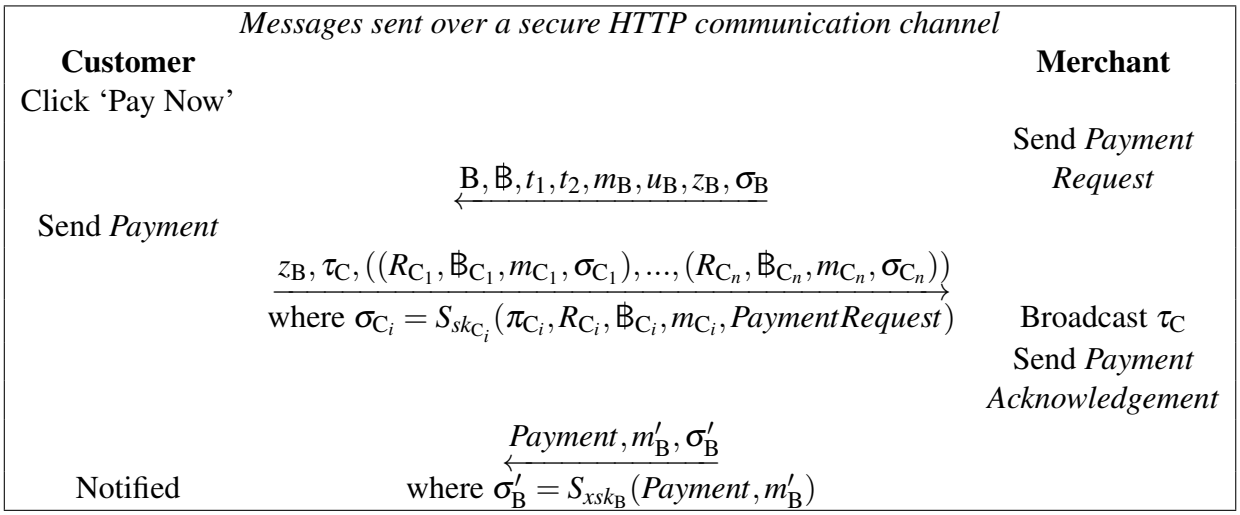


Figure 4.6 A single customer sends a payment to the merchant

the number of times a customer can download the same *Payment Request* to support paying from multiple devices or sharing with others.

The **Payment** message aims to associate each transaction input π_{C_i} with a refund address R_{C_i} by endorsing the refund address using the same keys that authorised the transaction input. We assume the customer is no longer responsible for broadcasting the payment transaction τ_C to the Bitcoin network; instead, the responsibility of broadcasting the payment transaction should fall on the merchant (as recommended by one of the original authors of the Payment Protocol[7]). For simplicity, we describe our solution using a single *Payment* message and payment transaction τ_C[8].

Each refund address endorsement signature is $\sigma_{\mathrm{C}_i} = S_{sk_{\mathrm{C}_i}}(\pi_{\mathrm{C}_i}, R_{\mathrm{C}_i}, \text{฿}_{\mathrm{C}_i}, m_{\mathrm{C}_i}\ Payment\ Request)$, where S is the signature algorithm, sk_{C_i} is the private key which corresponds to the key that authorised the transaction input, π_{C_i} is a concatenation of the elements that constitute the signed transaction input, R_{C_i} is the refund address, ฿_{C_i} is the number of bitcoins to refund, m_{C_i} is an additional memo from the customer and *Payment Request* is the signed message from the merchant. These parameters were chosen to clarify which transaction input is associated with the endorsed refund address and to ensure this endorsement is only valid for this *Payment Request* message. The concatenated information π_{C_i} is the data stored inside the transaction input and includes: the previous transaction identification hash, an index for the output in the referenced transaction and the script which contains a signature to authorise the payment and its corresponding public key.

[7] https://groups.google.com/forum/#!msg/bitcoinj/ymFRupTSRJQ/zANj2RpslCcJ

[8] Our solution continues to allow customers to send one or more *Payment* messages to the merchant until all requested bitcoins have been received. Furthermore, these messages can contain a list payment transactions.

A new refund policy for the merchant is required as each transaction input is responsible for endorsing a refund address. We propose the bitcoins associated with each refund address must be equal to (or less than) the number of bitcoins sent in the respective transaction input. The merchant must also check that the total number of bitcoins associated with all refund addresses in this message is equal to (or less than) the number of bitcoins he received in the payment transaction. These checks are necessary as the payment transaction can have additional outputs for change and the merchant needs to ensure he does not refund more bitcoins than he received from each transaction input. For example, if the transaction has a single input of ฿5 (from the customer) and two outputs: ฿4 (to the merchant) and ฿1 (to the customer as change), then the merchant must ensure the customer is only refunded ฿4 (and not ฿5).

The message content sent to the merchant is outlined in Figure 4.6 and includes: the merchant-specific data z_{B}, the complete transaction τ_{C} and a list of refund addresses alongside their associated endorsement signatures and the number of bitcoins to refund $((R_{\mathrm{C}_1}, ฿_{\mathrm{C}_1}, m_{\mathrm{C}_1}, \sigma_{\mathrm{C}_1}), ..., (R_{\mathrm{C}_n}, ฿_{\mathrm{C}_n}, m_{\mathrm{C}_n}, \sigma_{\mathrm{C}_n}))$.

The **Payment Acknowledgement** message is signed using the merchant's X.509 private key and repeats the customer's *Payment* message alongside an additional memo m'_{B}. The signature is $\sigma'_{\mathrm{B}} = S_{xsk_{\mathrm{B}}}(Payment, m'_{\mathrm{B}})$ where S is the signature algorithm and xsk_{B} is the private key that corresponds to the merchant's public key in their X.509 certificate.

We simplified the notation for σ_{C_i} to only show the case when the customer endorses the ith refund address using a *single* signing key. However, if the multi-signature approach is used to authorise the transaction input, then a threshold of k signing keys should be used to endorse the respective refund address. Each customer with control of 1 of k keys can independently authorise the transaction input and the corresponding refund address. Both signatures which endorse the refund address and authorise transaction input are sent to the other co-signers to be included in the *Payment* message.

4.5.2 Discussion

Our solution provides a **proof of endorsement** as the refund address received by the merchant is signed using the same set of keys used to authorise the transaction. This evidence removes the customer's plausible deniability for their involvement in the *Silkroad Trader* attack. Crucially, this is a detective solution which allows a third party to determine whether the customer was involved in supplying payment addresses for an illicit trader as opposed to preventing it entirely. Furthermore it provides an incentive for the merchant to use the refund address sent during the Payment Protocol which prevents the *Marketplace Trader* attack. The merchant does not need to distinguish whether or not the payment has been split which

preserves the customers privacy and prevents the malicious co-signer attack as co-signers cannot endorse the refund addresses of others. These additional signatures are handled by the wallet on behalf of the user. Also, **no connection to the peer to peer network** is required for the customer as the merchant is responsible for broadcasting the payment transaction τ_C and this prepares the Payment Protocol to **support off-chain transactions** such as the Lightning Network [100].

Furthermore, we explored other potential solutions such as requesting the customer to provide a signature from the refund address at the time of payment (instead of using the same keys that authorised the transaction) or including secret data inside the merchant-specific data field z_B. The former is not satisfactory as proving ownership of the refund address does not necessarily link the refund address to the same keys that authorised the transaction. Although if it is used in combination with our solution then it may act as a preventive measure as it would require the illicit trader's co-operation which creates an additional barrier to perform the attack. The latter remains vulnerable to the *Marketplace Trader* attack as the rogue trader has access to the *Payment Request* message. As well, the attacks introduced in this chapter also stem from the fact that merchants have no community-accepted refund protocol today. While researchers have proposed secure post-payment communication protocols [77] in the past which could conceivably be used to support arranging refund in a private and authenticated manner, this remains a subject for further investigation in future work.

Payment Processors are expected to perform anti-money-laundering policies on behalf of their merchants [43]. The state of New York recently released Bitlicense [115] to outline regulation for Bitcoin businesses. Our solution enhances the book-keeping required for this license as the signed *Payment* message is cryptographic evidence that the pseudonymous customer has endorsed the transaction-related information required for auditing by investigators. We improve the mandatory customer receipt which is currently a static web-page or an e-mail by using the *Payment Acknowledgement* message as a cryptographic receipt as it is signed by the merchant's X.509 private key. Similar cryptographic evidence has been explored in the Bitcoin research community and two examples include: providing a warrant to hold a mixer accountable in the event of any wrongdoing with Mixcoin/Blindcoin [18][124], and to compute a proof of solvency [44] that demonstrates the business is financially in good standing to customers.

Clustering techniques have been demonstrated to link a group of Bitcoin addresses to a single pseudonymous user [9]. Meiklejohn et al. [82] identified that 374.49 BTC stolen from Betcoin in April 2012 and 4,588 BTC from the Bitcoinica theft in May 2012 were sold at Bitcoin-24, Mt Gox, BTC-e, CampBX and Bitstamp. Also, Reid et al. [101] tracked 25k stolen bitcoins and deduced LulSec's involvement in the theft. These analysis tech-

niques using the Blockchain are currently supporting criminal charges in the Silkroad Trial [4]. However, privacy-enhancing protocols [75][105][81] and altcoins [84][86] are actively reducing the effectiveness of these analysis techniques. Nevertheless, these techniques provide a platform for the *Silkroad Trader* attack as independent observers may discover merchants sending bitcoins to an illicit trader and then publicly release the 'evidence'. Our solution provides the merchant with publicly verifiable evidence to demonstrate a customer's deception.

4.5.3 Inherent issues due to Bitcoin

We outline four issues that are inherent due to Bitcoin that need to be considered for our solution:

First, the *proof of endorsement* evidence can only authenticate pseudonymous customers as the Payment Protocol lacks the type of *real-life identity endorsement* that comes with banks. While protecting an honest merchant, our solution cannot prevent a malicious merchant simulating both attacks and insisting they were tricked.

Second, in a similar way to the original protocol, an observer of the Blockchain may be able to link the payment and refund transactions using the denominations of bitcoins sent and received.

Third, customers can re-sign the transaction to change the identification hash and broadcast it to the network. We recommend the merchant keeps a copy of the payment transaction received in the *Payment* message as a re-signed transaction cannot be used to verify the endorsement in the future.

Fourth, we assume merchants maintain the UTXO (Unspent Transaction Output) set to participate in the Payment Protocol. Without this list of spendable outputs, the merchant cannot independently verify transactions or calculate the number of bitcoins to refund for each transaction input. On the other hand, customers do not require the UTXO set and can continue to use SPV (Simplified Payment Verification) wallets for the Payment Protocol.

4.5.4 Solution performance

All tests are carried out on a MacBook Pro mid-2012 running OS X 10.9.1 with 2.3GHz Intel Core i7 and 16 GB DDR3 RAM. Time performance in Table 4.1 represents both the current Payment Protocol implementation and our proposed modifications for the Bitcoin Core Client while utilising 1 core. Furthermore, both signing operations in steps 3 and 8, and the verification operation in step 9, are performed using the Secp256k1 implementation which has recently replaced OpenSSL in Bitcoin Core [131]. Each step was executed 100 times and the reported times represent the average.

Step	Description	Time
	Customer in the current protocol	
1	Verify merchant's certificate and chain of certificates authenticity	0.83 ms
2	Verify merchant's signature on the Payment Request message	0.08 ms
3	Sign a single transaction input	0.08 ms
4a	Fetch a list of previously generated refund addresses $R_{C_1}, ..., R_{C_k}$	0.72 ms
4b	Generate a new refund address R_C from the wallet's key pool	110.55 ms
5	Update wallet's address book with the refund address R_C	72.68 ms
	Total without 4b:	*74.39 ms*
	Total with 4b:	*184.94 ms*
	Merchant in the current protocol	
6	Verify the customer's payment transaction	0.29 ms
	Total:	*0.29 ms*
	Additional changes proposed for the customer	
7	Produce endorsement signature σ_C using the private key sk_C	0.11 ms
	New Total without 4b:	*74.49 ms*
	New Total with 4b:	*185.04 ms*
	Additional changes proposed for the merchant	
8	Fetch the transaction input's referenced transaction output	0.01 ms
9	Verify the transaction input's endorsement signature σ_C	0.13 ms
	New Total:	*0.43 ms*

Table 4.1 Time performance for proposed changes to the Payment Protocol

Steps 1–5 represent the customer's perspective in the current Payment Protocol's implementation. The wallet verifies the merchant's certificate authenticity using the chain of certificates that lead to a trusted root authority and verifies the merchant's signature on the *Payment Request* message before authorising at least one transaction input to authorise the payment. Then, the wallet fetches a list of pre-generated refund addresses and Step 4b only occurs if this list is empty as a new refund address must be generated. This refund address is associated with the payment for future reference. These steps require 74.39 ms if the list of pre-generated refund addresses is not empty, otherwise 184.94 ms is required. Our proposed change in Step 7 takes 0.11 ms and requires the customer's wallet to sign an endorsement message for the refund address, obtaining the signature σ_C. In total, the time required for the customer is 185.04 ms with Step 4b, and 74.49 ms without Step 4b.

Step 6 represents the merchant's perspective in the current Payment Protocol's implementation and requires 0.29 ms to check if the payment transaction with a single input is valid. We propose in Steps 8–9 that the merchant fetches the transaction output referenced in the payment transaction's input to let the merchant check the number of bitcoins associated with each refund address. Then, the transaction input's public key C is used to verify the

endorsement signature. These proposed changes require 0.14 ms, and in total the time required for the merchant is 0.43 ms.

4.6 Payment Processors Response

We privately disclosed our attacks to the Payment Processors and received the following response:

BitPay acknowledged “the researchers have done their homework" and that “refunds are definitely a significant money laundering attack vector". They are now actively monitoring for refund activity on behalf of their merchants. Furthermore, after we disclosed our results, BitPay released a new refund flow [16] that recommends using the refund address provided in the Payment Protocol rather than the one supplied by email.

Coinbase acknowledged the ‘Silkroad Trader’ attack as a good example of an authentication vulnerability in the Payment Protocol. To prevent the *Marketplace Trader* attack, Coinbase no longer provides merchants the API to change the refund address if it has been supplied by the Payment Protocol. Also, they have updated their user documentation to discourage merchants sending the refund using their own bitcoins to bypass the API changes.

Bitt is preparing to launch merchant services for the Caribbean and acknowledged both attacks. They believe the endorsement evidence may support Payment Processors become more ‘airtight’ for future regulation.

These temporary mitigation measures help to address the *Marketplace Trader* attack, but not the *Silkroad Trader* attack. To fully address the latter, the BIP70 standard would need to be revised, as we have discussed in Section 5.

4.7 Conclusion

This chapter presented two attacks that leverage an authentication vulnerability in Bitcoin’s Payment Protocol and the refund policies of the two largest Payment Processors: Coinbase and BitPay. We experimentally demonstrated both attacks on real-life merchants using a proof of concept wallet before proposing a solution that provides the merchant with cryptographic evidence that the refund address received during the Payment Protocol has been endorsed from the same pseudonymous customer who authorised the transaction. In the next chapter, we discuss the scalability issues facing Bitcoin and provide a survey on payment channels. These protocols permit two users to privately transact and only the final aggregation of payments is stored in the blockchain.

Chapter 5

Towards Bitcoin Payment Networks

5.1 Introduction

The Bitcoin community fears that Bitcoin [90], the 'Money for the Internet' and currently the most popular cryptocurrency, cannot scale to meet future demand. Today, the network can process 3.3–7 tps (transactions per second) as each block is artificially capped at 1MB and network measurements in 2015 [31] highlight that 90% of the network's peer to peer network can only support an effective throughput of up to 27 tps. This is dwarfed by established payment providers such as Visa, which facilitates about 2,000 tps, with a peak capacity of 56,000 tps [122].

Bitcoin's capacity limitations are increasingly felt by users in the form of delayed transaction processing and rising transaction fees. Users currently pay about 3 to 7 US cents per transaction (independent of the amount transferred). The costs of sending a transaction could continue to rise as competition for space in the Blockchain increases and the protocol's monetary policy continually reduces the minting of new coins that rewards 'miners' for securing the network.

A simple short-term fix to increase Bitcoin's capacity would be to increase the maximum block size, allowing more transactions to be included in each block. While multiple such proposals exist [7][45][97], none have actively been adopted as the community cannot agree if the size of blocks should be increased at all, incrementally, dynamically or whether an artificial cap is required at all.

Long-term research has focused on two directions to improve scalability:

1. Redesigning the underlying Blockchain protocol to support more transactions per second [12, 39, 69, 112], and

2. Facilitating 'off-chain transactions' where transactions are only committed to the Blockchain if an adjudicator is required [32, 100].

In this chapter we explore the latter direction and provide an overview of payment networks which are composed of two components; a payment channel and a mechanism to route payments across a series of channels. This scaling approach bootstraps trust from the blockchain in order to support local settlement of payments amongst the involved parties, instead of publishing all transactions for global endorsement from the Bitcoin network. Crucially, there are no trust-assumptions for the local settlement, payments are confirmed immediately due to the nature of local settlement and all parties are guaranteed to receive their fair share of a channel's balance if they are on-line when the channel is closed.

With that said, we hope to provide a practical perspective in regards to the adoption of payment networks by illustrating several caveats at the heart of this scaling approach. In Section 5.2 we show the evolution of constructing payment channels to support a substantial number of payments and remove the expiry time associated with a channel has so far been hindered by Bitcoin's transaction design. This is evident in Section 5.3 which presents Duplex Micropayment Channels [33] and Lightning Channels [100] as both protocols require a complex transaction structure and an elaborate ordering for signing transactions. Once there is an existing network of channels, Section 5.4 highlights that this scaling approach imposes an additional difficulty on sending a payment as the involved parties are responsible for finding a route of on-line channels that connects them (and has sufficient capacity). Finally depending on the channel construction, each party must be on-line at a specified time or periodically monitor the blockchain, to avoid the counter-party effectively reversing payments by closing the channel on a previously authorised balance for both parties.

5.2 Background

We first introduce the necessary background about Bitcoin's lock time rules as used by payment networks. Then, we present how to establish and send bitcoins in basic payment channels with an untrusted counterparty.

5.2.1 Time Locks in Bitcoin

It is possible to lock coins for a specified duration of time using *lock times* in Bitcoin. A lock time can be given as a future block height or time-stamp that is stored in a block's header. Most importantly, these lock times are specific to transaction outputs and not the entire transaction. There are two types of lock time rules that are required for payment channels:

Absolute Lock Time ensures an entire transaction[1] or a child transaction that is spending an output of a parent transaction[2] cannot be accepted into the Blockchain until a specified absolute block height k (or time) in the future,

Relative Lock Time ensures a child transaction that is spending an output of a parent transaction cannot be accepted into the Blockchain until the parent transaction has achieved a relative depth of λ blocks[3].

5.2.2 Payment Channel Establishment

A payment channel allows two parties to send numerous payments to each other. Instead of settling all transactions directly on the Blockchain, a payment channel only requires two transactions: one to open the channel, and one to close it and settle the final balance. The cornerstone of payment channels is depositing bitcoins into a multi-signature address controlled by both parties and having the guarantee that all bitcoins are eventually refunded at a mutually agreed time if the channel expires.

Spilman [114] proposed the first payment channel establishment protocol without the need to trust the counterparty. This protocol has a *Funding Transaction* that stores the depositor's bitcoins and requires the authorisation of both parties to spend, and a *Refund Transaction* that returns the funds to the depositor if no payments have been authorized or the counterparty abandons the protocol. The lock time on the refund determines the lifetime of the payment channel. To establish the channel, the *Funding Transaction* is created by the depositor and remains unpublished until she received valid signatures for the *Refund Transaction* from the counterparty.

Another possibility to realise refunds is through *Non-Interactive Time-Locked Refunds*, that recently became available in Bitcoin with the activation of BIP 65 [119]. *Non-Interactive Time-Locked Refunds* account for the channel timeout directly in the multi-signature output of the *Funding Transaction*, which means that dedicated refund transactions are no longer necessary. Once the *Absolute Lock Time* specified in the output's script has expired, the refund condition can be redeemed without the cooperation of the counterparty. While the output can only refund a single party, refunding multiple parties is possible with dedicated multi-signature outputs representing each participant's deposit.

In practice, the use of payment channel protocols is currently hindered by the problem of transaction malleability and the infeasibility to build upon unsigned transactions. Transaction

[1] The `nLockTime` field of the transaction

[2] The output's script contains the `OP_CHECKLOCKTIMEVERIFY` opcode.

[3] The output's script contains the `OP_CHECKSEQUENCEVERIFY` opcode.

malleability allows a co-signer or an external third party to change the identification hash of a transaction before it is accepted into the Blockchain. This is a problem for contracts that sign child transactions before the parent transaction, whose outputs are being spent, is included in the Blockchain. If a modified parent transaction is accepted into the Blockchain, then all pre-signed child transactions become invalid. Furthermore, it is impossible to build upon unsigned transactions as adding signatures to a transaction changes its identification hash. BIP 66 [130] has been deployed to prevent third-party signature malleability, but co-signer and other types of malleability remain. BIP 141 [71], however, solves these malleability issues and allows to build upon unsigned transactions, and is likely to be deployed soon.

5.2.3 Basic Payment Channels

We introduce unidirectional and bidirectional channels that leverage the *Funding Transaction*'s multi-signature output. The establishment protocol in Section 5.2.2 is used to set up both channels. Bitcoins are sent using subsequent *Payment Transactions* that have two outputs to send each party their respective bitcoins.

Unidirectional channels were first implemented by Corallo in Bitcoinj [52] to allow a customer to send incremental payments to a merchant. Each payment has two outputs: the first increases the amount of bitcoins sent to the merchant, and the second returns change to the customer. This introduces the *Replace by Incentive* rule as the merchant only signs and broadcasts the latest *Payment Transaction* that sends them the most bitcoins. Payments can be made until the channel expires or the whole deposit has been transferred to the merchant.

Bidirectional channels require the *Payment Transaction* to be associated with an *Absolute Lock Time*. Each incremental payment decrements the lock time by a safety margin Δ that represents the expected time for transactions to be accepted into the Blockchain. This introduces the *Replace by Timelock* rule as the latest *Payment Transaction* is guaranteed to be accepted into the Blockchain before any previously authorised transaction. Changing the direction of payments (i.e. Alice sends coins to Bob, and then Bob sends coins to Alice) requires both parties to sign a new transaction that decrement the channel's lifetime.

5.3 Proposed Payment Channel Protocols

In this section, we outline two proposals for payment channels. The first one is called Duplex Micropayment Channels, due to Decker and Wattenhofer [32]. It extends the number of transactions that can occur within the lifetime of a bidirectional channel. The second one is

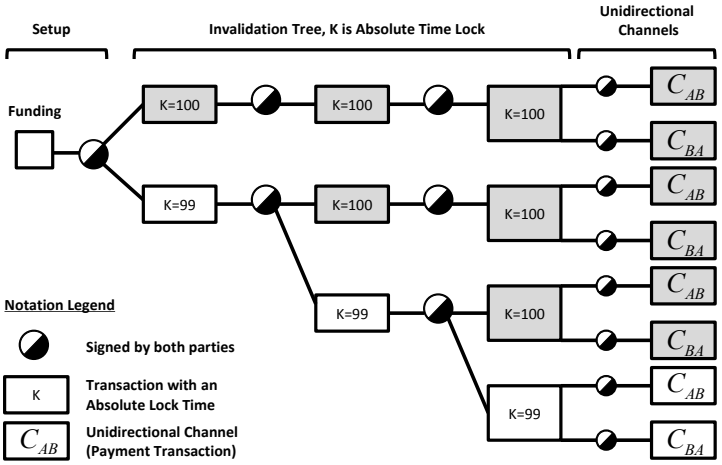


Figure 5.1 **Duplex Micropayment Channels**. A *Funding Transaction* is stored in the Blockchain. All payments occur in the unidirectional channels. If either channel becomes exhausted, then a new branch is created in the *Invalidation Tree* with a smaller *Absolute Time Lock k*. This invalidates all previous branches, and resets the balance of both unidirectional channels. For illustration $\Delta = 1$.

called Lightning Channels, proposed by Poon and Dryja [100], and allows the channel to remain open indefinitely.

5.3.1 Duplex Micropayment Channels

Decker and Wattenhofer propose Duplex Micropayment Channels which enable bidirectional payment channels with a finite lifetime [32]. Their scheme builds upon an initial *Funding Transaction* with deposits from two parties A and B. It consists of two *Unidirectional channels* C_{AB}, C_{BA} that together allow bidirectional payments, and an *Invalidation Tree* that sets the minimum lifetime of the channel and is responsible for resetting the bitcoins available to spend in both channels C_{AB}, C_{BA}. This reset is necessary as Alice may receive 1 BTC from Bob in the unidirectional channel C_{BA}, but her unidirectional channel to Bob C_{AB} has exhausted its supply of bitcoins. To continue bidirectional payments it is necessary to refresh C_{AB} with Alice's 1 BTC.

The core idea of Duplex Micropayment Channels is to apply the *Replace by Timelock* rule (cf. Section 5.2.3) using a tree of timelocked transactions instead of a single transaction. This structure is called an *Invalidation Tree*, exemplarily shown in Figure 5.1. The nodes in the tree represent Bitcoin transactions; each edge corresponds to the spending of the previous node's output and is signed by both parties. Each transaction $T_{d,k}$ has a depth d in the tree and is associated with an *Absolute Lock Time k*. Nodes in the active branch $(T_{1,k}, T_{2,k}, ..., T_{d,k})$

have *Absolute Lock Times* that are less than previously authorised branches. This guarantees that the active branch is accepted into the Blockchain before previously authorised branches.

The leaf node $T_{d,k}$ on the active branch has two outputs to represent each unidirectional channel C_{AB} and C_{BA}. Each output can be spent if either of the following conditions is satisfied:

1. The first condition requires the signature of both parties to authorise a *Payment Transaction*.

2. The second condition requires the signature of the depositor and that the current Blockchain height is greater than an *Absolute Lock Time* $k_{\max}$. This lock time $k_{\max}$ is the maximum waiting time before the bitcoins can be returned to the depositor using a *Refund Transaction*.

Payments are sent in the unidirectional channels C_{AB}, C_{BA} as the sender signs subsequent *Payment Transactions* that have two outputs: the first sends bitcoins to the receiver, while the second returns change to the sender. The *Replace by Incentive* rule implies that the receiver will only sign the *Payment Transaction* that sends them the most bitcoins. If the receiver is unresponsive, the depositor is guaranteed to have their bitcoins refunded once the maximum lifetime of the channel $k_{\max}$ expires. It is possible to reset the balance of both unidirectional channels once a channel has exhausted its supply of bitcoins. Resetting the channels requires replacing the current active branch in the *Invalidation Tree* with a new branch whose leaf node $T_{d,k}$ acts as an anchor for a new set of unidirectional channels.

In the following, we explain how to establish the channel, send bidirectional payments, reset the balance of both *Unidirectional channels* and finally settle the payment channel on the Blockchain.

Channel Establishment First, both parties combine their funds in an unsigned *Funding Transaction*. Then, they build the first branch $(T_{1,k}, T_{2,k}, \ldots, T_{d,k})$ of the *Invalidation Tree* and exchange signatures for the branch and the *Payment Transactions* that represent the unidirectional channels C_{AB}, C_{BA}. Finally, both parties sign and broadcast the *Funding Transaction*.

The *Absolute Lock Time* $k_{\max}$ of the *Refund Transaction* should exceed the *Absolute Lock Time* of the leaf node $T_{d,k}$ by at least a safety margin Δ. This ensures that the unidirectional channels have enough time to be accepted into the Blockchain before the *Refund Transactions* become available to spend.

Send a Payment Each payment requires the sender to sign a new *Payment Transaction*. The receiver only signs the transaction that sends them the most bitcoins if they want to settle a dispute on the Blockchain. If either unidirectional channel has exhausted its supply of

bitcoins, then both parties must cooperate to reset the balance of both channels before further payments can be made.

Reset Balance Resetting the balance of the unidirectional channels requires both parties to agree a new active branch in the *Invalidation Tree*. Figure 5.1 demonstrates several branch replacements. An example includes the current active branch $(T_{1,99}, T_{2,99}, T_{3,99})$ replacing the previous branch $(T_{1,99}, T_{2,99}, T_{3,100})$.

First, both parties find the first node $T_{\alpha,k}$ closest to the leaves that has a greater *Absolute Lock Time* than it's parent node by a safety margin of Δ, otherwise the root node $T_{1,k}$ is chosen.

Second, a new branch is created, starting with the chosen node whose *Absolute Lock Time* is decremented by the safety margin Δ (i.e. $T_{\alpha,k-\Delta}$). Each child node in this new branch is given an *Absolute Lock Time* greater than $k-\Delta$ (e. g., the maximum value initially chosen when the tree was established). As lock times are *transitive*, the reduced lock time of the parent transaction automatically invalidates all previously authorised branches. Note, that when the lock time of the root node $T_{1,k-\Delta}$ is decremented, then the channel's minimum lifetime $k_{\min}$ is also reduced.

Third, both parties exchange signatures for the new active branch. The signatures for the node $T_{\alpha,k-\Delta}$ are only exchanged once both parties have successfully exchanged signatures for the *Payment Transactions* that represent the unidirectional channels C_{AB}, C_{BA} and all of its child nodes $(T_{\alpha+1,k}, ..., T_{d,k})$.

Settle Channel Both parties cooperate to sign and broadcast a transaction that has no *Absolute Lock Time* and has two outputs to send each party their respective final balance. If both parties cannot cooperate, then either party can broadcast the current active branch for inclusion into the Blockchain once the *Absolute Lock Times* expire. If the receiver in a unidirectional channel does not broadcast the *Payment Transaction* that sends them the most bitcoins, then the depositor waits until $k_{\max}$ to broadcast the *Refund Transaction*.

5.3.2 Lightning Channels

Poon and Dryja propose bidirectional payment channels called Lightning Channels that can remain open indefinitely [100]. Sending a payment requires the cooperation of both parties to authorise a new channel state that represents each party's new balance, before revoking the channel's current state. The revocation mechanism requires both parties to check the Blockchain periodically to detect if a previously revoked channel state has been submitted. If a revoked state is detected, the counterparty that did not broadcast the revoked state can issue a penalty to claim all bitcoins in the channel. Initially, this revocation was performed by exchanging signatures for the penalty transaction. An improvement proposed by Back is

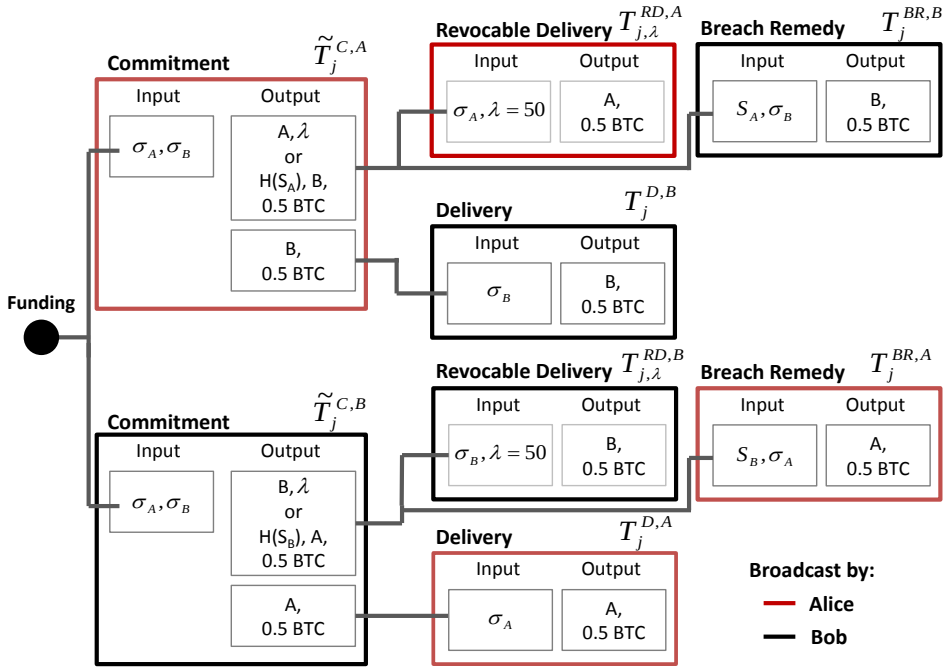


Figure 5.2 **Lightning Channels**. Each party has a *Commitment Transaction* that only they can broadcast. To settle a dispute on the Blockchain, either party can broadcast their *Commitment Transaction* and *Revocable Delivery Transaction* to claim their share of bitcoins. In response, the counterparty broadcasts their *Delivery Transaction* to receive their share of bitcoins. If a *Commitment Transactions* has previously been revoked, the counterparty can broadcast the *Breach Remedy Transaction* to steal all bitcoins in the channel.

outlined in [107] to use the pre-image S of a revocation hash $H(S)$. This revocation hash is used in our description of Lightning Channels.

To describe the protocol, we use the following notation: transactions are denoted as $T_{j,\lambda}^{\beta,X}$, where β is an acronym for the transaction's name, X is the party that can broadcast the transaction (i.e. it requires party X's signature to complete the transaction), j is the number of updates that have occurred in the channel and λ is an optional *Relative Lock Time*. Furthermore, T only requires the broadcaster to sign, $\tilde{T}$ requires both parties to sign and $\sigma_{X,j}^{\beta}$ represents party X's signature for the j^{th} transaction β.

Figure 5.2 presents the underlying transaction structure of Lightning Channels. The channel relies upon an initial *Funding Transaction* $\tilde{T}^F$ that is signed by both parties and deposits bitcoins into a multi-signature address. The symmetric structure of the scheme provides each party with the following transactions:

The *Commitment Transaction* $\tilde{T}_j^{C,X}$ spends the multi-signature output and therefore requires signatures from both parties. Each party receives the counterparty's signature in advance. The first output sends bitcoins to the broadcaster, while the second outputs sends

bitcoins to the counterparty. Spending the second output only requires the counterparty's signature, but spending the first output is only possible if either of the following two conditions is fulfilled:

1. The first condition requires a signature from the broadcaster and has a *Relative Lock Time* such that the *Commitment Transaction* $\tilde{T}_j^{C,X}$ needs to achieve a depth of λ before the output is spendable.

2. The second condition requires the pre-image $S_{X,j}$ of a revocation hash $h_{X,j} = H(S_{X,j})$ and a signature from the counterparty.

These conditions allow *Commitment Transactions* to be revoked by revealing the pre-image $S_{X,j}$ to the counterparty. The normal payout to the broadcaster is delayed by a *Relative Lock Time* λ to allow the counterparty to claim the funds if the transaction has previously been revoked. Revoking a *Commitment Transaction* allows both parties to update the channels while ensuring that no revoked transaction will ever be published.

The *Revocable Delivery Transaction* $T_{j,\lambda}^{RD,X}$ requires the signature of the broadcaster, and sends bitcoins to the broadcaster. It can only be broadcast once the broadcaster's *Commitment Transaction* has achieved a depth of λ in Bitcoin's blockchain.

The *Delivery Transaction* $T_{j,\lambda}^{D,X}$ requires the signature of the counterparty that did not broadcast the *Commitment Transaction* and immediately sends the counterparty their share of bitcoins.

The *Breach Remedy Transaction* $T_j^{BR,X}$ requires the signature of the counterparty that has not broadcast the revoked *Commitment Transaction* and the pre-image $S_{X,j}$ of the *Commitment Transaction*'s revocation hash. It can only be broadcast once a revoked *Commitment Transaction* has been accepted into the Blockchain. No *Relative Lock Time* is associated with this transaction to allow the counterparty to issue the penalty immediately.

In the following and in Figure 5.3, we explain how to establish the channel, send bidirectional payments, and finally settle the payment channel on the Blockchain.

Channel Establishment Alice sends Bob her revocation hash $h_{A,j}$ and a transaction input π_A for the *Funding Transaction* $\tilde{T}^F$. Bob responds with an unsigned *Funding Transaction* $\tilde{T}^F$ that includes inputs of both parties, a signature $\sigma_{B,j}^C$ for Alice's *Commitment Transaction* $T_j^{C,A}$ and his revocation hash $h_{B,j}$. Then, Alice sends a signature $\sigma_{A,j}^C$ for Bob's *Commitment Transaction* $T_j^{C,B}$ and a signature σ_A^F for the *Funding Transaction* T^F. Finally, Bob signs and broadcasts the *Funding Transaction* $\tilde{T}^F$ to the network.

Send a Payment Sending a payment requires the cooperation of both parties to authorise a new set of transactions to represent the channel's new balance before invalidating the current set of transactions.

Channel Establishment:
$A \rightarrow B$: Revocation hash $h_{A,j}$ and her transaction input π
$B \rightarrow A$: Unsigned $\tilde{T}^F$, signature $\sigma^C_{B,j}$ for $T^{C,A}_j$, revocation hash $h_{B,j}$
$A \rightarrow B$: Signature $\sigma^C_{A,j}$ for $T^{C,B}_j$, signature σ^F_A for T^F
$B \rightarrow N$: Broadcast $\tilde{T}^F$ to the Bitcoin network.
Send a Payment:
$A \rightarrow B$: New revocation hash $h_{A,j+1}$
$B \rightarrow A$: Signature $\sigma^C_{B,j+1}$ for new $T^{C,A}_{j+1}$, new revocation hash $h_{B,j+1}$
$A \rightarrow B$: Signature $\sigma^C_{A,j+1}$ for new $T^{C,B}_{j+1}$, pre-image $S_{A,j}$ of previous revocation hash $h_{A,j}$
$B \rightarrow A$: Pre-image $S_{B,j}$ of previous revocation hash $h_{B,j}$

Figure 5.3 Sequence of messages exchanged between two parties A and B to establish a Lightning Channel and send payments.

Alice sends Bob a new revocation hash $h_{A,j+1}$. Bob responds with a signature $\sigma^C_{B,j+1}$ for Alice's new *Commitment Transaction* $T^{C,A}_{j+1}$, and his new revocation hash $h_{B,j+1}$. Alice checks that the bitcoins are correctly distributed to each party in $T^{C,A}_{j+1}$ before sending her signature $\sigma^C_{A,j+1}$ for Bob's new *Commitment Transaction* $T^{C,B}_{j+1}$. Bob then also checks that the bitcoins are correctly distributed to each party. Once the channel's new state has been authorised, Alice sends the pre-image $S_{A,j}$ of her revocation hash to Bob, and this pre-image revokes her current *Commitment Transaction* $T^{C,A}_j$. Bob checks if the pre-image correctly revokes Alice's current *Commitment Transaction* before sending Alice the pre-image $S_{B,j}$ for his revocation hash $h_{B,j}$. Finally, Alice checks if this pre-image revokes Bob's current *Commitment Transaction* $T^{C,B}_j$.

Settle channel To settle the channel, both parties cooperate to sign and broadcast a transaction that sends each party their share of bitcoins. If either party does not cooperate, then the dispute is settled on the Blockchain. In a dispute, either party broadcasts their most recent *Commitment Transaction* $\tilde{T}^{C,A}_j$ and *Revocable Delivery Transaction* $T^{RD,A}_{j,\lambda}$. The counterparty must then broadcast their *Delivery Transaction* $T^{D,B}_j$ to receive their share of the coins.

5.3.3 Comparison of Duplex Micropayment and Lightning Channels

This section provides a comparison of Duplex Micropayment Channels and Lightning Channels. We focus on the computation, storage and network access required for both schemes before highlighting if resource-limited participants can outsource responsibility to a trusted third party. Finally, we discuss the total number of transactions that can be facilitated.

Expiry time and channel throughput. Table 5.1 highlights that both channels support a high-throughput compared to the earlier (and simpler) payment channels which have a

Table 5.1 A comparison of the high-level properties of all payment channel constructions. We consider whether a channel supports bi-directional payments, if the channel has an expiry time, whether one or both parties have a copy of the latest state and if the channel can support a high-throughput.

	Bidirectional	**No Expiry Time**	**Mutual Copy**	**High Throughput**
Unidirectional	✗	✗	✗	✗
Bidirectional	✓	✗	✗	✗
Duplex	✓	✗	◐	✓
Lightning	✓	✓	✓	✓

restricted throughput. In a unidirectional channel the coins can only be sent in one-direction which restricts the total number of coins that can be sent and in a bi-directional channel the throughput is restricted due to decrementing the channel's expiry time when changing payment direction. As a result there is a trade-off between small expiry times and increasing the channel's maximum throughput.

In Duplex Micropayment Channels, all payments are authorised within the pair of unidirectional channels which effectively mimics a bi-directional channel and relies on replace by incentive (i.e. the lock time is not decremented after every payment or when payment direction is changed). The channel still has an expiry time due to its reliance on an absolute lock time, but the expiry time is only reduced when resetting the balance of both unidirectional channels and the invalidation tree is used to reduce the impact of this reduction.

In Lightning Channel, all payments are replaced by revocation as both parties authorise the new channel state (i.e.. balance of both parties) before revoking the previous state (i.e. old balance of both parties). There is no requirement to decrement the lock time for a payment due to the revocation, but it does rely on a relative lock time. This allows the network to decide the final expiry time only when the channel is closed on the Blockchain without the counterparty's co-operation.

Mutual Copy of Latest State. Table 5.1 highlights whether one or both parties have a copy of the latest authorised payment which determines who is responsible for publishing the final payment during an uncooperative closure. All simple payment channels effectively rely on replace by incentive where only the receiver can broadcast the latest authorised payment and the sender must wait until the channel expires to close the channel (which may also reverse payments). Duplex Micropayment Channels can be considered in-between as both parties have a copy of the invalidation tree, but each party will only have a copy of the latest authorised payment in the unidirectional channel where they receive coins. Unlike the other payment channels, Lightning guarantees that both parties have a copy of the latest authorised

Table 5.2 The number of signatures required for each step in Duplex Micropayment Channels and Lightning Channels. Also, d represents each node in the *Invalidation Tree* and α is the number of replaced nodes in the *Invalidation Tree*.

	Set up	**Payment**	**Reset**	**Settle (Co-op)**	**Settle (Dispute)**
Duplex	$(d+2)\times 2$	1	$(\alpha+1)\times 2$	1×2	1×2
Lightning	2×2	1×2	0	1×2	3

payment due to the nature of replace by revocation as each party only revokes the old state after receiving a signed copy of the new state.

Blockchain privacy. A new pair of addresses A_1, B_1 is generated by both parties for the *Funding Transaction*'s multi-signature output. If both channels are closed cooperatively, then A_1, B_1 can be reused in the closing transaction that sends each party their final balance. In terms of privacy, an external Blockchain observer can see the number of bitcoins each address received, but not identify which receiving address A_1, B_1 corresponds to which depositing address A_0, B_0.

Throughout the lifetime of both schemes, A_1, B_1 can be reused in each intermediary transaction to minimise computing addresses. If a dispute is settled on the Blockchain, then a Duplex Micropayment Channel achieves the same privacy as using a single transaction to close the channel. While Lightning Channels also provide these privacy guarantees, they allow to identify which address A_1, B_1 raised the dispute by identifying the address that received bitcoins in the *Revocable Delivery Transaction*.

In Duplex Micropayment Channels, both parties can have a different pair of addresses for each unidirectional channel. By inspecting the outputs of both *Payment Transactions* it is not possible to derive which addresses belong to the same owner. It is also not possible to determine which party raised the dispute using the transactions from the Blockchain only. The parties in Lightning Channels can compute 3 addresses to be used for both sets of *Commitment Transactions*. For example, Alice's *Commitment Transaction*'s first output sends bitcoins to either A_1 or B_1, and the second output sends bitcoins to B_2. Her *Revocable Delivery Transaction* sends bitcoins to A_2 and Bob's *Delivery Transaction* sends bitcoins to B_3. Here, it is still possible to determine which addresses raised the dispute and also the final share of bitcoins each set of addresses received.

The number of signatures required in each payment channel is shown in Table 5.2. To establish the channel, both schemes require each party to sign a *Funding Transaction*. In Duplex Micropayment Channels each party must also sign d nodes in the *Invalidation Tree* and a *Payment Transaction* representing the unidirectional channel, whereas Lightning Channels require each party to sign an additional *Commitment Transaction*.

To send a new payment, the Duplex Micropayment Channels require a single signature from the sender. If either unidirectional channel has exhausted its supply of bitcoins, then both parties cooperate to reset the balance of both channels. This reset requires each party to sign α replacement transactions in the *Invalidation Tree* and an additional signature for the new *Payment Transactions* that represent the unidirectional channels. Lightning Channels always require a single signature from both parties to authorise a new pair of *Commitment Transactions*. This has implications for popular hubs as Duplex Micropayment Channels do not require the hub's involvement to receive bitcoins (except to perform a reset), while the hub must sign every payment in Lightning Channels.

In both schemes parties can cooperatively close the channel by signing a transaction that settles the final balance. However, they must sign the remaining intermediary transactions if the channel is not closed cooperatively. The unsigned transactions in Duplex Micropayment Channels include the *Payment Transaction*, or a *Refund Transaction* if the counterparty does not sign their *Payment Transaction*. In Lightning Channels, either party can broadcast their *Commitment Transaction* and sign a *Revocable Delivery Transaction* to claim their bitcoins. In response, the counterparty must sign the *Delivery Transaction* to receive their share of bitcoins. If the *Commitment Transaction* was previously revoked, then the counterparty must instead sign a *Breach Remedy Transaction*.

Local storage requirements. In Duplex Micropayment Channels, both parties store $d+1$ transactions which include the current active branch in the *Invalidation Tree* and the most recently received *Payment Transaction* from the counterparty. In Lightning Channels, each party stores the pre-image for every revocation hash used by the counterparty and the current *Commitment Transaction*. To prevent the need to brute-force the revocation hash using all stored pre-images, parties should also store a mapping between each pre-image and corresponding revocation hash.

Storage in the Blockchain. When both parties cooperate, both schemes only require 2 transactions, the *Funding Transaction* and a *Settlement Transaction*, to be committed to the Blockchain. In the worst case, when parties do not cooperate, Duplex Micropayment Channels require $1+d+2$ transactions. The d transactions represent the *Invalidation Tree*'s active branch, plus the *Funding Transaction* and 2 additional transactions representing either *Payment Transactions* for the unidirectional channels or their *Refund Transactions*. Lightning Channels always require 4 transactions. Besides the *Funding Transaction*, this includes a *Commitment Transaction*, the corresponding *Delivery Transaction* and either a *Breach Remedy Transaction* or a *Revocable Delivery Transaction*.

Frequency of access to the Blockchain. Duplex Micropayment Channels rely on the *Replace by Timelock* rule to ensure that only the latest authorised branch in the *Invalidation*

Tree is accepted into the Blockchain. Therefore, after establishing the channel, neither party needs to access the Blockchain until the channel has expired. Then, however, it is important to push the *d* transaction in the latest branch and the *Payment Transaction* that sends the party their bitcoins as soon as they become valid. Lightning Channels require both parties to periodically scan the Blockchain for previously revoked transactions. The frequency of monitoring is based on the mutually agreed *Relative Lock Time* that delays the *Revocable Delivery Transaction*'s acceptance into the Blockchain.

Outsourcing responsibilities is possible in both schemes to reduce the burden for resource-limited parties. In Duplex Micropayment Channels, a trusted third party can be responsible for broadcasting the active branch of the *Invalidation Tree* and the *Payment Transaction* that sends the party their bitcoins once the channel has expired. The branch must be broadcast within the safety-margin Δ time span, otherwise previously authorised branches become spendable. In Lightning Channels, the trusted third party is provided with signed *Breach Remedy Transactions*, pre-images of revocation hashes and the identification hash of previously revoked *Commitment Transactions*. It is then responsible for monitoring the Blockchain for the previously revoked *Commitment Transactions* and broadcasting the corresponding *Breach Remedy Transactions*.

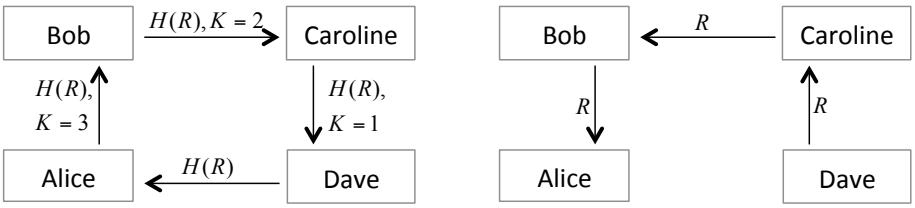
The total number of transactions that can occur in Duplex Micropayment Channels is based on the number of bitcoins sent, the frequency of resets to replenish the unidirectional channels, the depth of the tree, and the *Absolute Lock Time* associated with each node. In comparison, the only limitation for Lightning Channels is the bidirectional activity between both parties.

5.4 Routing Payments

We present how to fairly exchange bitcoins across several payment channels with Hashed Time-Locked Contracts (HTLCs) before discussing how to mitigate routing interruptions and the challenge of route discovery on the payment network.

5.4.1 Hashed Time-Locked Contract (HTLC)

Hashed Time-Locked Contracts fairly exchange bitcoins across several payment channels. They allow two parties to open channels with separate Payment Service Providers (PSP), and then construct a route of payment channels that connects both channels. If we have three channels, Alice $\rightarrow$ Bob, Bob $\rightarrow$ Caroline and Caroline $\rightarrow$ Dave, then Alice can send bitcoins



(a) Setting up HTLC (b) Synchronizing HTLC balances

Figure 5.4 (a) The final receiver provides the initial sender with the HTLC hash $H(R)$. This is shared along the HTLC route and the HTLC transfer's *Absolute Lock Time k* is decremented for each hop. (b) Outlines how the pre-image R is revealed in the reverse order for each hop to claim their bitcoins once the final receiver is given $H(R)$. For illustration the decrement time is $\omega = 1$.

to Dave, via Bob and Caroline. This fair exchange guarantees that intermediary hops receive their bitcoins, once they have sent their bitcoins to the next hop.

Figure 5.4 outlines the exchange of messages required to fairly exchange bitcoins across all intermediary channels. Once a route has been selected, the final receiver (Dave) provides the initial sender (Alice) with an HTLC hash $H(R)$. The initial sender commits bitcoins in a payment channel shared with their PSP under the condition that the pre-image R is revealed within k blocks. Each intermediary along the route decrements the lock time k and repeats this commitment with the next hop. The lock time k is decremented by ω to provide the intermediary a timespan for their bitcoins to be sent to the next hop and to allow them to claim their bitcoins from the previous hop. The last hop is the final receiver, who receives the payment amount contingent upon providing R.

The final receiver can claim the bitcoins by creating a transaction that reveals R and spends the HTLC output. Revealing R then allows the next party to also claim the bitcoins from their previous hop. However, it is not necessary to claim the outputs using Bitcoin's blockchain. Instead, both parties may simply agree to update their shared channel to reflect the new balance. This can be done along the route: every pair of participants updates their channels until the initial sender's bitcoins are taken.

In Duplex Micropayment Channels a new HTLC output is either added as part of a new *Payment Transaction* in the unidirectional channel or as an additional output to the leaf node $T_{d,k}$ in the *Invalidation Tree* (cf. Figure 5.5). This HTLC output is claimable if either of the following two conditions is satisfied:

1. The first condition requires a signature from both parties.

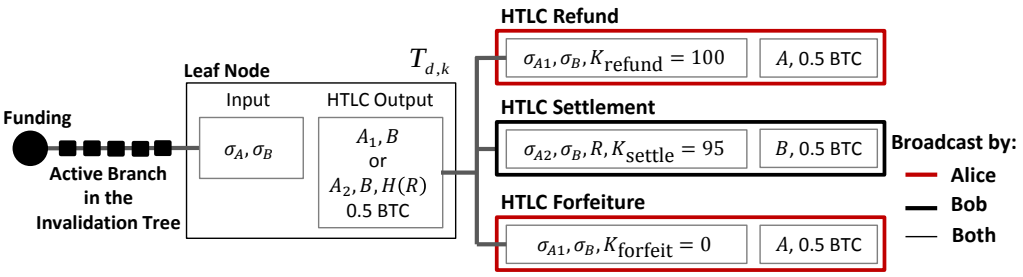


Figure 5.5 The HTLC is attached as an additional output to the *Invalidation Tree*'s leaf node. We have omitted unidirectional channels for space. The *HTLC Refund Transaction* guarantees that the bitcoins are returned to the sender if R is not revealed by block k_{refund}, the *HTLC Settlement Transaction* sends bitcoins to the receiver if the pre-image R of the HTLC hash is revealed by block k_{settle}, and the *HTLC Forfeiture Transaction* can later be signed to cancel the transfer.

2. The second condition requires a signature from both parties[4] and the pre-image R of the HTLC hash $H(R)$.

This HTLC output can be spent using either of the following three transactions. The *HTLC Refund Transaction* guarantees that the bitcoins are returned to the sender by k_{refund} and satisfies the first condition of the HTLC output. The *HTLC Settlement Transaction* requires the pre-image R of the HTLC hash and sends bitcoins to the counterparty. This transaction has an *Absolute Lock Time* k_{settle} to delay the receiver claiming the bitcoins to ensure that the HTLC output can later be cancelled if both parties agree. This transaction spends the second condition of the HTLC output. The *HTLC Forfeiture Transaction* has no lock time and is signed by both parties to cancel the HTLC transfer. It spends the first condition of the HTLC output and guarantees that the bitcoins are returned to the sender, even if the pre-image R of the HTLC hash is revealed.

It is the responsibility of the initial sender to compute the lock time k_{refund} for each hop along the route. The sender needs to ensure that each hop's k_{refund} is greater than the hop's chosen k_{settle} (i.e. $k_{\text{refund}} > k_{\text{settle}}$). Both lock times must also be greater than the hop's leaf node $T_{d,k}$ lock time k_{leaf} (i.e. $k_{\text{settle}} > k_{\text{leaf}}$). This is required as a refund or settlement cannot happen until the leaf node $T_{d,k}$ is included in Bitcoin's blockchain. Also, a timespan between the leaf node's lock time k_{leaf}, and the settlement lock time k_{settle} is required to allow the hop to cancel the HTLC transfer using a *HTLC Forfeiture Transaction*. This means that the bitcoins are potentially locked for the entire lifetime of the channel.

The initial sender's k_{refund} is passed to the next hop on the route, who should decrement k_{refund} by ω for use in their channel. This repeats as each hop passes their k_{refund} to the

[4]The sender must have a different Bitcoin address for each condition in the HTLC output, otherwise the receiver can use the signature to satisfy either condition.

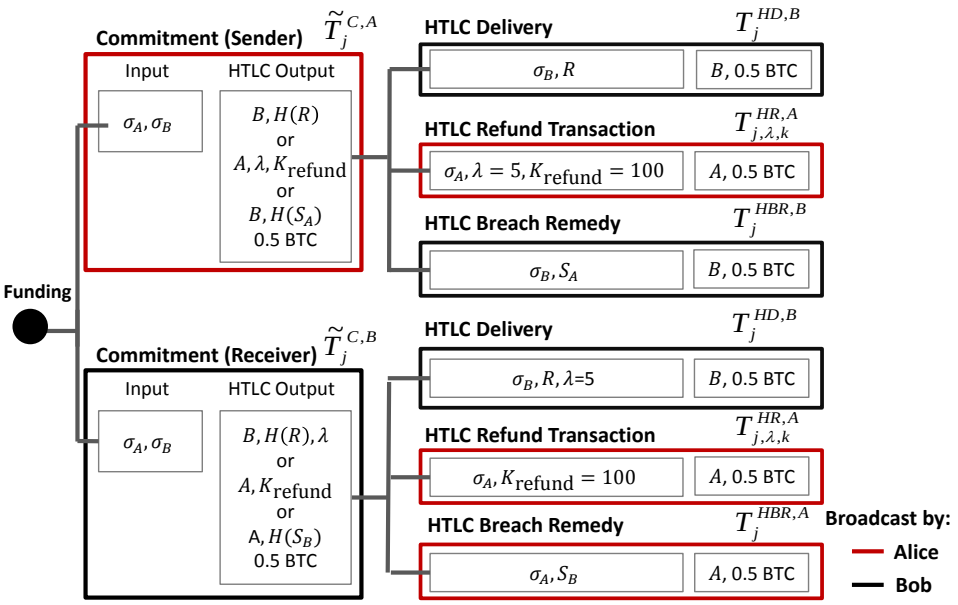


Figure 5.6 The HTLC is an additional output in both *Commitment Transactions* and we have omitted the other outputs for space. The delivery sends bitcoins to the receiver if they know R, the refund guarantees that the bitcoins are returned to the sender, and the breach remedy sends bitcoins to the counterparty whom did not broadcast a revoked *Commitment Transaction*.

next hop. The initial sender's k_{refund} must therefore take into account the largest leaf node's lock time k_{leaf} in the route. In the worst case, if the largest k_{leaf} is associated with the final receiver's channel, then the lock time k_{refund} for all hops along the route must also be larger than the largest k_{leaf}.

Finally, the hash $H(R)$ and the pre-image R can be discarded once the HTLC transfer has been forfeited or a new active branch in the *Invalidation Tree* is mutually agreed to cancel the HTLC transfer.

In Lightning Channels, a new HTLC output is associated with both *Commitment Transactions* in a new channel state (cf. Figure 5.6). The broadcaster of a *Commitment Transaction* is restrained by a *Relative Lock Time* λ before she can claim the bitcoins in the HTLC output. This provides the counterparty an opportunity to claim the bitcoins. We explain the role of λ once the commonality of the HTLC output's redemption criteria for each *Commitment Transaction* has been presented. The bitcoins can be claimed if either of the following three conditions is satisfied:

1. The first condition requires a signature from the receiver and the pre-image R of the HTLC hash $H(R)$.

2. The second condition requires a signature from the sender once the *Absolute Lock Time* k_{refund} has expired.

3. The third condition requires a signature from the counterparty that did not broadcast the revoked *Commitment Transaction*, and the pre-image S of the broadcaster's revocation hash $H(S)$.

The broadcaster of a *Commitment Transaction* must consider both the *Absolute Lock Time* k_{refund} that dictates when the HTLC transfer expires, and the *Relative Lock Time* λ which requires the broadcaster's *Commitment Transaction* to achieve a depth of λ blocks in the Blockchain before the broadcaster can claim their bitcoins. To illustrate the additional burden of λ for either party's redemption criteria, we consider the sender's and receiver's *Commitment Transaction* in turn.

The sender only broadcasts their *Commitment Transaction* to claim a refund using the *HTLC Refund Transaction* if the HTLC transfer expires, satisfying the second condition. To take the delay caused by λ into account, the sender's *Commitment Transaction* must be accepted into the Blockchain at block height $k_{\text{refund}} - \lambda$ for the *HTLC Refund Transaction* to become spendable at the correct time. If the refund is not claimed at the correct time, then R might be later revealed, and the sender is not guaranteed to receive the bitcoins from the previous hop. This λ provides a timespan for the counterparty to claim the bitcoins using the pre-image R of the HTLC hash $H(R)$, satisfying the first condition, or with the pre-image S_A of the revocation hash $H(S_A)$, satisfying the third condition.

The receiver only broadcasts their *Commitment Transaction* if they know the pre-image R of the HTLC hash. The bitcoins are redeemed using the *HTLC Delivery Transaction* that satisfies the first condition. To take the delay caused by λ into account, the receiver's *Commitment Transaction* must be accepted into Bitcoin's blockchain at block height $k_{\text{refund}} - \lambda - \Delta$. This makes certain that the *HTLC Delivery Transaction* has a safety margin Δ timespan to be accepted into Bitcoin's blockchain before the HTLC transfer expires. This λ provides a timespan for the counterparty to claim the bitcoins if the transfer has expired or if the pre-image S_B of the receiver's revocation hash $H(S_B)$ has been revealed.

The initial sender is responsible for computing the lock time k_{refund} in advance for each hop. This k_{refund} is passed along the route, and each hop decrements k_{refund} by ω for use in their channel. However, the initial sender's k_{refund} must take into account the hop with the largest *Relative Lock Time* λ. In the worst case, if the largest λ is associated with the final receiver's channel, then the lock time k_{refund} for all channels must also include the largest λ. This potentially locks the bitcoins for a long period of time as resource-limited parties may require a large λ as it dictates the frequency in which each party must monitor the Blockchain. Also, the pre-image R can be discarded once both parties have exchanged the pre-images S_A, S_B of the revocation hashes $H(S_A), H(S_B)$.

5.4.2 Routing Interruptions

The routing is interruptible if an intermediary is unresponsive after accepting the HTLC transfer. The initial sender and final receiver should wait until the HTLC transfer expires before reattempting the transfer. To overcome this issue, Poon has proposed a rollback protocol [99] that allows the final receiver to refund the initial sender using a second route. This allows the initial sender to reattempt the HTLC transfer, and if the intermediary returns, the initial sender's bitcoins are refunded using the second route. This rollback is only performed if the final receiver has not revealed the first route's pre-image R_1 of the HTLC hash $H(R_1)$.

To perform the rollback, the initial sender provides the final receiver with a new HTLC hash $H(R_2)$ and an *Absolute Lock Time* k_{expire} that represents the expiry time for the initial sender's HTLC transfer with their immediate hop in the first route. The final hop in the second route sends the initial sender the refunded bitcoins and must have an expiry time k_{refund} that is greater than k_{expire}. This provides a timespan for the initial sender's bitcoins to be claimed in the first route, and for them to receive the refund in the second route.

Finally, the receiver commits bitcoins to the next hop under the condition that the pre-images of the first route's HTLC hash $H(R_1)$ and the sender's HTLC hash $H(R_2)$ are revealed. Each hop decrements the lock time $k_{\max}$ by ω, and commits bitcoins to the next hop if R_1, R_2 are revealed. The final hop is the sender who has knowledge of R_2, but can only claim the bitcoins if R_1 is revealed from the first HTLC transfer.

5.4.3 Challenges for Route Discovery

We now discuss the challenges that payment networks face for route discovery. These challenges include the type of connections available for routing bitcoins, the topology of the network, the difficulty in building reputation systems to help assess the risk of other nodes and the financial incentives for routing bitcoins.

Node connections on the network rely on the user's role. End users might exclusively connect to PSPs who are responsible for establishing channels with other PSPs to find routes on the network. How these channels are funded will also differ as end users might fund the channel with the PSP, while the PSP to PSP channels are mutually funded. Also, the total number of bitcoins in a channel restricts the bitcoins an end user can receive. For example, if the user and PSP share a channel $U \leftrightarrow P$ in which the PSP has a balance of x bitcoins, then the user can only receive up to x bitcoins without establishing a new channel. The receiver must send bitcoins to the PSP to receive further payments.

Route planning methods include source-routing in which the sender pre-determines the payment route, and per-hop routing where only the final destination is given to the network and each hop finds the best route.

The former approach simplifies calculating the fee and lock time for the transfer. It is compatible with onion-routing which only reveals the previous and next hop, and allows the sender to enforce the order in which each node is visited. However, this does not prevent a node using intermediaries to route the bitcoins to the next hop. It might also be possible to identify that the next hop is the final recipient using the remaining fee and lock time.

Per-hop routing outsources the route finding to the network and can adapt to changing network conditions. The final destination is revealed to each node to help them find a route, offering less privacy than source-routing. Nodes could potentially offer low (or negative) fees to encourage others to select them for routing and then sell the transaction data to interested third parties. In per-hop routing, estimating the total fee and maximum lock time is difficult as the route is not known in advance, and mechanisms must be put into place that prevent nodes from taking excessive fees.

The topology of the network hinges on the ease of becoming a PSP and the potential regulation as money transmitters.

A straightforward approach is a hub-and-spoke model [108] with thousands of well-connected hubs who share route-finding information amongst themselves. In this model, end users open channels with hubs, and it is the hub's responsibility to find a route to the receiver's hub. These hubs are expected to be reliable and fast which potentially results in negligible HTLC transfer halts. This is important for Duplex Micropayment Channels as bitcoins can be locked for the channel's remaining lifetime if a transfer halt occurs. Also, the usage of Duplex Micropayment Channels reduces the computational overhead for hubs as they are only required to sign HTLC transfers while sending bitcoins, and not to receive them. In this model, end-users may not be responsible for computing or revealing the pre-image R of the HTLC hash as it would allow them to halt transfers.

The fundamental issue with the hub-and-spoke model is that it may lack Bitcoin's open-membership property. Instead, the community's vision for a payment network is to allow anyone to become a PSP which requires a mesh network and gossip protocol to advertise PSP services. Poon has proposed that Bitcoin's blockchain can aid route discovery as the user can identify which PSPs share channels [108]. If a route has been found, there is no guarantee that the PSPs are reliable due to the nature of open-membership. Lightning Channels are more suitable for unreliable transfers as a failure only temporarily locks the bitcoins. However, the computation overhead increases as PSPs must sign both sending and receiving HTLC transfers.

To prevent routing failures naturally leads to a reputation system to assess the risk of using a PSP. However, in the event of a failure, it is arguably a non-trivial problem to determine the reason behind the halt due to the private nature of routing and the inability to inspect other payment channels. For example, if the hop $C \rightarrow D$ settles on the Blockchain, it is not possible to determine if Dave had knowledge of the pre-image R, refused to disclose it, or simply raised a dispute to tarnish the reputation of Caroline [106]. Also, it might not be possible to identify which hop halted the transfer when disputes are not settled on the Blockchain. Furthermore, revealing the internal channel state of intermediaries cannot always reveal the reason behind a halt. For example, it is not possible to identify the duration of time before R was disclosed.

Routing fees need to cover the costs for a PSP to operate a secure node. These costs potentially include a 'risk' charge based on the number of bitcoins maintained in their hot wallet[5] to facilitate transfers. The PSP might charge if they are required to deposit bitcoins into the channel for end-users (i.e. merchants) who expect to mostly receive bitcoins. Also, negotiating a fee for routing is not straightforward as some PSPs may fluctuate fees to move funds in a maxed-out channel in a certain direction [26]. How fees are negotiated depends on the topology of the network as PSPs might have direct connections with each hop in the route or rely on fees advertised via a gossip protocol. Finally, the fee structure for end-users can be regular installments or on a per-transfer basis.

5.5 Conclusion

In this chapter, we presented an overview of payment networks in Bitcoin. We compared two prominent proposals, Duplex Micropayment Channels and Lightning Channels before discussing how to fairly exchange bitcoins across two or more channels using Hashed Time-Locked Contracts. Finally, we highlighted challenges for route discovery in payment networks. It is our hope that this chapter will motivate other researchers to begin tackling the open problems associated with Blockchain-based payment networks.

It is important to note that this is the final chapter that focuses on Bitcoin. So far, we have highlighted authentication and scalability issues that arise in Bitcoin due to its decentralised nature and pursuit for financial privacy. The fact that these issues exist is remarkable considering that Bitcoin's sole (and simple) purpose is to trade a single asset (i.e. bitcoins). Furthermore, it is unfortunate that some solutions to these issues (as seen in this chapter) are complex due to their reliance on Bitcoin's limited scripting facility.

[5] A wallet is frequently accessed and potentially connected to the internet.

In the next chapter, this thesis focues on Ethereum which is inspired by Bitcoin, but supports expressive smart contracts. We empirically test whether cryptographic protocols can be implemented as smart contracts and the feasibility of executing these protocols on the Ethereum network. Remarkably, we will see that while Ethereum has significantly greater scalability issues than Bitcoin, the expressiveness of its scripting facility simplifies protocols that can be built and executed on Ethereum.

Chapter 6

A Smart Contract for Boardroom Voting with Maximum Voter Privacy

6.1 Introduction

Ethereum is the second most popular cryptocurrency with a $870m market capitalisation as of November 2016. It relies on the same innovation behind Bitcoin [90]: namely, the Blockchain which is an append-only ledger. The Blockchain is maintained by a decentralised and open-membership peer-to-peer network. The purpose of the Blockchain was to remove the centralised role of banks for maintaining a financial ledger. Today, researchers are trying to re-use the Blockchain to solve further open problems such as coordinating the Internet of Things [58], carbon dating [27], and healthcare [34].

In this chapter, we focus on decentralised internet voting using the Blockchain. E-voting protocols that support verifiability normally assume the existence of a public bulletin board that provides a consistent view to all voters. In practice, an example of implementing the public bulletin board can be seen in the yearly elections of the International Association of Cryptologic Research (IACR) [62]. They use the Helios voting system [1] whose bulletin board is implemented as a single web server. This server is trusted to provide a consistent view to all voters. Instead of such a trust assumption, we explore the feasibility of using the Blockchain as a public bulletin board. Furthermore, we consider a decentralised election setting in which the voters are responsible for coordinating the communication amongst themselves. Thus, we also examine the suitability of the Blockchain's underlying peer-to-peer network as a potential authenticated broadcast channel.

There already exist proposals to use a Blockchain for e-voting. The Abu Dhabi Stock Exchange is launching a Blockchain voting service [57] and a recent report [19] by the

Scientific Foresight Unit of the European Parliamentary Research Service discusses whether Blockchain-enabled e-voting will be a transformative or incremental development. In practice, companies such as The Blockchain Voting Machine [55], FollowMyVote [10] and TIVI [127] propose solutions that use *the Blockchain as a ballot box* to store the voting data.

These solutions achieve *voter privacy* with the involvement of a trusted authority. In FollowMyVote, the authority obfuscates the correspondence between the voter's real world identity and their voting key. Then, the voter casts their vote in plaintext. In TIVI, the authority is required to shuffle the encrypted votes before decrypting and counting the votes. In our work, we show that the voter's privacy does not need to rely on a central authority to decouple the voter's real world identity from their voting key, and the votes can be counted without the cooperation of a central authority. Furthermore, these solutions only use the Blockchain as an append-only and immutable global database to store the voting data. We propose that the network's consensus that secures the Blockchain can also enforce the execution of the voting protocol itself.

To date, both Bitcoin and Ethereum have inherent scalability issues. Bitcoin only supports a maximum of 7 transactions per second [30] and each transaction dedicates 80 bytes for storing arbitrary data. On the other hand, Ethereum explicitly measures computation and storage using a gas metric, and the network limits the gas that can be consumed by its users. As deployed today, these Blockchains cannot readily support storing the data or enforcing the voting protocol's execution for national scale elections. For this reason, we chose to perform **a feasibility study of a boardroom election** over the Blockchain which involves a small group of voters (i.e. 40 participants) whose identities are publicly known before the voting begins. For example, a boardroom election may involve stakeholders voting to appoint a new director.

We chose to implement the boardroom voting protocol as a smart contract on Ethereum. These smart contracts have an expressive programming language and the code is stored directly on the Blockchain. Most importantly, all peers in the underlying peer-to-peer network independently run the contract code to reach consensus on its output. This means that voters can potentially not perform all the computation to verify the correct execution of the protocol. Instead, the voter can trust the *consensus computing* provided by the Ethereum network to enforce the correct execution of the protocol. This enforcement turns *detection* measures seen in publicly verifiable voting protocols into *prevention* measures.

Our contributions. We provide the first implementation of a decentralised and self-tallying internet voting protocol. The Open Vote Network [51] is a board-room scale voting protocol that is implemented as a smart contract in Ethereum. The Open Vote Network provides maximum voter privacy as an individual vote can only be revealed by a full-collusion

attack that involves compromising all other voters; all voting data is publicly available; and the protocol allows the tally to be computed without requiring a tallying authority. Most importantly, our implementation demonstrates the feasibility of using the Blockchain for decentralised and secure e-voting.

6.2 Background

6.2.1 Self-Tallying Voting Protocols

Typically, an e-voting protocol that protects the voter's privacy relies on the role of a trustworthy authority to decrypt and tally the votes in a verifiable manner. E-voting protocols in the literature normally distribute this trust among multiple tallying authorities using threshold cryptography; for example, see Helios [1]. However, voters still need to trust that the tallying authorities do not collude altogether, as in that case, the voter's privacy will be trivially breached.

Remarkably, Kiayias and Yung [66] first introduced a *self-tallying* voting protocol for boardroom voting with subsequent proposals by Groth [48] and Hao et al. [51]. A self-tallying protocol converts tallying into an open procedure that allows any voter or a third-party observer to perform the tally computation once all ballots are cast. This removes the role of a tallying authority in an election as anyone can compute the tally without assistance. These protocols provide *maximum ballot secrecy* as a full collusion of the remaining voters is required to reveal an individual vote and *dispute-freeness* that allows any third party to check whether a voter has followed the voting protocol correctly. Unfortunately, self-tallying protocols have a fairness drawback as the last voter can compute the tally before anyone else[1] which results in both *adaptive* and *abortive* issues.

The adaptive issue is that knowledge of the tally can potentially influence how the last voter casts their vote. Kiayias and Yung [66] and Groth [48] propose that an election authority can cast the final vote which is excluded from the tally. However, while this approach is applicable to our implementation discussed later, it effectively re-introduces an authority that is trusted to co-operate and not to collude with the last voter. Instead, we implement an optional round that requires all voters to hash their encrypted vote and store it in the Blockchain as a commitment. As a result, the final voter can still compute the tally, but is unable to change their vote.

[1] It is also possible for voters that have not yet cast their vote to collude and compute the partial tally of the cast votes. For simplicity, we discuss a single voter in regards to the fairness issue.

The abortive issue is that if the final voter is dissatisfied with the tally, they can abort without casting their vote. This abortion prevents all other voters and third parties from computing the final tally. Previously, Kiayias and Yung [66] and Khader et al. [65] proposed to correct the effect of abortive voters by engaging the rest of the voters in an additional recovery round. However, the recovery round requires full cooperation of all the remaining voters, and will fail if any member drops out half-way. We highlight that the Blockchain and smart contracts can enforce a financial incentive for voter participation using a deposit and refund paradigm [67]. This allows providing a new countermeasure to address the abortive issue: all voters deposit money into a smart contract to register for an election and are refunded upon casting their vote. Any voter who does not vote before the voting deadline simply loses their deposit.

In the next section we present Open Vote Network[51] before discussing its smart contract implementation on Ethereum. We chose to implement this protocol instead of others (e.g., [66, 48]) because it is the most efficient boardroom voting protocol in the literature in each of the following aspects: the number of rounds, the computation load per voter and the bandwidth usage [51]. As we will detail in Section 6.3, the efficiency of the voting protocol is critical as even with the choice of the most efficient boardroom voting protocol, its implementation for a small-scale election is already nearing the capacity limit of an *existing* Ethereum block.

6.2.2 The Open Vote Network Protocol

The Open Vote Network is a decentralized two-round protocol designed for supporting small-scale boardroom voting. In the first round, all voters register their intention to vote in the election, and in the second round, all voters cast their vote. The systems assumes an authenticated broadcast channel is available to all voters. The self-tallying property allows anyone (including non-voters) to compute the tally after observing messages from the other voters. In this chapter, we only consider an election with two options, e.g., yes/no. Extending to multiple voting options, and a security proof of the protocol can be found in [51].

A description of the Open Vote Network is as follows. First, all n voters agree on (G, g) where G denotes a finite cyclic group of prime order q in which the Decisional Diffie-Hellman (DDH) problem is intractable, and g is a generator in G. A list of eligible voters $(P_1, P_2, ..., P_n)$ is established and each eligible voter P_i selects a random value $x_i \in_R \mathbb{Z}_q$ as their private voting key.

Round 1. Every voter P_i broadcasts their voting key g^{x_i} and a (non-interactive) zero knowledge proof $ZKP(x_i)$ to prove knowledge of the exponent x_i on the public bulletin board.

$ZKP(x_i)$ is implemented as a Schnorr proof [111] made non-interactive using the Fiat-Shamir heuristic [42].

At the end, all voters check the validity of all zero knowledge proofs before computing a list of reconstructed keys:

$$Y_i = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}$$

Implicitly setting $Y_i = g^{y_i}$, the above calculation ensures that $\sum_i x_i y_i = 0$.

Round 2. Every voter broadcasts $g^{x_i y_i} g^{v_i}$ and a (non-interactive) zero knowledge proof to prove that v_i is either no or yes (with respect to 0 or 1) vote. This one-out-of-two zero knowledge proof is implemented using the Cramer, Damgård and Schoenmakers (CDS) technique [29].

All zero knowledge proofs must be verified before computing the tally to ensure the encrypted votes are well-formed. Once the final vote has been cast, then anyone (including non-voters) can compute $\prod_i g^{x_i y_i} g^{v_i}$ and calculate $g^{\sum_i v_i}$ since $\prod_i g^{x_i y_i} = 1$ (see [51]). The discrete logarithm of $g^{\sum_i v_i}$ is bounded by the number of voters and is a relatively small value. Hence the tally of yes votes can be calculated subsequently by exhaustive search.

Note that for the election tally to be computable, all the voters who have broadcast their voting key in Round 1 must broadcast their encrypted vote in Round 2. Also note that in Round 2, the last voter to publish their encrypted vote has the ability to compute the tally before broadcasting their encrypted vote (by simulating that he would send a no-vote). Depending on the computed tally, he may change his vote choice. In our implementation, we address this issue by requiring all voters to commit to their votes before revealing them, which adds another round of commitment to the protocol.

The decentralised nature of the Open Vote Network makes it suitable to implement over a Blockchain. Bitcoin's blockchain could be used as the public bulletin board to store the voting data for the Open Vote Network. However, this requires the voting protocol to be externally enforced by the voters. Instead, we propose using Ethereum to enforce the voting protocol's execution. This is possible as conceptually Ethereum can be seen as a global computer that can store and execute programs. These programs are written as smart contracts, and their correct execution is enforced using the same network consensus that secures the Ethereum blockchain. Furthermore, its underlying peer-to-peer network can act as an authenticated broadcast channel.

6.3 The Open Vote Network over Ethereum

We present an implementation of the Open Vote Network over Ethereum. The code is publicly available[2]. Three HTML5/JavaScript pages are developed to provide a browser interface for all voter interactions. The web browser interacts with an Ethereum daemon running in the background. The protocol is executed in five stages, and requires voter interaction in two (and an optional third) rounds. We give an overview of the implementation in the following.

6.3.1 Structure of Implementation

There are two smart contracts that are both written in Ethereum's Solidity language. The first contract is called the voting contract. It implements the voting protocol, controls the election process and verifies the two types of zero knowledge proofs we have in the Open Vote Network. The second contract is called the cryptography contract. It distributes the code for creating the two types of zero knowledge proofs[3]. This provides all voters with the same cryptography code that can be used locally without interacting with the Ethereum network. We have also provided three HTML5/JavaScript pages for the users:

- **Election administrator** (`admin.html`) administers the election. This includes establishing the list of eligible voters, setting the election question, and activating a list of timers to ensure the election progresses in a timely manner. The latter includes notifying Ethereum to begin registration, to close registration and begin the election, and to close voting and compute the tally.
- **Voter** (`vote.html`) can register for an election, and once registered must cast their vote.
- **Observer** (`livefeed.html`) can watch the election's progress consisting of the election administrator starting and closing each stage and voters registering and casting votes. The running tally is not computable.

We assume that voters and the election administrator have their own Ethereum accounts. The Web3 framework is provided by the Ethereum Foundation to faciltiate communication between a user's web browser and their Ethereum client. The user can unlock their Ethereum account (decrypt their Ethereum's private key using a password) and authorise transactions

[2] https://github.com/stonecoldpat/anonymousvoting

[3] We have included the code to create and verify the two types of zero knowledge proofs in the cryptography contract. The code is independent of the Open Vote Network and can be used by other smart contracts.

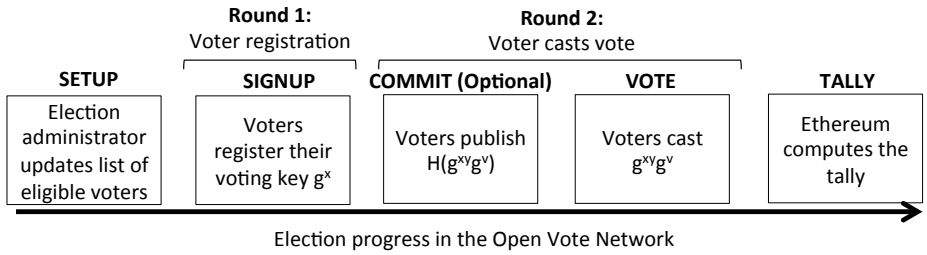


Figure 6.1 There are five stages to the election.

directly from the web browser. There is no need for the user to interact with an Ethereum wallet, and the Ethereum client can run in the background as a daemon.

6.3.2 Election stages

Figure 6.1 presents the five stages of the election in our implementation. The smart contract has a designated owner that represents the election administrator. This administrator is responsible for authenticating the voters with their user-controlled account and updating the list of eligible voters. A list of timers is enforced by the smart contract to ensure that the election progresses in a timely manner. The contract only allows eligible voters to register for an election, and registered voters to cast a vote. Furthermore, the contract can require each voter to deposit ether upon registration, and automatically refund the ether when their vote is accepted into the Blockchain. Each stage of the election is described in more detail below:

SETUP. The election administrator authenticates each voter with their user-controlled account and updates the voting contract to include a whitelist of accounts as eligible voters. He defines a list of timers to ensure that the election progresses in a timely manner:

- $t_{finishRegistration}$: all voters must register their voting key g^{x_i} before this time.
- $t_{beginElection}$: the election administrator must notify Ethereum to begin the election by this time.
- $t_{finishCommit}$: all voters must commit to their vote $H(g^{x_i y_i} g^{v_i})$ before this time. This is only used if the optional COMMIT stage is enabled.
- $t_{finishVote}$: all voters must cast their vote $g^{x_i y_i} g^{v_i}$ before this time.
- π: a minimum length of time in which the commitment and voting stages must remain active to give voters sufficient time to vote.

The administrator also sets the registration deposit d, the voting question, and if the optional COMMIT stage should be enabled. Finally, the administrator notifies Ethereum to transition from the SETUP to the SIGNUP stage.

SIGNUP. All eligible voters can choose to register for the vote after reviewing the voting question and other parameters set by the election administrator. To register, the voter computes their voting key g^{x_i} and $ZKP(x_i)$. Both the key and proof are sent to Ethereum alongside a deposit of d ether. Ethereum does not accept any registrations after $t_{finishRegistration}$. The election administrator is responsible for notifying Ethereum to transition from the SIGNUP to either the optional COMMIT or the VOTE stage. All voter's reconstructed keys $g^{y_0}, g^{y_1}, ..., g^{y_n}$ are computed by Ethereum during the transition.

COMMIT(Optional). All voters publish a hash of their vote $H(g^{x_i y_i} g^{v_i})$ to the Ethereum blockchain. The contract automatically transitions to the VOTE stage once the final commitment is accepted into the Blockchain.

VOTE. All voters publish their (encrypted) vote $g^{x_i y_i} g^{v_i}$ and a one-out-of-two zero knowledge proof to prove that v_i is either zero or one. The deposit d is refunded to the voter when their vote is accepted by Ethereum. The election administrator notifies Ethereum to compute the tally once the final vote is cast.

TALLY. The election administrator notifies Ethereum to compute the tally. Ethereum computes the product of all votes $\prod_i g^{x_i y_i} g^{v_i} = g^{\sum_i v_i}$ and brute forces the discrete logarithm of the result to find the number of yes votes.

As mentioned before, Open Vote Network requires all the registered voters to cast a vote to enable tally calculation. The deposit d in our implementation provides a financial incentive for registered voters to vote. This deposit is returned to the voter if they follow through with the voting protocol and do not drop out. The list of timestamps defined by the election administrator determines if the voter's deposit d is forfeited or refunded. There are three refund scenarios if a deadline is missed:

- Registered voters can claim their refund if the election does not begin by $t_{beginElection}$.
- Registered voters who have committed can claim their refund if not all registered voters commit to their vote by $t_{finishCommit}$.
- Registered voters can claim their refund if not all registered voters cast their vote by $t_{finishVote}$.

6.4 Design Choices

In this section, we discuss the design choices we made when developing the implementation. In particular, we elaborate on some attack vectors that are addressed in our smart contract and clarify the trust assumptions that are required for our implementation to be secure.

Individual and public verifiability. We assume that the voter's machine, including their web browser, is not compromised. The voter has an incentive to ensure their machine is secure. If the machine or web browser is compromised, the voter's ether is likely to be stolen. The voter can check that their vote has been *recorded as cast* and *cast as intended* by inspecting the Blockchain and decrypting their vote using the key x_i. Also, the voter, or any observer for that matter, can independently compute the tally to verify that the cast votes have been *tallied as recorded*. Unfortunately, this public verifiability does not provide any coercion resistance as the voting is conducted in a "unsupervised" environment. The voter may vote under the direct duress of a coercer who stands over their shoulder. The voter can also reveal x to prove how their vote was cast to others. As such, in a similar fashion to Helios [1], we note that our implementation is only suitable for low-coercion elections.

Voter authentication. Smart contracts can call other smart contracts. As a result, there exist two methods to identify the caller. The first is `tx.origin` that identifies the user-controlled account that authorised the transaction, and not the immediate caller. The second is `msg.sender` that identifies the immediate caller which can be a contract or a user-controlled address. Initially, a developer might use `tx.origin` as it appears the appropppriate choice to identify the voter. Unfortunately, this approach allows a malicious smart contract to impersonate the voter and register for an election.

To illustrate, a voter is given the interface to a smart contract called BettingGame. This lets the voter place a bet using `BettingGame.placeBid()`. Unknowingly to the voter, if this function is called, then BettingGame will call `TheOpenVoteNetwork.register()` and register a voting key on behalf of the voter. To overcome this issue, we recommend using `msg.sender` as it identifies the immediate caller whose address should be in the list of eligible voters.

Defending against replay attacks. All voting keys g^{x_i} and their zero knowledge proofs $ZKP(x_i)$ are publicly sent to the Ethereum blockchain. A potential attack is that another eligible voter can attempt to register the same voting keys by replaying g^{x_i} and $ZKP(x_i)$. This would also let them later copy the targeted voter's vote. We highlight that the commitment (i.e., input arguments to the hash function) in the zero knowledge proof includes `msg.sender` and Ethereum will not accept the zero knowledge proof $ZKP(x_i)$ if `msg.sender` does not match the account that is calling the contract. As such, it is not possible to replay another voter's key g^{x_i} without their co-operation. This also applies to the one-out-of-two zero knowledge proofs.

Blocking re-entrancy. A hacker recently exploited a re-entrancy vulnerability in theDAO to steal over 3.6 million ether. Luu et al highlight [72] that 186 distinct smart contracts stored on the Blockchain (including theDAO) are also potentially vulnerable. This attack

relies on the contract sending ether to the user before deducting their balance. The attacker can recursively call the contract in such a way that the sending of ether is repeated, but the balance is only deducted once. To prevent this attack, we follow the advice of Reitwiessner [102] to first deduct the voter's balance before attempting to send the ether.

The role of timers. The election administrator sets a list of timers to allow Ethereum to enforce that the election progresses in a timely manner. A minimum time interval π (unit in seconds) is set during the SETUP stage to ensure each stage remains active for at least a time interval of length π. In particular, the rules $t_{finishCommit} - t_{beginElection} > \pi$ and $t_{finishVote} - t_{finishCommit} > \pi$ are enforced to provide sufficient time for voters to commit to and cast their vote. Also, it provides a window for the voter's transaction to be accepted into the Blockchain. This is necessary to prevent a cartel of miners ($< 51\%$) attempting to censor some transactions. Voters need to check that π is not a small value such as $\pi = 1$. In this case, the voting stage can finish one second after the election begins. As a result, all voters are likely to lose their deposits. Of course, both the COMMIT and VOTE stage can finish early if all voters have participated.

The block's timestamp is used to enforce the above timers. Ethereum has a tight bound on the timestamp which must conform to the following two rules. First, a new block's timestamp must be greater than the previous block. Second, the block's timestamp must be less than the user's local clock. Furthermore, the miner's ability to drift a block's timestamp by 900 seconds (15 minutes) as reported in [72] is no longer possible [35].

Ethereum miners. The tip of the Blockchain is uncertain and the state of a contract at the time of signing a transaction is not guaranteed to remain the same. Furthermore, miners control the order of transactions in a block, and can control the order of a contract's execution if there are two or more transactions calling the same contract. Although the order of voting keys or casting a vote does not matter, the order of transactions is important if a timer is about to expire. For example, if the voter attempts to register around the time that $t_{finishRegistration}$ expires, then the miner can prevent the registration in two ways. First, the miner can choose a block timestamp that expires the $t_{finishRegistration}$ timer. Second, if the miner has the voter's registration transaction and the election administrator's begin election transaction, he can order the transactions in the block such that the smart contract begins the election before allowing the voter to register for the election. Unfortunately, in both cases, the voter's registration will fail.

It is important that voters authorise their transactions in good time before the stage is destined to end. We must assume that the majority of miners are not attempting to disrupt the election. A smaller cartel of miners ($< 51\%$) can potentially delay transactions being accepted into the Blockchain using techniques such as selfish mining [40][109] or feather

forking [94]. This ability of miners to delay a transaction is a fundamental problem for any contract.

The election administrator. An election administrator is required to add voters to the list of eligible voters, set the election's parameters and to begin the registration stage. Unfortunately, smart contracts cannot execute code without the notification of an external user-controlled account. As such, a user is still required to notify the smart contract to begin the election and compute the tally. Deciding who is responsible for notifying Ethereum is an implementation trade-off and we have assumed it is the election administrator's role. If necessary, the contract can be modified to incorporate a time-out where if the election adminstration does not complete their task (e.g. notify the contract to compute the tally) within a time period, then any registered voter canperform the notification. However, in that case it is possible that two or more voters attempt to notify Ethereum at the same time and broadcast transactions to the network. If this happens, only one transaction can begin the election or compute the tally. All unsuccessful transactions will still be stored in the Blockchain and all the broadcasting users will still be charged transaction fees.

Removing the COMMIT stage. The COMMIT stage prevents the final voter computing the tally and using this information to decide how to vote. It is possible to remove this stage if we require the election administrator (or a separate external party) to perform some extra tasks. In this case, the administrator is the first voter to register a voting key g^x and deposit of d ether before voter registration begins. Next, he is required to merely reveal his secret x once all voters have cast their vote. Revealing x allows Ethereum to calculate a final dummy vote and compute the tally. The administrator is now trusted not to collude with the last voter. This approach removes the COMMIT phase but requires extra an trust assumption.

Do voters need to use Ethereum? Today, all voters need to download the full Ethereum blockchain to confirm the voting protocol is being executed correctly. In the future, voters will be able to use the Light Ethereum Subprotocol (LES) [36] which is similar to Bitcoin's simplified payment verification (SPV) protocol. In LES, voters will only verify the voting protocol's state and not be required to store the full Blockchain.

Most importantly, it is possible for the voter to participate in the voting protocol without the full Blockchain. In this case, the voter merely broadcasts their transactions and trusts the consensus mechanism of the Ethereum network to enforce the correct execution of the protocol. This would enable voters who have devices with limited resources to vote in our implementation. We have provided `livefeed.html` to allow voters to visit an external website and confirm their registration or vote has been recorded in the Blockchain.

Entity: Transaction	Cost in Gas	Cost in $
A: VoteCon	3,779,963	0.83
A: CryptoCon	2,435,848	0.54
A: Eligible	2,153,461	0.47
A: Begin Signup	234,984	0.05
V: Register	763,118	0.17
A: Begin Election	3,085,449	0.68
V: Commit	70,112	0.02
V: Vote	2,490,412	0.55
A: Tally	746,485	0.16
Administrator Total	12,436,190	2.74
Voter Total	3,323,642	0.73
Election Total	145,381,858	31.98

Table 6.1 A breakdown of the costs for 40 participants using the Open Vote Network. We have approximated the cost in USD ($) using the conversion rate of 1 ether = $11 and the gas price of 0.00000002 ether which are the real world costs in November 2016. Also, we have identified the cost for the election administrator 'A' and the voter 'V'.

6.5 Experiment on Ethereum's Test Network

Our implementation was deployed on Ethereum's official test network that mimics the production network. We sent 126 transactions to simulate forty voters participating in the protocol. Each transaction's computational and financial cost is outlined in Table 6.1. Each transaction by the election administrator (denoted by the prefix 'A:' in the table) is broadcast only once, and each transaction by a voter (denoted by the prefix 'V:' in the table) is broadcast once per voter, i.e., a total of 40 times. Also, we have rounded the cost in US Dollars (denoted by $) to two decimal places.

We had to split the Open Vote Network into two contracts as the code was too large to store in an Ethereum block which has a capacity of approximately 4.7 million gas. The voting contract VoteCon (80% of block capacity, and $0.83 transaction fee) contains the protocol logic. The cryptography contract CryptoCon (52% of block capacity, and $0.54 transaction fee) contains the code to create and verify the two types of zero knowledge proofs we have in the protocol.

CryptoCon can be reused by other contracts requiring similar zero knowledge proofs. It is important to note that the code for computing the zero knowledge proofs is run locally on the voter's machine, and no transactions are sent to the network. CryptoCon's purpose is to ensure that all voters have access to the same cryptography code.

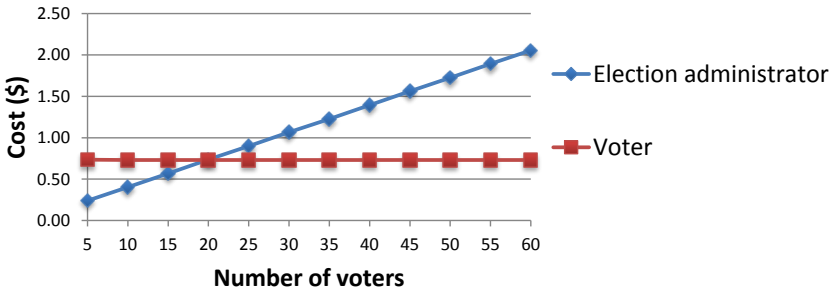


Figure 6.2 The average cost for the election administrator and the voter based on the number of voters participating in the election.

As the figures show, voter registrations and vote casting cost around 16% and 53% of block capacity, respectively. This suggests that the current block sizes in Ethereum support at most six voter registration per block and at most one vote casting per block. Recall that blocks are currently generated in Ethereum at a rate of one block per 12 seconds.

Overall, running the election with 40 voters costs the election administrator $2.74. The total election cost including the cost for the administrator and the voters is $31.98 which breaks down to a reasonable cost of $0.73 per voter.

To see how the cost for the election administrator and the voter vary with different number of voters we have carried out experiments with 5, 10, 15, . . . , 60 voters. Figure 6.2 highlights the distribution of cost for the election administrator and the voter based on the number of voters participating in the election. This shows that the election administrator's cost increases linearly based on the number of voters, and the voter's cost remains constant.

All testing was performed on the test network due to an ongoing DoS attack, starting from 22 September 2016, on Ethereum's production network [22]. Miners set the block's gas limit to 1,500,000 gas to reduce the impact on the network and a hard fork [21] was deployed on 18 October 2016 to prevent the attack. However, a second DoS attack began on 19 October 2016. Ethereum developers have recommended a temporary gas limit of 2,000,000 until the next scheduled hard fork. As such, the Open Vote Network cannot run on the production network at this time.

6.5.1 Timing Analysis

Table 6.2 outlines the timing analysis measurements for tasks in the Open Vote Network. All measurements were performed on a MacBook Pro running OS X 10.10.5 equipped with 4 cores, 2.3GHz Intel Core i7 and 16 GB DDR3 RAM. All time measurements are rounded up to the next whole millisecond. We use the Web3 framework to facilitate communication

Action	Avg. Time (ms)
Create ZKP(x)	81
Register voting key	142
Begin election	277
Create 1-out-of-2 ZKP	461
Cast vote	573
Tally	132

Table 6.2 A time analysis for actions that run on the Ethereum daemon.

between the web browser and the Ethereum daemon. All tasks are executed using `.call()` that allows us to measure the code's computation time on the local daemon.

The cryptography smart contract is responsible for creating the zero knowledge proofs for the voter. The time required to create the proofs is 81 ms for the Schnorr proof and 461 ms for the one-out-of-two zero knowledge proof. These actions are always executed using `.call()` as this contract should never receive transactions.

The voting smart contract is responsible for enforcing the election process. Registering a vote involves verifying the Schnorr zero knowledge proof and in total requires 142 ms. To begin the election requires computing the reconstructed public keys which takes 277 ms in total for forty voters. Casting a vote involves verifying the one-out-of-two zero knowledge proof which requires 573 ms. Tallying involves summing all cast votes and brute-forcing the discrete logarithm of the result and on average takes around 132 ms.

We decided to distribute the cryptography code using the Ethereum blockchain to allow all voters to use the same code. Running the code on the voter's local daemon is significantly slower than using a seperate library such as OpenSSL. For example, creating a Schnorr signature using OpenSSL on a comparable machine requires 0.69 ms [77]. This slowness is mostly due to the lack of native support for elliptic curve math in Ethereum smart contracts. The Ethereum Foundation has plans to include native support and we expect this to significantly improve our reported times.

6.6 Discussion on Technical Difficulties

In this section, we discuss the difficulties faced while implementing the Open Vote Network on Ethereum.

Lack of support for cryptography. Ethereum supports up to 256-bit unsigned integers. For this reason, we chose to implement the protocol over an elliptic curve instead of a finite field. However, Solidity does not currently support Elliptic Curve cryptography, and we had

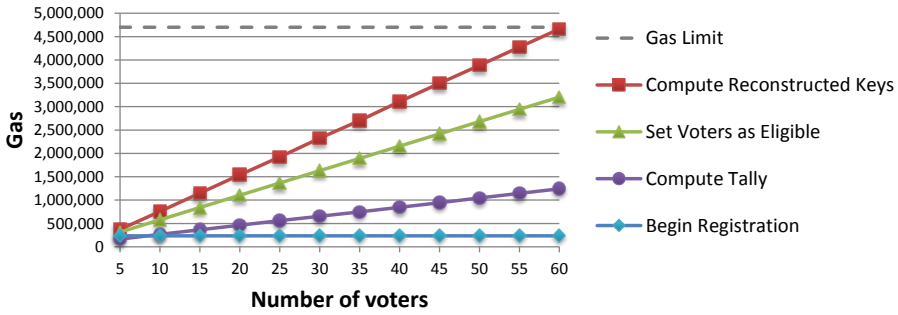


Figure 6.3 The gas cost for the election administrator based on the number of voters participating in the election.

to include an external library to perform the computation. Including the library led to our voting contract becoming too large to store on the Blockchain. As previously discussed, we separated the program into two smart contracts: one voting contract and one cryptography contract. The cryptographic computations are expensive and this results in a block only being able to support six voter registrations, or a single vote.

Call stack issues. The call stack of a program has a hard-coded limit of 1024 stack frames. This limits the amount of local memory available, and the number of function calls allowed. These limitations led to difficulty while implementing the 1-out-of-2 ZKP as the temporary variables typically required exceeded the hard-coded limit of 16 local variables [61]. We had to use variables extremely sparingly to ensure that the 1-out-of-2 ZKP could be implemented.

Lack of debugging tools. The Mix IDE that provides a solidity source code debugger has been discountined [37] and could not be used for our work. Remix is the replacement for the Mix IDE and it provides a debugger for contracts at the assembly level, but this is too low for debugging Solidity contracts. Instead, we had to create `Events` that log data along with the contract to help with debugging which is incorporated into the contract before deployment.

Mitigate loss of voting key. The voting key is kept secret by the voter and needs to be stored on their local machine. This is important to ensure that if the voter's web browser crashes or is closed, then the voting key is not lost. We provide a standalone Java program `votingcodes.jar` to generate the voting key and store it in `votingcodes.txt`. The voter is required to upload this file to their web browser.

Maximum number of voters. Figure 6.3 demonstrates the results of our experiment and highlights the breakdown of the election administrator's gas consumption. Except for opening registration, the gas cost for each task increases linearly with the number of voters. The gas limit for a block was set at 4.7 million gas by the miners before the recent DoS attacks. This means that the smart contract reaches the computation and storage limit if it

is computing the voter's reconstructed keys for around sixty registered voters. This limit exists as all keys are computed in a single transaction and the gas used must be less than the block's gas limit. To avoid reaching this block limit, we currently recommend a safe upper limit of around 50 voters. However, the contract can be modified to perform the processing in batches and allow multiple transactions to complete the task.

6.7 Conclusion

In this chapter, we have presented a smart contract implementation for the Open Vote Network that runs on Ethereum. Our implementation was tested on the official Ethereum test network with forty simulated voters. We have shown that our implementation can be readily used with minimal setup for elections at a cost of $0.73 per voter. The cost can be considered reasonable as this voting protocol provides maximum voter privacy and is publicly verifiable. This is the first implementation of a decentralised internet voting protocol running on a Blockchain. It uses the Ethereum blockchain not just as a public bulletin board, but more importantly, as a platform for *consensus computing* that enforces the correct execution of the voting protocol.

Chapter 7

Conclusion

7.1 Summary

This thesis explored bootstrapping trust from the blockchain in order to build and run cryptographic applications. Remarkably, each application leveraged the blockchain in subtly different ways. We provide a final summary on how the blockchain was used for each cryptographic application before concluding.

Chapters 3 and 4 relied on the blockchain's immutability for storing information. The former chapter stored each party's pseudonymous identity in the blockchain to bootstrap authenticated key exchange. We proposed two protocols Diffie-Hellman over Bitcoin and YAK over Bitcoin in response to observing that real-world merchants are unable to correctly re-authenticate customers that used Bitcoin as a payment method. On the other hand, the latter chapter relied on the blockchain for storing transaction information (i.e. origin/recipient of coins). This transaction substantiates the publicly verifiable evidence that is privately exchanged during the revised BIP70: Payment Protocol.

Chapter 5 re-purposed the blockchain to become an arbitrator for dispute resolution. This re-purposing is heralded as a scaling approach for cryptocurrencies. It permits two parties to store a depoist in the blockchain, privately transact (i.e. re-distribute each party's share of this deposit) and then confirm the aggregation of all payments in a single transaction. If there is a dispute, both parties can submit transactions to the blockchain (i.e. cryptographic evidence) within a grace period, and the blockchain will enforce the correct determination. Remarkably, the only academically published protocol is Duplex Micropayment Channels [33], whereas basic payment channels, Lightning Channels and Hashed Time-Locked Contracts are scattered across mailing lists, chat rooms and forums. Unfortunately, this increases the difficulty for researchers to find and comprehend this emerging field's state-of-the-art.

As such, the contribution and motivation for our survey was to concisely summarise payment channel protocols and provide future research directions.

Chapter 6 relied on the blockchain to enforce the cryptographic application's correct execution using the same consensus protocol that secures the blockchain. This allowed us to demonstrate the first practical implementation of a self-tallying and decentralised internet voting protocol as a smart contract for Ethereum. Most importantly, this research also provided the first empirical study of executing cryptographic protocols on the Ethereum network. This research concluded that the Etheruem network as deployed can support cryptography, but it is not ready for wide-spread use. For example, each encrypted vote in the Open Vote Network required 53% of a block's capacity which effectively only allows Ethereum to permit one vote every twelve seconds.

To conclude, our research focused on demonstrating the feasibility of running cryptographic applications that bootstrap trust from the Blockchain. Both the secure end-to-end communication protocols and the revised Payment Protocol relied on the blockchain's immutability for storing information in order to bootstrapping trust for the protocols. On the other hand, payment networks re-purposed the blockchain to become an arbitrator that could enforce the correct outcome for the protocol based on cryptographic evidence provided by each party. Finally, the Open Vote Network relied on the Blockchain to bootstrap correct enforcement of the protocol and to act as a central bulletin board that provides a consistent view to all overs.

7.2 Future work

Future work is suggested as follows.

- In Chapter 3, it would be useful to investigate if the authenticated key exchange protocols can be applied for decentralised marketplaces such as OpenBazaar[1]. The only guaranteed form of identity for the seller or buyer in these marketplaces is represented in terms of the bitcoin addresses involved in the transaction. Therefore, it is likely that these marketplaces are ideal for relying solely on the blockchain for authenticated and secure end-to-end communication.

- In Chapter 3 and 4, BIP70: Payment Protocol may require further revision before it is suitable for a payment network as there is no guarantee that the bitcoin transaction will be stored in the blockchain. However, the protocol execution transcript of a payment channel should permit any observor to verify that it was executed correctly

[1] https://www.openbazaar.org/

(i.e. dispute-freeness). This might be satisfactory as a basis for linking the bitcoin transaction to messages exchanged during the Payment Protocol.

- In Chapter 5, it appears possible to build an off-chain channel that maintains state (i.e. a state channel). Conceptually, both parties can authorise a third layer on top of Lightning Channels that represents a contract. Both parties can then co-operatively execute this contract off-chain and once the contract reaches its final state both parties can co-operatively reset the channel's state. This reset will involve updating the lighting channel to effectively invalidate the previous contract while atomically creating a new one.

- In Chapter 6, it is worth attempting to build the Open Vote Network in an Ethereum state channel. All parties can deposit coins into the state channel and perform the self-tallying election off-chain. In the event a single party aborts the off-chain protocol, then the election's latest state can be broadcast and accepted into the blockchain/voting contract. This triggers a timer that requires the aborted party to continue the protocol before a time-out otherwise their deposit is forfeited. The benefit of this approach is that the blockchain can be used as an arbitrator to enforce that all parties participate in the election while avoiding the expensive gas costs of running the election on-chain.

References

[1] Adida, B. (2008). Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348.

[2] Alcio (2015). Monitor pay to script hash adoption. http://p2sh.info/, Accessed on 21/05/15.

[3] Ali, S. T., McCorry, P., Lee, P. H.-J., and Hao, F. (2015). Zombiecoin: Powering next-generation botnets with bitcoin. In *Bitcoin Workshop at Financial Cryptography and Data Security*. Springer.

[4] Allison, I. (2015). Silk Road prosecutors talk about Bitcoin, Ripple and money laundering. *International Business Times*. http://www.ibtimes.co.uk/silk-road-prosecutors-talk-about-bitcoin-ripple-money-laundering-1517414.

[5] Andresen, G. (2012). Pay to Script Hash. *Bitcoin Improvement Process*. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki, Accessed on 07/12/15.

[6] Andresen, G. (2014). Conversation about OP_SCHNORRVERIFY. *Freenode IRC bitcoin-wizards*. https://botbot.me/freenode/bitcoin-wizards/, Accessed on 10/05/15.

[7] Andresen, G. (2015). BIP 101: Increase maximum block size. https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki, Accessed on 19/04/16.

[8] Andresen, G. and Hearn, M. (2013). BIP 70: Payment Protocol. *Bitcoin Improvement Process*. https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki, Accessed on 15/01/15.

[9] Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., and Capkun, S. (2013). Evaluating user privacy in bitcoin. In *Financial Cryptography and Data Security*, pages 34–51. Springer.

[10] Aradhya, P. (2016). Distributed Ledger Visible To All? Ready for Blockchain? *Huffington Post*. http://www.huffingtonpost.com/pradeep-aradhya/are-we-ready-for-a-global_b_9591580.html, Accessed on 19/04/16.

[11] A.Tanzarian (2014). Ethereum Raises 3,700 BTC in First 12 Hours of Ether Presale. https://cointelegraph.com/news/ethereum-raises-3700-btc-in-first-12-hours-of-ether-presale, Accessed on 30/12/16.

[12] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., and Wuille, P. (2014). Enabling Blockchain Innovations with Pegged Sidechains. https://blockstream.com/technology/sidechains.pdf, Accessed on 01/06/2017.

[13] Baddeley, M. (2004). Using e-cash in the new economy: An economic analysis of micro-payment systems. *Journal of Electronic Commerce Research*, 5(4):239–253.

[14] Barber, S., Boyen, X., Shi, E., and Uzun, E. (2012). Bitter To Better: How To Make Bitcoin a Better Currency. In *Financial Cryptography and Data Security*, pages 399–414. Springer.

[15] BBC (2015). New Paypal partnership enables limited Bitcoin payments. http://www.bbc.co.uk/news/technology-29341886, Accessed on 06/01/15.

[16] BitPay (2015). New Invoice Adjustment and Refund Flow. https://blog.bitpay.com/new-refund-flow/, Accessed on 20/09/15.

[17] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. (2015). Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (Oakland) 2015*. Springer.

[18] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. (2014). Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security*, pages 486–504. Springer.

[19] Boucher, P. (2016). What if blockchain technology revolutionised voting? *Scientific Foresight Unit (STOA), European Parliamentary Research Service*. http://www.europarl.europa.eu/RegData/etudes/ATAG/2016/581918/EPRS_ATA(2016)581918_EN.pdf.

[20] Buterin, V. (2013). A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper, Accessed on 30/12/16.

[21] Buterin, V. (2016a). Long-term gas cost changes for io-heavy operations to mitigate transaction spam attacks. *Ethereum Blog*. https://github.com/ethereum/EIPs/issues/150, Accessed on 01/11/16.

[22] Buterin, V. (2016b). Transaction spam attack: Next Steps. *Ethereum Blog*. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/, Accessed on 01/10/16.

[23] Carlsten, M., Kalodner, H., Weinberg, S. M., and Narayanan, A. (2016). On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167. ACM.

[24] Castillo, M. (2016). Ethereum Executes Blockchain Hard Fork to Return DAO Funds. http://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds/, Accessed on 3/01/17.

[25] Certicom Research (2000). SEC 2: Recommended Elliptic Curve Domain Parameters. *Standards for Efficient Cryptography Group*.

[26] CJP (2015). Routing on the lightning network? http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000031.html, Accessed on 19/04/16.

[27] Clark, J. and Essex, A. (2012). CommitCoin: Carbon Dating Commitments with Bitcoin. In *Financial Cryptography and Data Security*, pages 390–398. Springer.

[28] Coinbase (2015). How do I do a customer refund with the API? https://support.coinbase.com/customer/portal/articles/ 1521752-how-do-i-do-a-customer-refund-with-the-api-, Accessed on 15/05/15.

[29] Cramer, R., Damgård, I., and Schoenmakers, B. (1994). Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer.

[30] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., and Gün, E. (2016a). On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*.

[31] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E. G., Song, D., and Wattenhofer, R. (2016b). On Scaling Decentralized Blockchains. In *3rd Workshop on Bitcoin and Blockchain Research*.

[32] Decker, C. and Wattenhofer, R. (2015a). A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In Pelc, A. and Schwarzmann, A. A., editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 9212 of *Lecture Notes in Computer Science*, pages 3–18. Springer International Publishing.

[33] Decker, C. and Wattenhofer, R. (2015b). A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18. Springer.

[34] Ekblaw, A., Azaria, A., Halamka, J. D., and Lippman, A. (2016). A case study for blockchain in healthcare:"medrec" prototype for electronic health records and medical research data. http://dci.mit.edu/assets/papers/eckblaw.pdf, Accessed on 26/10/2016.

[35] eth (2016). How do Ethereum mining nodes maintain a time consistent with the network? *Ethereum Wiki*. https://github.com/ethereum/wiki/wiki/Light-client-protocol, Accessed on 6/2/2017.

[36] Ethereum (2016a). Light client protocol. *Ethereum Wiki*. https://github.com/ethereum/wiki/wiki/Light-client-protocol, Accessed on 01/06/16.

[37] Ethereum (2016b). The mix ethereum dapp development tool. *GitHub*. https://github.com/ethereum/mix, Accessed on 10/10/16.

[38] Ethereum (2017). Code to create a contract address in Ethereum. https://github.com/ethereum/pyethereum/blob/782842758e219e40739531a5e56fff6e63ca567b/ethereum/utils.py#L62, Accessed on 10/01/17.

[39] Eyal, I., Gencer, A. E., Sirer, E. G., and van Renesse, R. (2016). Bitcoin-NG: A Scalable Blockchain Protocol. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, March 16-18, 2016, Santa Clara, CA, USA*.

[40] Eyal, I. and Sirer, E. G. (2014). Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer.

[41] Felfoldi, Z. (2017). Introduction of the Light Client for DApp developers. https://blog.ethereum.org/2017/01/07/introduction-light-client-dapp-developers/, Accessed on 23/01/17.

[42] Fiat, A. and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko, A. M., editor, *Crypto'86*, volume 263 of *LNCS*, pages 186–194. Springer.

[43] Fincen (2015). Request for Administrative Ruling on the Application of FinCEN's Regulations to a Virtual Currency Payment System. http://www.fincen.gov/news_room/rp/rulings/pdf/FIN-2014-R012.pdf, Accessed on 07/09/15.

[44] G. Dagher and B. Bunz and J. Bonneau and J. Clarke and D. Boneah (2015). Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges. In *The 22nd ACM Conference on Computer and Communications Security*.

[45] Garzik, J. (2015). BIP 100: Making Decentralized Economic Policy. http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf, Accessed on 19/04/16.

[46] Grassmuck, V. (1997). Money on the Internet Strong Privacy Protection vs. Data Trail. http://waste.informatik.hu-berlin.de/grassmuck/Texts/ecash.e.html, Accessed on 20/12/16.

[47] Greenspan, G. (2016). Why Many Smart Contract Use Cases Are Simply Impossible. http://www.coindesk.com/three-smart-contract-misconceptions/, Accessed on 30/12/16.

[48] Groth, J. (2004). Efficient maximal privacy in boardroom voting and anonymous broadcast. In *International Conference on Financial Cryptography*, pages 90–104. Springer.

[49] Hankerson, D., Vanstone, S., and Menezes, A. (2004). *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer.

[50] Hao, F. (2010). On robust key agreement based on public key authentication. In *Financial Cryptography and Data Security*, pages 383–390. Springer.

[51] Hao, F., Ryan, P. Y., and Zielinski, P. (2010). Anonymous voting by two-round public discussion. *IET Information Security*, 4(2):62–67.

[52] Hearn, M. (2011). bitcoinj. https://bitcoinj.github.io/, Accessed on 19/04/16.

[53] Hearn, M. (2014). Re: [Bitcoin-development] BIP 70 refund field. *Bitcoin-Development*. http://sourceforge.net/p/bitcoin/mailman/message/ 32157661/, Accessed on 01/02/15.

[54] Heilman, E., Kendler, A., Zohar, A., and Goldberg, S. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144.

[55] Hertig, A. (2015). The First Bitcoin Voting Machine Is On Its Way. *Motherboard Vice*. http://motherboard.vice.com/read/the-first-bitcoin-voting-machine-is-on-its-way, Accessed on 01/09/16.

[56] Hertig, A. (2017). Where's Casper? Inside Ethereum's Race to Reinvent its Blockchain. http://www.coindesk.com/ethereum-casper-proof-stake-rewrite-rules-blockchain/, Accessed on 22/01/17.

[57] Higgins, S. (2016a). Abu Dhabi Stock Exchange Launches Blockchain Voting. *CoinDesk*. http://www.coindesk.com/abu-dhabi-exchange-blockchain-voting/, Accessed on 12/10/16.

[58] Higgins, S. (2016b). IBM Invests $200 Million in Blockchain-Powered IoT. *CoinDesk*. http://www.coindesk.com/ibm-blockchain-iot-office/, Accessed on 24/12/16.

[59] Higgins, S. (2016c). Russia's Central Securities Depository Tests Blockchain Assets Exchange. http://www.coindesk.com/russia-central-securities-depository-blockchain-assets/, Accessed on 10/01/17.

[60] Hileman, G. (2016). State of blockchain q1 2016: Blockchain funding overtakes bitcoin. http://www.coindesk.com/state-of-blockchain-q1-2016/, Accessed on 3/01/17.

[61] Horrocks, R. (2015). Error while compiling: Stack too deep. *Ethereum Stack Exchange*. http://ethereum.stackexchange.com/a/6065, Accessed on 01/09/16.

[62] International Association for Cryptologic Research (2016). About the helios system. http://www.iacr.org/elections/eVoting/about-helios.html, Accessed on 09/10/2016.

[63] ISO/EIEC (2008). ISO/IEC 14888: Information technology – Security techniques – Digital signatures with appendix. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=44226, Accessed on 01/04/14.

[64] Johnson, D., Menezes, A., and Vanstone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63.

[65] Khader, D., Smyth, B., Ryan, P. Y., and Hao, F. (2012). A fair and robust voting system by broadcast. In *5th International Conference on Electronic Voting*, volume 205, pages 285–299. Gesellschaft für Informatik.

[66] Kiayias, A. and Yung, M. (2002). Self-tallying elections and perfect ballot secrecy. In *International Workshop on Public Key Cryptography*, pages 141–158. Springer.

[67] Kumaresan, R. and Bentov, I. (2014). How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 30–41. ACM.

[68] Lemieux, V. L. and Lomas, E. (2016). Trusting records: Is blockchain technology the answer? *Records Management Journal*, 26(2).

[69] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. (2015). Inclusive block chain protocols. In Böhme, R. and Okamoto, T., editors, *Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 528–547. Springer Berlin Heidelberg.

[70] Lo, S. and Wang, J. (2014). *Bitcoin as Money?* Current Policy and Perspectives. http://www.bostonfed.org/economic/current-policy-perspectives/2014/cpp1404.pdf, Accessed on 01/10/14.

[71] Lombrozo, E., Lau, J., and Wuille, P. (2015). BIP 141: Segregated Witness. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, Accessed on 19/04/16.

[72] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM.

[73] Malone-Lee, J. and Smart, N. P. (2003). Modifications of ECDSA. In Nyberg, K. and Heys, H., editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg.

[74] Maurer, B., Nelms, T., and Swartz, L. (2013). "When perhaps the real problem is money itself!": the practical materiality of Bitcoin. *Social Semiotics*, 23(2):261–277.

[75] Maxwell, G. (2013). CoinJoin: Bitcoin privacy for the real world. https://bitcointalk.org/index.php?topic=279249, Accessed on 20/05/15.

[76] McCorry, P., Möser, M., Shahandasti, S. F., and Hao, F. (2016a). Towards bitcoin payment networks. In *Australasian Conference on Information Security and Privacy*, pages 57–76. Springer.

[77] McCorry, P., Shahandashti, S. F., Clarke, D., and Hao, F. (2015). Authenticated key exchange over bitcoin. In *Security Standardisation Research*, pages 3–20. Springer.

[78] McCorry, P., Shahandashti, S. F., and Hao, F. (2016b). Refund attacks on bitcoin's payment protocol. *Financial Cryptography and Data Security*.

[79] McCorry, P., Shahandashti, S. F., and Hao, F. (2017). A smart contract for boardroom voting with maximum voter privacy.

[80] McDonald, L. (2009). Crash of a titan: The inside story of the fall of Lehman Brothers. http://www.independent.co.uk/news/business/analysis-and-features/crash-of-a-titan-the-inside-story-of-the-fall-of-lehman-brothers-1782714.html, Accessed on 3/01/17.

[81] Meiklejohn, S. and Orlandi, C. (2015). Privacy-enhancing overlays in bitcoin. In *Bitcoin Workshop at Financial Cryptography and Data Security*. Springer.

[82] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2013). A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM.

[83] Merrill, N. (2015). The Calyx Institute: Privacy by design for everyone. https://www.calyxinstitute.org/support-us/donate-via-bitcoin, Accessed on 06/01/15.

[84] Miers, I., Garman, C., Green, M., and Rubin, A. (2013). Zerocoin: Anonymous Distributed E-cash from Bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE.

[85] Miller, V. (1986). Use of Elliptic Curves in Cryptography. In *Advances in Cryptology—CRYPTO'85 Proceedings*, pages 417–426. Springer.

[86] Monero (2015). Monero is a secure, private, untraceable currency. It is open-source and freely available to all. https://getmonero.org/home, Accessed on 08/12/15.

[87] Morgenson, G. (2011). Secrets of the Bailout, Now Told. http://www.nytimes.com/2011/12/04/business/secrets-of-the-bailout-now-revealed.html, Accessed on 3/01/17.

[88] Möser, M. and Böhme, R. (2015). Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *International Conference on Financial Cryptography and Data Security*, pages 19–33. Springer.

[89] Mozilla (2015). Help protect the open Web. https://sendto.mozilla.org/page/content/give-bitcoin/, Accessed on 06/01/15.

[90] Nakamoto, S. (2008a). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Accessed on 01/01/15.

[91] Nakamoto, S. (2008b). Bitcoin P2P e-cash paper. http://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html, Accessed on 20/12/16.

[92] Nakamoto, S. (2009). Bitcoin v0.1 released. http://www.mail-archive.com/cryptography@metzdowd.com/msg10142.html, Accessed on 20/12/16.

[93] Nakamoto, S. (2010). Transactions and Scripts: DUP HASH160 ... EQUALVERIFY CHECKSIG. https://bitcointalk.org/index.php?topic=195.msg1611, Accessed on 3/01/17.

[94] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies*. Princeton University Press.

[95] Nayak, K., Kumar, S., Miller, A., and Shi, E. (2016). 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. IEEE.

[96] Odlyzko, A. (2003). The case against micropayments. In *International Conference on Financial Cryptography*, pages 77–83. Springer.

[97] Pair, S. (2016). A Simple, Adaptive Block Size Limit. https://medium.com/@spair/a-simple-adaptive-block-size-limit-748f7cbcfb75, Accessed on 19/04/16.

[98] Peters, G. W. and Panayi, E. (2015). Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. *Available at SSRN 2692487*.

[99] Poon, J. (2015). Payment Re-routing. http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000018.html, Accessed on 19/04/16.

[100] Poon, J. and Dryja, T. (2016). The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, Accessed on 19/04/16.

[101] Reid, F. and Harrigan, M. (2011). An analysis of anonymity in the bitcoin system. In *Privacy, security, risk and trust (passat), 2011 IEEE Third International Conference on and 2011 IEEE third international conference on social computing* , pages 1318–1326.

[102] Reitwiessner, C. (2016). Smart contract security. https://blog.ethereum.org/2016/06/10/smart-contract-security/ Accessed on 01/09/16.

[103] Rizzo, P. (2015). Expedia Exec Says Bitcoin Spending Has Exceeded Estimates. http://www.coindesk.com/expedia-exec-bitcoin-payments-have-exceeded-estimates/, Accessed on 06/01/15.

[104] Ron, D. and Shamir, A. (2013). Quantitative Analysis of the Full Bitcoin Transaction Graph. In *Financial Cryptography and Data Security*, pages 6–24. Springer.

[105] Ruffing, T., Moreno-Sanchez, P., and Kate, A. (2014). Coinshuffle: Practical decentralized coin mixing for bitcoin. In *Computer Security-ESORICS 2014*, pages 345–364. Springer.

[106] Russell, R. (2015a). Loop attack with onion routing.. http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-August/000153.html, Accessed on 19/04/16.

[107] Russell, R. (2015b). Reaching The Ground With Lightning (draft 0.2). https://github.com/ElementsProject/lightning/blob/master/doc/deployable-lightning.pdf, Accessed on 19/04/16.

[108] Russell, R. (2015c). Routing on the lightning network? http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000019.html, Accessed on 19/04/16.

[109] Sapirshtein, A., Sompolinsky, Y., and Zohar, A. (2016). Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*. Springer.

[110] Schildbach, A. (2014). Re: [Bitcoin-development] BIP 70 refund field. *Bitcoin-Development*. http://sourceforge.net/p/bitcoin/mailman/message/ 32157651/, Acccessed on 01/02/15.

[111] Schnorr, C.-P. (1991). Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174.

[112] Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer.

[113] Sparkes, M. (2015). Britons can now buy Dell computers with Bitcoin. http://www.telegraph.co.uk/technology/news/11425250/ Britons-can-now-buy-Dell-computers-with-Bitcoin.html Accessed on 26/02/15.

[114] Spilman, J. (2013). Anti DoS for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html, Accessed on 19/04/16.

[115] State, N. Y. (2015). Chapter i regulations of the superintendent of financial services, part 200. virtual currencies. *Department of finance services*.

[116] Stewart, H. (2013). Eurozone bailouts: which countries remain? https://www.theguardian.com/business/2013/dec/13/eurozone-bailouts-greece-portugal-cyprus-spain, Accessed on 3/01/17.

[117] Szilagyi, P. (2016). Whoa... Geth 1.5. https://blog.ethereum.org/2016/11/17/whoa-geth-1-5/, Accessed on 23/01/17.

[118] Tapscott, D. and Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin.

[119] Todd, P. (2014). OP_CHECKLOCKTIMEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, Accessed on 01/01/16.

[120] Tor (2015). Make A Donation. https://www.torproject.org/donate/donate.html.en, Accessed on 06/01/15.

[121] Treanor, J. (2015). RBS sale: Fred Goodwin, the £45bn bailout and years of losses. https://www.theguardian.com/business/2015/aug/03/rbs-sale-fred-goodwin-bailout-years-of-losses, Accessed on 3/01/17.

[122] Trillo, M. (2014). Put to the Stress Test. http://visatechmatters.tumblr.com/post/96025603185/put-to-the-stress-test-visanet-gets-pushed-to-the, Accessed on 19/04/2016.

[123] Tur, M. (2015). Can BitPay refund my order? https://support.bitpay.com/hc/en-us/articles/203411523-Can-BitPay-refund-my-order-, Accessed on 07/04/2015.

[124] Valenta, L. and Rowan, B. (2015). Blindcoin: Blinded, accountable mixes for bitcoin. In *Bitcoin Workshop at Financial Cryptography and Data Security*.

[125] Vaudenay, S. (2002). The Security of DSA and ECDSA. In *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 309–323. Springer Berlin Heidelberg.

[126] Walport, M. (2016). Distributed ledger technology: Beyond blockchain. uk government office for science. Technical report, Tech. Rep.

[127] Wire, B. (2016). Now You Can Vote Online with a Selfie. *Business Wire*. http://www.businesswire.com/news/home/20161017005354/en/Vote-Online-Selfie, Accessed on 25/10/16.

[128] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*.

[129] Wuille, P. (2012). BIP 32: Hierarchical Deterministic Wallets. *Bitcoin Foundation*. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, Accessed on 10/05/15.

[130] Wuille, P. (2015a). BIP 66: Strict DER signatures. https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki, Accessed on 19/04/16.

[131] Wuille, P. (2015b). Switch to libsecp256k1-based ECDSA validation. *Bitcoin Github Repository*. https://github.com/bitcoin/bitcoin/pull/6954, Accessed on 31/12/15.

[132] Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communications Security*.