Bismuth: A Blockchain-Based Program For Verifying Responsible Data Usage

by

Keeley Donovan Erhardt

S.B., C.S. M.I.T. (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

© Keeley Donovan Erhardt, MMXVII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author . . . .
Department of Electrical Engineering and Computer Science
August 16, 2017

Certified by . . . .
Alex "Sandy" Pentland
Toshiba Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by . . . .
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Bismuth: A Blockchain-Based Program For Verifying Responsible Data Usage

by

Keeley Donovan Erhardt

Submitted to the Department of Electrical Engineering and Computer Science
on August 16, 2017, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science

Abstract

The amount of digital information generated, collected and stored is growing at a staggering rate. Data-driven insights are increasingly relied upon to make decisions, directly impacting individuals. The burgeoning importance of data in shaping the world around us requires a shift in the current data ownership, exchange and usage paradigm. Responsible data use should be verifiably free from leaking sensitive information, discriminatory usage, illegal applications, and other misuse. Additionally, a standard of correctness for computations executed against datasets should be enforced. Enlisting trusted parties to vet the code being executed against sensitive data can reduce the prevalence of irresponsible or malevolent data usage. Trusted parties can attest to attributes of the code—for example, that the code is privacy-preserving, or that it is legal to execute against data collected from users in a certain country, or that a computation reliably and correctly computes an answer as advertised—to ensure that individuals' personal information is used appropriately. This thesis presents an illustration of a design to structure, vet and verify the code that is executed against sensitive data, along with a proposal for using blockchain-based smart contracts to audit and enforce proper usage of vetted code to promote a paradigm of "safe" question-and-answer exchange. Finally, this thesis demonstrates Bismuth, a blockchain-based program built to implement the ideas presented in this work and to assist in a transition towards more thoughtful and responsible data usage.

Thesis Supervisor: Alex "Sandy" Pentland
Title: Toshiba Professor of Media Arts and Sciences

Acknowledgments

This work is generously supported by the Ripple Fellowship. Any opinions, findings, conclusions and recommendations expressed in this thesis are those of the author and do not necessarily reflect the views of Ripple.

Professor Alex “Sandy” Pentland, for serving as both a supervisor and a role model.

My family, especially my parents, for their unwavering support throughout my life.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

The amount of digital information generated, collected and stored is growing exponentially—by 2020 there is expected to be more than 16 zettabytes (16 trillion GB) of useful data [Turner et al., 2014]. Digital devices such as smart phones, credit cards, satellites, wearable devices, and connected "things" generate massive quantities of data. People further produce data through online activities such as posting on social media, exchanging email, and browsing the Internet. This data plays an increasingly important role in our day-to-day life, informing individual, business and policy decisions and driving human action. The economic and societal benefits accrued from data-driven decision making are immense; big and open data is predicted to increase European GDP by 1.9 percent by 2020—equivalent to a full year of economic growth in the European Union [Buchholtz et al., 2014].

While data is undeniably useful for understanding and shaping the world around us, an open question remains—who owns this data? Currently, the majority of data is aggregated and stored by private companies and governments, rather than by individuals. Companies are increasingly adopting a business model that exchanges free products and services for user data.[1] Google offers an impressive array of "free" services, generating revenue from the personal information it collects from its users.[2] Data aggregation at this scale can be beneficially exploited: Google Flu Trends, a free daily flu forecast for the United States launched by Google.org in 2008, famously attempted to use search data to detect the onset of seasonal flu epidemics. Google researchers demonstrated that by combining the billions of queries stored by the company with flu tracking information published by the US Centers for Disease Control

[1]This phenomenon is often referenced by the popular aphorism, "if you're not paying for something, you're not the customer; you're the product being sold," first mentioned in a comment by user blue_beetle within the famed MetaFilter community in August 2010.

[2]Google provides an exhaustive overview of the data it collects on users of Google services (currently available at https://privacy.google.com/your-data.html). The list includes (among other datapoints): (1) "things you do"—information you search for, websites you visit, and your location; (2) "things you create'—emails you send and receive on Gmail, contacts you add, and calendar events; and (3) "things that make you *you*"—name, email address and password, and birthday.

and Prevention (CDC), they could produce accurate estimates of flu prevalence two weeks earlier than possible using only the CDC data [Butler, 2008].[3]

Google Flu Trends and related areas of research have the potential to deliver substantial benefits to society. However, realizing these benefits is not contingent upon acquiescing to a paradigm in which private companies maintain complete control over individuals' information. As an increasing amount of data is generated, questions of privacy and data ownership must be confronted.

The "New Deal on Data": a proposal for personal data ownership. Presented at the World Economic Forum (WEF) in 2009, the "New Deal on Data" proposes a novel framework for open information markets predicated on the idea that people should own their data [Pentland, 2009]. Drawing from the three tenets of ownership laid out in Old English Common Law—possession, use, and disposal—the New Deal on Data mandates that individuals have the right to, (1) *possess* their data, (2) fully control the *use* of their data, and (3) *dispose* or *distribute* their data. When people retain possession over their data, companies perform the function of a Swiss bank, simply holding data (ideally anonymously) on behalf of their users. When people, rather than companies, act as data owners, they gain the ability to destroy or remove and transfer their data from a company's control if they are unsatisfied with how it is being used or managed.

Under an open information market in which individuals own their data, data can be more fairly and transparently utilized and monetized. Programs such as Google Flu Trends would still be possible. Google could request specific and explicit permission from its users to collect, aggregate and analyze all search queries involving influenza related terms. By demonstrating the value of such a program, Google could incentivize voluntary participation by its users. Obtaining fine-grained consent reduces the likelihood of harmful and exploitative usage of personal data.

Unfortunately, even under the New Deal on Data, data be misused. Data might be used in a discriminatory manner, or used illegally. Additionally, sensitive information

[3]For both the 2012-2013 and 2013-2014 seasons, Google Flu Trends overestimated the prevalence of influenza by more than 50 percent. The program was subsequently dismantled, becoming a poster child for the fallibility of big data.

might be inadvertently leaked. A company could request permission from each of its users to mine and aggregate search data related to sexually transmitted diseases, promising to anonymize the data so that the individuals' search data cannot be tied to them. However, most anonymization techniques have been shown to be ineffective [Ohm, 2009]; released "anonymized" datasets often result in the leakage of sensitive data [Wjst, 2010].

Open Algorithms (OPAL): sending the code to the data. OPAL is a new paradigm designed to promote responsible data use by separating the code used to generate insights from the data itself.[4] An important feature of the design is that raw data is kept under the protection of the individual, organization, company, or government responsible for the data and never publicly released. Instead of a public release, the design promotes "sending the code to the data [Greenwood et al., 2014]." Code in this context could refer to queries, computations, or other algorithms; the complexity of the code can range from a simple SQL or MongoDB query to MATLAB code to an intricate machine learning algorithm. "Sending the code to the data" refers to an entity sending a query, computation, or other algorithm to the party in custody of the data to execute on his/her behalf, circumventing the need to release raw data.

Contribution of this work. Expanding on the OPAL paradigm, this thesis presents three primary contributions:

(1) An illustration of a design that supports signed attestations from trusted parties ascribing attributes to code executed against sensitive data.

(2) A proposed architecture to implement a two-way peg binding the action of "sending the code to data" to blockchain transactions.

(3) A prototype implementation of the aforementioned ideas via a program called Bismuth.

To validate Bismuth, the program has been incorporated into the open-source OPAL server and tested within this architecture [CxSci, 2017].

[4]The Open Algorithms (OPAL) project is a multi-partner effort led by the Data-Pop Alliance, Imperial College London, Orange, the World Economic Forum, and the MIT Media Lab.

1.1 Blockchain

In 2008, a pseudonymous person or persons named Satoshi Nakamoto released a new protocol for "A Peer-to-Peer Electronic Cash System" using a cryptocurrency called bitcoin [Nakamoto, 2008]. Bitcoin are represented as transactions recorded on a globally distributed digital ledger, known as a blockchain. A blockchain is established by a set of rules—in the form of distributed computations—that ensure the integrity of arbitrary data without relying on a trusted third party. These rules aggregate bitcoin transactions into 'blocks', which are then added to a 'chain' of existing blocks using cryptographic signatures. Blockchains can store token balances, as well as more complex information, by securing a set of private keys. The technology exhibits five critical characteristics—blockchains are:

1. **Distributed:** geographically disparate computers run software to host the ledger.
2. **Public:** anyone can view the state of the network at any time and all transactions.
3. **Inclusive:** there are no restrictions on who can operate a node in the network.
4. **Immutable:** all transactions that occur are verified, cleared and stored in a time-stamped block that references the previous block, thereby creating a chain.
5. **Historical:** the ledger retains unalterable information on every previous state.

Modern blockchain designs are capable of storing state and establishing permissions to modify and update state through self-administering and self-executing scripts run in a distributed virtual machine. These scripts are known as smart contracts, and have the ability to define and regulate rules to govern users' interactions with the underlying blockchain. Further details are provided in Sections 2 and 3.

1.2 Bismuth

This thesis details the Bismuth program, a concrete implementation of a two-way peg mechanism that associates the act of "sending the code to the data" to immutable, publicly audit-able records on a blockchain. The ideas presented in this work are general enough to guide implementations of two-way pegs associating actions in a variety of domains with blockchain transactions. However, in the interest of illustrating the underlying concepts as explicitly as possible through working code and concrete examples, this thesis focuses on integrating Bismuth into the OPAL architecture.

It should be noted that in the blockchain ecosystem, the connotation associated with the terminology "two-way peg" (2WP) differs slightly from the meaning adopted in this work. In 2014, a group of researchers, loosely affiliated with Blockstream, proposed *pegged sidechains*, a technology to enable bitcoins and other ledger assets to be transferred between multiple blockchains.[5] The authors defined a two-way peg as "the mechanism by which coins are transferred between sidechains and back at a fixed or otherwise deterministic exchange rate [Back et al., 2014]." Rather than associating interactions occurring on different blockchains (a main chain and its sidechain), this work aims to associate on- and off-chain actions. A two-way peg in this context is a mechanism by which off-chain, or physical world, interactions are associated with blockchain transactions in a verifiable manner.

1.3 Paper organization

The remainder of this paper is organized as follows. Section 2 provides an introduction to distributed ledger technology and a summary of a number of the more popular distributed ledger implementations. An overview of Ethereum, the blockchain implementation selected for use in the initial Bismuth prototype, is presented in Section 3. Section 4 outlines a design for creating and verifying signed attributes associated with executable code, and a mechanism for developing a two-way peg. Section 5 details the integration of Bismuth into the OPAL architecture, and Section 6 describes optimizations and future work. Section 7 concludes with a discussion of the contributions of this thesis and future directions.

[5]As defined by Back et al. in *Enabling Blockchain Innovations with Pegged Sidechains*, a sidechain is a blockchain that validates data from other blockchains. A pegged sidechain is a sidechain whose assets can be imported from and returned to other chains; that is, a sidechain that supports two-way pegged assets.

2 Distributed ledger technology

A distributed ledger is a consensus of replicated, shared, and synchronized data geographically spread across multiple sites, countries, or institutions [Walport, 2016]. Ledgers have existed for millennia, and are used to record many things, most commonly assets such as money and property. Distributed ledgers extend the abilities of traditional ledgers by enabling real-time and secure data sharing and updating between physically disparate parties.

A core challenge in maintaining distributed ledgers is achieving consensus—all network participants must be in agreement about the current state of the ledger.[1] This requires updates to the ledger to be received in the proper order and with minimal delay by all participants. Achieving decentralization further requires that all participants have the ability to both receive and make updates to the ledger that are recognized by the entirety of the network.

2.1 Cryptocurrencies

Nakamoto's white paper on "Bitcoin: A Peer-to-Peer Electronic Cash System", presented the first practical design for a Byzantine fault tolerant, decentralized and trust-less distributed ledger.[2] Bitcoin launched a new class of cryptocurrencies that built on Nakamoto's innovation to support their own implementations of decentralized transfer of monetary value.

[1]Distributed consensus can be most easily explained by the Two Generals' Problem. Two generals are attempting to coordinate an attack on a city located in a valley between two hills. The generals are located on the two hills above the city and must decide the exact time to attack the city by sending a messenger through the valley and up to the other generals hill. There is a non-zero probability that the messenger they send is captured and killed in the city below. Due to the possibility of a generals acknowledgement being lost, achieving consensus with 100 percent confidence requires an infinite series of messengers.

[2]Byzantine fault tolerance (BFT) is the characteristic of a system that tolerates the class of failures known as the Byzantine Generals' Problem, which is a generalized version of the Two Generals' Problem.

2.1.1 Bitcoin

Bitcoin is an entirely decentralized digital currency, free from any centralized issuer or controller. It is the first cryptocurrency to achieve widespread adoption, and Bitcoin and its derivatives have since experienced massive growth in popularity, research, and market share. However, Bitcoin's greatest contribution is arguably not the currency itself, but rather the underlying blockchain technology as a mechanism for trust-less and distributed consensus.

A blockchain is a type of distributed ledger that is composed of unchangeable, digitally recorded data assembled into 'blocks'. Each block in the chain contains data (e.g. Bitcoin transactions) and is cryptographically hashed. The hash of the data in each block references the previous block, resulting in a linear 'chain' of blocks, referred to as a 'blockchain'.[3] Changing the data stored in any one block mutates the hash of all subsequent blocks, ensuring that any tampering of information is immediately detected by the participants in the network.

A blockchain requires distributed consensus to standardize on the valid ordering of transactions. Bitcoin introduced a novel, decentralized mechanism for distributed consensus known as *proof-of-work*. Proof-of-work (PoW) relies on participants in the network performing computational work to validate each block of transactions. The participants validating blocks of transactions to create new blocks are called miners. Miners receive the cumulative amount of fees from transactions included in the blocks they mine, as well as a block reward in the form of newly minted bitcoin.[4]

In order to validate a block of transactions and claim the block reward, miners compete to find a nonce value to include in the block header that results in the double-SHA256 hash of the block, treated as a 256-bit number, to be less than a

[3]A blockchain is only one example of a data structure which can be used to achieve distributed consensus for a shared ledger. There are many other implementations of distributed ledger technology that use different methodologies to achieve the same consensus (e.g. Ripple, MultiChain, HyperLedger Project).

[4]Bitcoin is a deflationary currency: the code permits only 21 million bitcoin to ever be in circulation. New bitcoin are created in the system through the block reward. When Bitcoin was first created, the block reward was set at 50 bitcoin per block mined. The code specifies that the block reward is to be cut in half after every 210,000 blocks mined, falling to zero after 64 halving events. At the time of this writing, the block reward is 12.5 bitcoin.

```
target: 00000000ffff0000000000000000000000000000000000000000000000000000
hash:   000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Figure 2-1: The double-SHA256 proof-of-work hash of the Bitcoin genesis block and the corresponding target, mined 2009-01-03 18:15:05 UTC.

dynamically adjusted target (at the time of this writing, approximately 2187). The Bitcoin network automatically adjusts this target depending on the hash rate of the network (the total hashes per second the network of miners can compute) to ensure that Bitcoin blocks are mined at a limited pace—approximately one block every 10 minutes.

Proof-of-work establishes a simple and effective consensus algorithm by making block creation computationally hard. This consensus algorithm accomplishes several tasks:

- it allows for free entry into the consensus protocol because anyone with a processing unit can compete in the creation of new blocks;
- it prevents Sybil attackers from recreating the entire blockchain in their favor;[5]
- it introduces new bitcoin into the system at a steady rate;
- it distributes bitcoin among participants using a reasonably fair distribution scheme;
- it makes blocks tamper-resistant since any change to a block requires additional work to generate a new proof.

2.1.2 Altcoins

Following the advent of Bitcoin, a wide range of alternative implementations of decentralized ledgers have been developed, and novel applications for distributed ledger technology have been proposed. These alternative cryptocurrencies, known as altcoins, attempt to improve on the original Bitcoin protocol to add new features and capabilities. Some altcoins attempt to augment the anonymity guarantees offered by the Bitcoin protocol (e.g. Zerocash and Zerocoin), while others experiment with

[5]A Sybil attack refers to an attacker subverting a reputation system by forging a large number of pseudonmous identities in a peer-to-peer network to gain disproportionally large influence. The attack is named after the subject of the book Sybil, a case study of a woman diagnosed with dissociative identity disorder.

alternative consensus algorithms to replace proof-of-work (e.g. Peercoin).[6]

Altcoins have two options for building their respective consensus protocols: building a protocol on top of Bitcoin, or developing an entirely independent network. Both options require trade-offs and carry their own sets of challenges. A protocol built on top of Bitcoin does not inherit Bitcoin's simplified payment verification features.[7] Furthermore, a meta-protocol cannot compel the underlying Bitcoin blockchain to exclude transactions which are invalid in the context of the meta-protocol. To overcome this limitation, a secure protocol built on top of Bitcoin must scan all the way to the initial Bitcoin genesis block to determine if certain transactions are valid according to its rules. On the other hand, building an independent network is time-consuming and difficult to implement correctly. Immature cryptocurrencies further suffer from low market liquidity, and decreased resiliency against attacks when the number of participants in the network is small.

2.2 Smart contracts

Bitcoin was originally designed as a decentralized cryptocurrency network for transacting monetary value, and the main purpose of the Bitcoin blockchain was to provide a secure ledger for these financial transactions. More recently, use cases for non-financial applications related to decentralized trust and store of value have begun to emerge associated with the Bitcoin blockchain, and an even wider range of projects have been proposed for decentralized ledgers more broadly.

In 1994, Nick Szabo coined the term *smart contract*, referring to an automated program that would function similar to a standard computer program's if-then statements. Real-world actions or assets could satisfy pre-programmed conditions that

[6]*Proof-of-Stake* (PoS) is a commonly proposed alternative to PoW for achieving distributed consensus. Proof-of-stake was first suggested in the bitcointalk forum in 2011, and first implemented by Peercoin in 2012. In a proof-of-stake system, the creator of the next block is chosen in a deterministic fashion—the likelihood of an account being selected depends on its wealth (i.e. its stake). No block rewards are distributed; participation is incentivized solely through transactions fees.

[7]Simplified payment verification (SPV) is a method for verifying if particular transactions are included in a block without downloading the entire block. The method is used by some lightweight Bitcoin clients.

would cause a smart contract to execute the corresponding code. Szabo's theories originally went unrealized because a digitally native financial system capable of supporting programmable transactions did not exist. More than a decade later, the development of Bitcoin resulted in blockchains capable of supporting first a small set of simple operations—primarily transactions of a currency-like token—and later blockchains capable of more complex operations, defined in full-fledged programming languages. Now general-purpose blockchains can act as programmable ledgers, capable of providing computing infrastructure to support complex logic and programmable transactions.

The term "smart contract" is overloaded and used to represent a broad range of concepts. This thesis defines smart contracts to be the code that is stored, verified and executed on an immutable distributed ledger whose inputs and outputs are maintained globally consistent by a distributed consensus protocol. The distributed ledgers have Virtual Machinery (VM), similar to the virtual machines of traditional cloud computing; for example, a blockchain VM is a virtual network of decentralized computers that are coordinated through the blockchain's consensus rules. A blockchain acts as a distributed computing architecture where every network node executes and records the same transactions. Every node in the network runs the blockchain's VM and executes the same instructions. This massive parallelism makes computation on a blockchain slower and less efficient than on a traditional computer, but maintains network consensus.

Smart contracts often encapsulate logic, rules, processes, or agreements between different parties. They can include arbitrary rules for ownership, transaction formats and state transition functions on the blockchain. As a result of running on top of a blockchain, the contracts adopt several unique characteristics compared to other software programs. Smart contract programs:

- are recorded on the blockchain, so are themselves permanent and immutable;
- can control blockchain assets, i.e. transfer and store cryptocurrency;
- are executed by the blockchain, so will always run exactly as written.

The Bitcoin, Ethereum, Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger

Corda (previously known as R3 Corda), and Rootstock (RSK) smart contract platforms were considered for use in the Bismuth program.

2.2.1 Bitcoin

The original Bitcoin protocol has simple scripting capabilities for its blockchain which support a highly limited concept of smart contracts. Bitcoin funds are typically controlled by public key, however, they can also be owned by a more complex script expressed in Bitcoin's simple, stack-based programming language. In order to spend the funds, a Bitcoin transaction must provide data that satisfies the script. This functionality enables multisig validation, bounties for solving computational problems and other capabilities. However, Bitcoin's scripting language is not Turing-complete and has several restrictive limitations:

- **Value-blindness:** a Bitcoin script is unable to provide fine-grained control over the amount withdrawn.[8]
- **Blockchain-blindness:** Bitcoin transactions are blind to certain blockchain data such as the nonce and previous block hash.
- **Lack of State:** the Bitcoin protocol recognizes only the concepts of spent or unspent, with no mechanism for handling additional internal state.

2.2.2 Ethereum

Ethereum is an alternative decentralized ledger protocol—independent of Bitcoin—designed as a platform for building decentralized applications. A decentralized application consists of a set of one or more smart contracts and client-side code. Decentralized applications are not controlled by any single individual, board, or other central entity,

[8]This limititation results from Bitcoin's concept of unspent transaction outputs. A Bitcoin transaction works by consuming a collection of objects called unspent transaction outputs (UTXOs) created by one or more previous transactions, and then producing one or more new UTXOs. These new UTXOs are then available to be consumed by future transactions. If a user wants to send a transaction of X coins to a particular address, the user must either have some set of UTXOs that exactly sum to X, or include a set of UTXOs summing to a value greater than X and a second destination UTXO called a "change output" that he/she controls. Therefore, for smart contracts to handle transactions of arbitrary amounts the contracts would need to have many UTXOs of varying denominations.

and are instead powered by smart contracts. Ethereum overcomes Bitcoin's smart contract limitations by providing a highly flexible and fundamental abstraction: a blockchain with a built-in Turing-complete programming language.

Similar to Bitcoin, Ethereum executes contracts under an incentivization scheme which pays miner fees based on the computing power spent. For this, it uses Ether, its native cryptocurrency, which carries real-world value. Contracts are executed as stack-based bytecode in the Ethereum Virtual Machine (EVM). Similar to the Bitcoin blockchain, the Ethereum blockchain is a permissionless ledger—anybody is free to join the network, perform transactions and/or access the source code of deployed smart contracts.

2.2.3 Hyperledger Project

Bitcoin and Ethereum operate in a public (or permissionless) environment where participants can join or leave at will. More recently, there has been increased interest in private (or permissioned) blockchains optimized for private settings in which participants are authenticated. The most prominent example of this new development is the Linux Foundation's Hyperledger Project which, "incubates and promotes a range of business blockchain technologies, including distributed ledger frameworks, smart contract engines, client libraries, graphical interfaces, utility libraries and sample applications [Foundation, 2017]." The initiative is an open source collaborative effort that consists of over 140 members.

The Hyperledger projects focus on enterprise use cases, utilizing permissioned ledgers. Hyperledger includes five distinct frameworks: Hyperledger Sawtooth, Hyperledger Iroha, Hyperledger Fabric, Hyperledger Burrow and Hyperledger Indy.[9] In addition, the R3 consortium of financial institution donated the code for their distributed ledger framework, Corda, to the Linux Foundation's Hyperledger Project in 2017.

[9]It should be noted that although these frameworks are distributed ledger technologies, none use a blockchain. Additionally, none implement PoW as a consensus algorithm—in a permissioned ledger, unlike in a public network, the identities of participants are known so there are more efficient mechanisms available to achieve Byzantine fault tolerance.

2.2.4 Rootstock (RSK)

Rootstock (RSK) is a smart-contract platform that incorporates a Turing-complete virtual machine, the Rootstock Virtual Machine (RVM), into Bitcoin.[10] RSK greatly expands upon the smart contract capabilities built into the original Bitcoin protocol. RSK is an evolution of QixCoin—a Turing-complete cryptocurrency created in 2013 by the same development team. RSK functions as a Bitcoin sidechain;[11] bitcoin are transferred into the Rootstock blockchain and become "Rootcoins" (RTC), which are equivalent to BTC and can be transferred back at any time. There is no currency issuance in Rootcoin, as all RTC are created from BTC from the Bitcoin blockchain. The RSK sidechain uses RTC to pay miners for transaction and contract processing.

2.3 Comparison of smart contract platforms

Bitcoin and its sidechains (such as RSK), Ethereum, and the various permissioned distributed ledger implementations that fall under the Hyperledger Project, are only a subset of the platforms available for developing smart contracts. However, these platforms are the most stable and/or mature currently available, so are the ones selected for consideration in this thesis. Table 2.1 outlines a summarized comparison of the platforms, with focus on their support for smart contract development.

[10]The Rootstock Virtual Machine (RVM) is compatible with the Ethereum Virtual Machine (EVM) at the opcode level, so smart contracts built for one platform can run on both.

[11]A sidechain is an independent blockchain whose native currency is pegged to the value of another blockchain currency automatically using proofs of payment.

	Bitcoin	**Ethereum**	**Rootstock (RSK)**
Description of Platform	Payments blockchain	General purpose blockchain	General purpose sidechain
Development	Bitcoin developers	Ethereum developers	RSK Labs
Currency	BTC	Ether	BTC via two-way peg
Mining Reward	Yes	Yes	Yes
Data Model	UTXO (Unspent Transaction Outputs)	Account data	UTXO
Consensus Algorithm	PoW (Proof of Work)	PoW	PoW
Network	Public	Public or Permissioned	Public
Language	C++	GoLang, C++, Python	
SPV Clients	Yes	Yes	Yes
Confidential Transactions	No	Via contracts	Native support planned using AppeCoin protocol
Turing-complete	No	Yes	Yes
Execution Environment		EVM (Ethereum Virtual Machine)	
Smart Contract Languages	Script	Solidity, Serpent, LLL	

	Hyperledger Fabric	**Hyperledger Sawtooth**	**Hyperledger Corda**
Description of Platform	Modular blockchain platform	Modular blockchain platform	Platform for financial industry
Governance	Linux Foundation	Linux Foundation	Linux Foundation
Development	IBM	Intel	R3
Currency	None	None	None
Mining Reward	N/A	N/A	N/A
Data Model	Key-value database	Transaction families	UTXO
Consensus Algorithm	BFT (Practical Byzantine Fault Tolerance)	PoET (Proof of Elapsed Time)	BFT or RAFT
Network	Permissioned	Permissioned	Permissioned
Language		Python	
SPV Clients			
Confidential Transactions	Yes	Yes	Yes
Turing-complete	Yes	Yes	Yes
Execution Environment			modified-JVM (Java Virtual Machine)
Smart Contract Languages	GoLang, Java		Java, Kotlin

Table 2.1: Comparison of smart contract platforms.

3 Ethereum architecture

The Bismuth program is agnostic with respect to the particular distributed ledger used in the two-way peg. Bismuth achieves this independence by specifying a standard interface for the two-way peg that can be implemented for any distributed ledger with the requisite logic capabilities. As a prototype, this thesis demonstrates an implemention of the Bismuth interface for the Ethereum blockchain.

Ethereum was selected for the initial implementation because it supports the desired smart contract functionality, can be run as a permissionless network, is open-source, and is arguably the most mature smart contract platform. In comparison, the original Bitcoin protocol does not support the logic capabilities required by Bismuth, the Hyperledger implementations not suitable due to the fact that they are permissioned networks, and although the potential for a Bitcoin sidechain with smart contract capabilities is intriguing, the Rootstock sidechain was not released in time for full consideration in this work (RSK Labs released the RSK Ginger testnet in May 2017). The remainder of this section covers concepts specific to Ethereum that are useful in understanding how Bismuth integrates with the Ethereum blockchain.

3.1 Accounts

In Ethereum, the state is composed of objects called accounts, each with their own 20-byte address. There exist two types of accounts: **externally owned accounts**, controlled by private keys, and **contract accounts**, controlled by their contract code. Both types of accounts can hold a balance in ether. Externally owned accounts have no associated code, and simply send messages to other accounts by creating and signing transactions. Contract accounts have code which activates every time a message is received by the account, allowing the account to read and write to internal storage, send new messages or create contracts. An account holds four fields:

- the **nonce**, a counter used to ensure each transaction is only processed once.

- the account's current **ether balance**.
- the account's **contract code**, if present.
- the account's **storage** (empty by default).

3.1.1 Elliptic curve key pairs and Ethereum addresses

The 20-byte Ethereum account addresses are derived from an elliptic curve keypair as follows:

1. Create a random private key.
   - 64 (hex) characters / 256 bits / 32 bytes
2. Derive the public key from this private key.
   - 128 (hex) characters / 512 bits / 64 bytes
3. Derive the address from this public key.
   - Take the keccak-256 hash of the hexadecimal form of a public key; keep only the last 20 bytes.
   - 40 (hex) characters / 160 bits / 20 bytes

In theory, every possible account already exists; however, the overwhelming majority of accounts are unclaimed and store no value or data. There are $2^{256} - 2^{32} - 977$ valid private keys that can be used to derive Ethereum addresses, resulting in a near-zero chance of randomly generating a private key corresponding to an Ethereum address that is already in use.[1] Furthermore, with current technology it is computationally infeasible to derive the private key corresponding to a given public key or Ethereum address.[2].

It is important to note that the only claim to ownership that a user has over a particular account is knowledge of the private key from which the account address can be derived. In fact, each Ethereum address can be generated by any of M unique

[1] $2^{256} - 2^{32} - 977$ is referred to as `N` in the Ethereum source code, and is the order of the generator of the elliptic curve secp256k1, which is used to generate Ethereum keypairs.

[2] The elliptic curve currently used by Ethereum is considered unsafe against quantum computing; a quantum computer would be able to derive an Ethereum private key from the corresponding public key. This threat is further discussed in Section 7

private keys (i.e. there is not a one-to-one mapping from private keys to Ethereum address), so knowledge of a corresponding private key is not unique.

3.2 Transactions and messages

Messages and transactions are both digitally signed pieces of data which can transfer data and/or value to other accounts (either externally owned accounts or contract accounts) Messages and transactions are similar concepts, except a message is produced by a contract and a transaction is produced by an external actor. Transactions set in motion all action on the Ethereum blockchain. Every time a contract receives a transaction, the contract's code executes in response to the input parameters included in the transaction. Each node participating in the network executes the corresponding contract code in the Ethereum Virtual Machine (EVM) as part of their verification of new blocks.

3.2.1 Transactions

A "transaction" is the signed data package that contains a message sent form an externally owned account. A transaction includes:

- A signature identifying the sender.
- The recipient of the message.
- A `VALUE` field, specifying the amount of wei to transfer from the sender to the recipient.[3]
- An optional data field.
- A `STARTGAS` value, representing the maximum number of computational steps the transaction execution is allowed to take.
- A `GASPRICE` value, representing the fee the sender pays per computational step.

[3]Wei is a denomination of ether: 1000000000000000000 wei = 1 ether.

3.2.2 Messages

Contracts can send "messages" to other contracts, which are never serialized and exist only in the Ethereum execution environment. A message includes:

- The sender of the message (implicit).
- The recipient of the message.
- A `VALUE` field, the amount of wei to transfer with the message.
- An optional data field.
- A `STARTGAS` value, representing the maximum number of computational steps the transaction execution is allowed to take.

3.2.3 Gas and transaction costs

Both transactions and messages include values associated with "gas" (`STARTGAS` and `GASPRICE`). Gas is the fundamental unit of computation in Ethereum, and the `STARTGAS` and `GASPRICE` values support Ethereum's anti-denial-of-service model. Each transaction sent in the network must set a limit on the number of computational steps of code execution it can use. This mechanism prevents accidental or hostile infinite loops and disincentivizes inefficient code.

Users pay a gas fee roughly proportional to the computational resources consumed, including: computation, bandwidth and storage. The `STARTGAS` is the gas limit a user accepts, and `GASPRICE` is the fee a user is willing to pay per gas. The cost of a transaction is calculated as: Total cost = `gasUsed` $*$ `gasPrice`, where `gasUsed` is the total gas consumed by the transaction and `gasPrice` is the price (in wei) of one unit of gas as specified by `STARTGAS` in the transaction.

3.3 Ethereum as a state transition system

The Ethereum blockchain can be thought of as a state transition system, where state transitions consist of the direct transfer of value and information between accounts using messages and transactions. As detailed in the Ethereum white paper, the state transition function, `APPLY(S,TX) -> S'`, can be defined as follows

[Buterin et al., 2013]:

1. Validate that the transaction is well-formed and has a valid signature, and that the nonce provided matches the nonce in the sender's account.[4]
2. Calculate the transaction fee as `STARTGAS * GASPRICE`, and determine the sending address from the signature. Subtract the transaction fee from the sender's account balance and increment the sender's nonce. If there is not enough balance to spend, return an error.
3. Initialize `GAS = STARTGAS`, and subtract gas at a rate proportional to the number of bytes in the transaction.
4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.
5. If the value transfer fails due to the sender not having enough money or the code execution running out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.
6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

Each transaction must provide a valid state transition from the previously accepted state (prior to the execution of the transaction) to some new state. In this way, Ethereum smart contracts act as state transition and persistent storage systems that have access to environmental information, such as block timestamps and headers and sender balances, and that manipulate data according to pre-programmed conditions and requirements.

3.3.1 Merkle Patricia trees

The fundamental data structure used in Ethereum is the Merke Patricia tree—a fully deterministic, cryptographically authenticated data structure that can be used to

[4]An account nonce is a transaction counter in each account. The nonce helps to prevent replay attacks in which a transaction that sends 10 coins from account A to B is repeatedly replayed by B until A's account balance is completely drained.

store key-value pairs.[5] The Merkle Patricia tree is a combination of a Merkle tree and a Patricia tree; it subsumes the advantages of each to create a data structure with two critical properties: (1) Each unique set of key-value pairs maps uniquely to a root hash—this makes it impossible to dishonestly report membership of a key-value pair in a trie (unless an attacker has 2^{128} computing power), and (2) Inserts, lookups and deletes can be achieved in logarithmic time. Each block header in Ethereum holds three Merkle Patricia trees, each for a different type of object: transactions, receipts (pieces of data that reflect the effects of each transaction) and account state.

[5]Alan Reiner first proposed Merkle Patricia trees, and the data structure was subsequently implemented in the Ripple protocol.

4 Bismuth

This section details a design for verified attributes, a description of a two-peg mechanism, and the structure of the Bismuth program. First, an expansion of the previously proposed OPAL paradigm is explicitly formalized as the exchange of questions for answers through a question-and-answer system that incorporates verified attributes for proposed questions. Next, the motivation and procedure for creating and verifying attributes of code is detailed, and the interface for a two-way peg is specified. Finally, Bismuth and the related considerations associated with storing information and value on a blockchain are discussed.

4.1 "Sending the code to the data"

Imagine a scenario in which the City of Boston wants to construct a new subway line. The city hires a team of forward-thinking city planners and engineers to design the new subway line—a team that recognizes the benefits of data-driven decision making. In order to guide their decision making, the team brainstorms and compiles a list of questions. The team wants to know: *What is the distribution of income for Boston subway riders? Which current subway stops are most over-utilized? What is the distribution of time spent on the subway for riders originating from South Boston?*

Answering the questions posed by the team of city planners and engineers questions requires historical ridership data (e.g. the peak number of riders per day for each station, the average distance traveled by a rider, etc.) for subway users in the Greater Boston area. Luckily, the Massachusetts Bay Transportation Authority (MBTA) has collected and stored ridership data spanning the past five years. Given that the ridership data is already under the control of the MBTA, it is both unnecessary and risky to release the raw data to the design team for the team to analyze itself. Instead, the team members can convert their questions into code (represented as queries, computations, and algorithms) and send the code (representing the original questions) directly to the MBTA for the MBTA to execute on their behalf. The MBTA can then compute answers to pertinent project questions and return the results to the new subway-line design team.

However, this solution is overly simplistic. Despite possessing data on all subway riders in the Greater Boston area, the MBTA has a responsibility to not simply answer any question posed to it. Overly specific questions might leak sensitive data by revealing the identities of individuals in the MBTA's ridership dataset, even if an attempt is made to anonymize the data. Other questions might inadvertently reveal information that could be used in a discriminatory manner. Additionally, the MBTA might be obligated under Massachusetts state law to withhold certain information from inquiries, and would therefore be breaking the law by answering questions that elicit this protected information. For the MBTA to responsibly protect its data from misuse, and to ensure that it remains compliant with city, state and federal law, it should only answer certain questions.

If the MBTA had preliminary knowledge about the expected characteristics of an answer resulting from a given question, it would be trivial to determine whether or not the question should be answered: if the characteristics of an answer are desirable, respond to the question, otherwise, do not respond to the question. A characteristic of an answer might be that the answer is anonymous. The associated question could then be said to have a "privacy-preserving" attribute. Another question might have the attribute that it is compliant with Massachusetts state law, so answering the question is legal in the State of Massachusetts.

Instead of simply asking arbitrary questions, the subway design team could send the MBTA only questions with acceptable attributes. What is considered "acceptable" would be defined by the data owner, domain area, or particular application on a case-by-case basis. In the case of providing data insights for the construction of a new subway line, the MBTA could specify that it will only respond to questions which are (1) non-discriminatory, (2) compliant with Massachusetts state law, and (3) preserve the anonymity of any one individual's travel patterns. Any questions asked about datasets held by the MBTA will not be answered unless they have verified attestations asserting all three attributes.

To ensure the credibility of an attribute associated with a question, the attribute must be attested to by a trusted and qualified party. For example, a lawyer from the American Civil Liberties Union (ACLU) could attest to the non-discriminatory nature of a question, a privacy expert could attest to the anonymity of the returned results, and a Boston city attorney could attest to a question's legality under Massachusetts state law. The ACLU lawyer might make a mistake; the code she verifies might in actuality be discriminatory. A verified attribute does not provide a mistake-free guarantee. However, the more trusted parties that assert that a question is free from discrimination, the more the question, and its associated anti-discriminatory attribute, can be trusted. Likewise, although it is challenging to guarantee that the results of a computation will be privacy-preserving, the more experts that come to this conclusion, the more confident a data holder can be in executing the code against sensitive data.

The uncertainty associated with each of a question's attributes can be quantified using a risk function. Each attribute has a corresponding risk function. The risk function is a mathematical equation that takes a dataset's schema as input, and outputs a probability $p \in [0,\ 1]$ that the attribute holds for the given dataset. Rather than asserting that a question is definitively privacy-preserving, a privacy expert can instead attest to a risk function that computes the probability p that the question produces anonymous results. This risk function might compute p a normalized value directly proportional to the number of unique rows in the dataset that the code is executed against. In other cases, the risk function might reduce to a binary classification. For example, the risk function associated with an attribute asserting the legality of a particular question might reduce to $p = 0$ if the dataset includes data from citizens of the European Union, else $p = 1$.

This model is highly scalable. Publishing both the human-readable and computer code representation of questions, along with their verified attributes, enables other parties with similar use cases to re-use the same questions against different datasets. Publicly releasing the questions asked by the Boston subway design team allows other cities contemplating similar transportation expansions to productively benefit from the team's work. If San Francisco wants to expand the Bay Area Rapid Transit (BART) system, the city's officials can re-use many of the same questions answered to help guide the design of the Boston subway expansion. The city officials can examine the attributes associated with the pre-existing questions and determine if they want to request additional attestations of the attributes, or if certain questions require new attributes specific to their use case. The officials might decide that they are satisfied with the non-discriminatory attestation provided by the ACLU lawyer and the privacy-preserving attestation supplied by the privacy expert, but would like a University of San Francisco School of Law professor to attest to the legality of the question under California state law. The updated questions can then be sent to the stewards of the BART ridership data to generate Bay Area-specific answers.

4.2 Verified attributes

Verified attributes represent the assertions that a trusted party makes about a question. An attribute is verifiable if the party providing the attestation can be provably linked to the assertion of the attribute. A digital signature is a mathematical scheme for demonstrating the authenticity of digital messages and documents, and be used as an efficient and secure tool for linking trusted parties to verified attributes. A valid digital signature proves:

- Authentication: the document was signed by a known party
- Non-repudiation: the party cannot deny having signed the document
- Integrity: the document was not altered in transit

Consider a dataset that contains mobile phone metadata on the phone calls and text exchanges between 9 million of Orange's customers in Senegal between January 1, 2013 and December 31, 2013.[1] This data can be productively and beneficially used; for example, mobile phone metadata can be used to understand traffic patterns

[1] This example is directly motivated by the D4D-Senegal challenge, an open innovation data challenge on anonymous call patterns of Orange's mobile phone users in Senegal. The challenge aimed to use data to aid in socio-economic development and to improve the well-being of the Senegalese population [de Montjoye et al., 2014].

or the propagation of malaria across a population. However, to ensure responsible usage the computational social science and mobile phone researchers analyzing the Orange data should not necessarily receive direct access to the raw data. Even if the records are pre-processed and anonymized, the records are still vulnerable to de-anonymization. Instead, the researchers can propose the questions that they would like to investigate, and the associated queries, computations and algorithms. Trusted parties (a consortium of telecommunication companies, a research committee focused on data anonymization techniques, etc.) can then assign attributes and associated risk functions to each of the researchers' questions, e.g. this question is privacy preserving given that the dataset has more than N = 10^6 unique customers represented and the timestamps of the records involved in the question span at least three months. In this example, the attribute being verified is that the question is privacy-preserving, and the risk function (which calculates the probability that the attribute is true for any given dataset based on the dataset's schema) simplifies to:

$$p = \begin{cases} 1 & \text{if} \quad \text{N} \geq 10^6 \text{ AND endTimeInDays - startTimeInDays} \geq 90 \text{ days} \\ 0 & \text{otherwise} \end{cases}$$

JSON Web Documents (JWD). A JWD holds optionally signed and/or encrypted JSON data for storage, retrieval, transmission and display. JWDs provide an ideal container for storing questions and their associated attributes: multiple signatures can be added to the same payload, tampering with the data stored within a JWD invalidates the signatures, and a JWD can specify an expiration time after which it is invalidated. The most common JWD signing algorithms are HMAC + SHA256, RSASSA-PKCS1-v1_5 + SHA256, and ECDSA + P-256 + SHA256.[2] RSA and ECDSA are examples of asymmetric cryptography, also known as public key cryptography, which uses public and private keys to encrypt and decrypt data. An asymmetric key scheme enables third parties to validate information signed with a private key using the associated public key.

[2]The specifications for JSON Web Tokens (JWTs), a related URL-safe means of representing claims between parties, defines additional algorithms for signing. RFC 7518: available at https://tools.ietf.org/html/rfc7518section-3.

```
{
 "payload": <payload contents>,
 "signatures": [
  {
   "protected": <integrity-protected header 1 contents>,
   "header": <non-integrity-protected header 1 contents>,
   "signature": "<signature 1 contents>"
  },
  ...
  {
   "protected": <integrity-protected header N contents>,
   "header": <non-integrity-protected header N contents>,
   "signature": "<signature N contents>"
  }
 ]
}
```

Listing 4.1: Sample JSON Web Document (JWD).

Attribute attestations are located in the protected header. This design allows multiple trusted parties to each sign the same payload, as the payload includes only information inherent to the question itself (i.e. the question's executable code, title, description, asking price, etc.) and to add only the attributes that they individually attest to with their respective signatures. Section 5 provides details about the JWD payload properties included in Listing 4.2.

```
{
 "payload": {
  "qid": "fa280399-7ef8-47d8-908c-c397af690e46",
  "language": "Python",
  "code": "import math... class DecisionTree... return score...",
  "version": "0.0.1",
  "title": "Credit Worthiness Predictor using a Decision Tree
     Classifier",
  "description": "Returns the credit worthiness of a loan applicant.",
  "type": "classifier",
  "ask": "25.10",
  "cur": "USD",
 },
 "signatures": [
  {
    "protected": {
      "Complies with the Equal Credit Opportunity Act of 1974 (ECOA)":
        {
          "function": $\prod_{t \in T} i_t$, T=[race, color, religion, national origin,
              sex, marital status, age, receives public assistance],
              $i_t = \begin{cases} 1 & \text{if} \quad \text{if no dataset field reveals trait } t \\ 0 & \text{otherwise} \end{cases}$
          "prose": "MUST NOT have any fields that indicate race, color,
              religion, national origin, sex, marital status, age, or if
              the individual receives public assistance."
        }
     }
    "header": <non-integrity-protected header 1 contents>,
    "signature": "<signature 1 contents>"
  }
 ]
}
```

Listing 4.2: Credit worthiness question with verified attributes.

4.3 Two-way peg

By definition, a question-and-answer data exchange is a data-driven relationship. Employing smart contracts as the primary mechanism for administering the data-driven interactions between parties provides cryptographically enforceable agreements governing the exchange of questions, answers, and payments. Working under the assumption that the schemas for all available datasets and all questions' attributes and associated risk functions are publicly accessible, recording question and answer interactions on a blockchain allows parties outside of the exchange to observe—

- which questions are being asked,
- the attributes of these questions, and
- the likelihood that a particular attribute holds in any given instance

—all calculated using the schema of the dataset specified in the question as input to the attribute's risk function.

In an effort to build a blockchain-agnostic system, the exchange of questions and answers occurs primarily outside the context of a blockchain. This design enables a standardized protocol for question-answer data exchange that is independent of the pegged blockchain. Bismuth supports using different blockchains interchangeably by specifying a standard interface for the two-way peg. The interface has no knowledge of how any particular blockchain implements transactions or data storage. Instead, Bismuth specifies three generalized mechanisms for pegging question-answer interactions to blockchain transactions:

- use the same keys for signing data on- and off-chain;
- mirror on- and off-chain state transitions;
- process payments for questions through transfer of value on the blockchain.

4.3.1 Elliptic curve digital signatures

Ethereum secures its network using elliptic curve cryptography, in particular, the Elliptic Curve Digital Signature Algorithm (ECDSA) with secp256k1 (the parameters of the ECDSA curve).[3]. When a participant in the Ethereum network sends a transaction, the transaction must be signed with the sender's private key. Signing a transaction with a private key ties the transaction to the private key's corresponding Ethereum address, i.e. the account of the sender.[4] Knowing only the public key associated with an Ethereum account does not enable another user to send a transaction

[3]Bitcoin also uses secp256k1; it is defined in Standards for Efficient Cryptography (SEC) (Certicom Research, http://www.secg.org/sec2-v2.pdf)

[4]A side effect of signing a transaction (in the Ethereum Frontier release) is the revelation of the public key associated with the originator of the transaction.

from that account. This is important because transactions cost gas (drawn from the sender's account balance) and can transfer ether out of the sender's account. A similar key-based cryptographic protocol exists for virtually all smart contract platforms, albeit the specific curve employed can vary, as can the signing algorithm used.

Signing both JWDs (off-chain) and blockchain transactions (on-chain) using the same keys provably associates the on-chain and off-chain information, thereby tying an entity's off-chain identity to a corresponding blockchain address. This is a desirable property when it is important to maintain consistent digital identities. Using the same keys to sign both the JWDs involved in the exchange of questions and answers and the blockchain transactions that deploy the corresponding smart contract and initiate susbequent interactions with the contract, pegs the two activities. Any information that is conveyed by a JWD can be associated with an immutable, time-stamped blockchain transaction, and both the on-chain and off-chain data are cryptographically proven to be generated by the entity in possession of the common private key.

Dual integration: Dual integration is the process of integrating information that exists outside of the context of a blockchain network with information stored on the blockchain. Dual integration provides an alternative mechanism for associating on- and off-chain signatures using keys. Instead of using the same keys for signing both JWDs and blockchain transactions, a dual integration enables a party to use two different sets of key: one set of keys to signs JWDs and one set to sign blockchain trasnactions. The two sets of keys are cryptographically associated using dual integration:

1. deploy a smart contract (from an unrelated blockchain address) that is capable of storing two checksums
2. produce the checksums (also known as "hashes" or "digital fingerprints") of the public keys associated with each of the keypairs (the keypair used for signing JWDs and the keypair used for signing blockchain transactions)
3. send a transaction storing the checksums to the smart contract

Although dual integration is more complex than simply using the same keys to sign both on- and off-chain data, it has two advantages: (1) the proof that the signer of a JWD and a blockchain transaction are the same party is concealed and only selectively revealed, and (2) a different signing algorithms and/or curve can be used to sign a JWD than is required by a specific smart contract platform for signing blockchain transactions. The association between the signer of a JWD and a blockchain transaction is concealed because it is impossible to associate a party's blockchain transactions to the party using only information gained from the party's JWD signatures, unless the party that signed the JWD reveals a public key that produces the same hash as the checksum stored in a previously deployed contract. Once this public key is revealed (provably associating the public key with the party that presents it) any past, present or future blockchain transactions sent using the party's other keypair can be provably linked through the revealed public key. The signing algorithms and/or curves used by each set of keypairs can be different because all that is needed to associaed the keypairs is a smart contract that stores the hash of the public key associated with each of the keypairs.

4.3.2 Parallel state transitions

Pegging on- and off-chain activities is simplified when activities conform to specific states, linked via defined state transitions.

A simple illustration of question-and-answer exchange involves two parties: a **data provider** which holds data in its custody, and a **questioner** who asks questions of the data provider by requesting that the provider run certain code against the provider's protected data. A typical interaction consists of a series of discrete steps:

1. The questioner creates a question by specifying pre-vetted code to execute against a particular dataset, payment for the cost of execution, and any other relevant information.
2. The questioner sends the prepared question to the data provider.
3. The data provider receives the question and validates that it has access to the requested dataset. If not, the data provider rejects the question. If the data provider does have access to the requested dataset, the provider calculate the

risk function for each of the question's associated attributes to determine the likelihood that each attribute holds if the question is executed against the requested dataset. Based upon the data provider's defined thresholds specifying the attributes required to protect for the specified dataset, the provider determine if the question is safe to answer.

4. If the question is safe, the data provider executes the code against the specified dataset and sends the computed answer to the questioner.
5. The data provider receives payment for the computation.

An protocol that pegs the exchange of questions and answers to blockchain transactions, should faciliatate a blockchain transaction corresponding to each of the stages in the above interaction.

Smart contract state transition system. A smart contract can be built as a state transition system. As an illustration, the human-readable smart contract code for `SingleDataProviderEscrowContract` is presented as a state transition system in Figure 4-1 (complete code provided in Appendix A).[5] State transitions within a smart contract can be triggered by sending a valid message or transaction to the address of a contract account. Every time a contract account receives a message or transaction its code activates, and this code can update the contract's state.

[5]Ethereum has four special-purpose, contract-oriented languages: Serpent (Python inspired), Solidity (JavaScript inspired), Mutan (Go inspired) and LLL (Lisp inspired). Each of these high-level languages compile down to low-level Ethereum Virtual Machine (EVM) code. The `SingleDataProviderEscrowContract` and all other smart contracts presented in this thesis are written in Solidity.

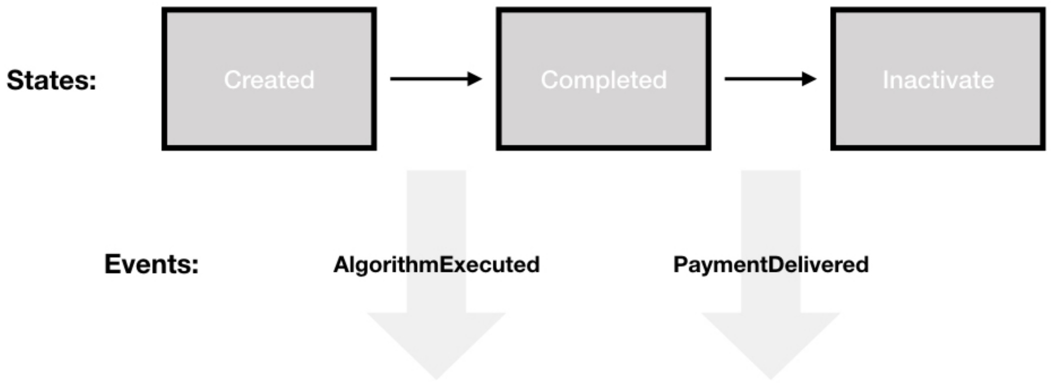

Figure 4-1: Smart contracts as state transition systems.

The `SingleDataProviderEscrowContract` first defines the valid set of states: `Created`, `Completed`, and `Inactive`, and initializes its starting state to `Created`.

```
enum State { Created, Completed, Inactive }
State public state = State.Created;
```

Listing 4.3: Declaration of valid states.

Events are used in Ethereum to facilitate communication between smart contracts and their user interfaces, and can be defined to correspond to state transitions. The `SingleDataProviderEscrowContract` begins in the `Created` state when it is first deployed by a questioner. After a data provider executes the questioner's requested algorithm against the specified dataset, the data provider sends a hash of the results to a function in the smart contract. This function call triggers an `AlgorithmExecuted` event and a state change from `Created` to `Completed`.

After the algorithm is executed and the results are delivered to the questioner, the data provider is able to retrieve the agreed upon payment for the computation from the smart contract (for additional explanations on this pull payment system, refer to Section 4.3). Following successful payment withdrawal, a `PaymentDelivered` event is emitted and the contract transitions from the `Completed` to the `Inactive` state.

```
event AlgorithmExecuted(address target, address questioner, string
   resultsUri);
event PaymentDelivered(address from, address to, uint amount);
```

Listing 4.4: State transition events.

Modifiers can be defined to ensure that state transitions occur as anticipated, resulting in improved code readability and heightened defense against race condition attacks such as the reentrancy and transaction ordering dependence attacks.[6] In this contract, a modifier is declared to validate that the contract is in an expected state prior to the execution of any modified functions.

```
modifier inState(State _state) {
  require(state == _state);
    _;
}
```

Listing 4.5: Assertions on smart contract state.

Functions that include the `inState` modifier will not execute if the current state of the contract does not match the state specified by the modifier. In order to call the `confirmAlgorithmExecuted` function, the contract must be in the `Created` state. This provides both a sanity check and can aid in protecting against race condition attacks. If the contract is in the proper state, the function will begin execution and the contract's state can be updated—here, the state transitions from

[6]A danger of calling external contracts is that they can take over control flow and make changes to data that the caller was not expecting. Reentrancy attacks exploit functions that can be called repeatedly. An attacker calls the vulnerable function multiple times, before previous invocations have completed, causing the different invocations to interact in destructive ways. A reentrancy attack involves executing malicious code within a single transaction. A transaction ordering attack exploits the race condition inherent to blockchains: the order of transactions within a block is vulnerable to manipulation. A user might therefore assume a particular state of a contract which does not hold when her transaction is processed.

`Created` to `Completed` and emits an `AlgorithmExecuted` event.

```
function confirmAlgorithmExecuted(string _resultsUri)
   onlyTarget()
   inState(State.Created)
   public
   returns(bool success)
{
   resultsUri = _resultsUri;
   AlgorithmExecuted(msg.sender, questioner, resultsUri);
   state = State.Completed;
   return true;
}
```

Listing 4.6: Confirmation of algorithm execution.

After the contract reaches the `Completed` state, the funds stored within it can be withdrawn (the value associated with the contract address). If the withdrawal is successful, a `paymentDelivered` event is emitted and the contract reaches the `Inactive` state.

```
function withdraw()
   onlyTarget()
   inState(State.Completed)
   public
   returns(bool success)
{
  uint payment = amount;
  amount = 0;
  if (msg.sender.send(payment)) {
   PaymentDelivered(questioner, msg.sender, amount);
     state = State.Inactive;
     return true;
   } else {
     amount = payment;
```

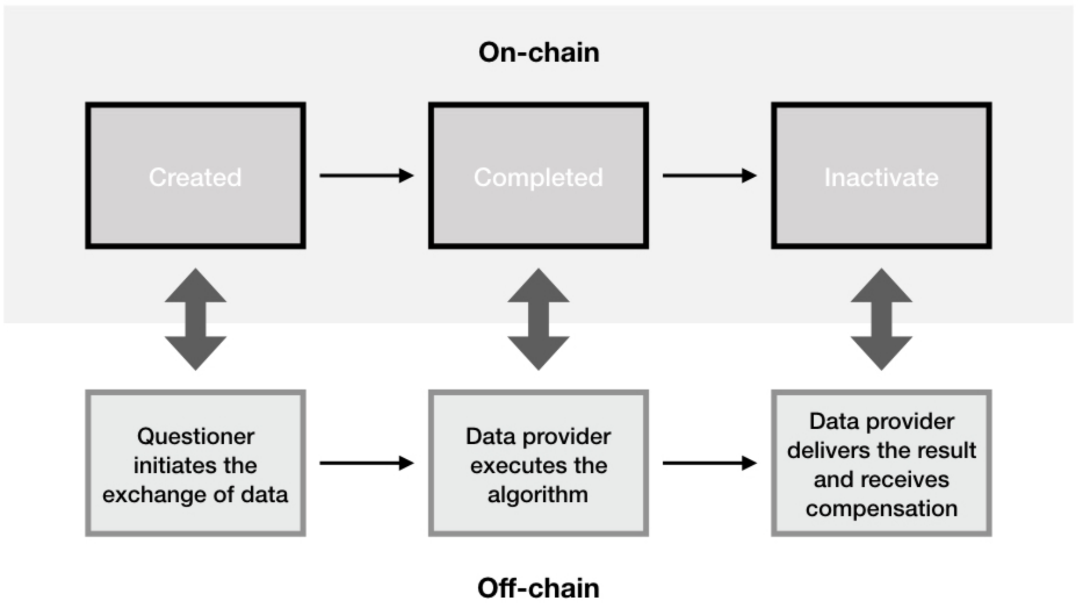
On-chain
Created
Completed
Inactivate
Questioner initiates the exchange of data
Data provider executes the algorithm
Data provider delivers the result and receives compensation
Off-chain

Figure 4-2: Parallel state transitions on- and off-chain.

```
    return false;
  }
}
```

Listing 4.7: Withdrawal pattern for payments.

Developing smart contracts as state transition systems makes the code substantially easier to reason about. Additionally, this design can more clearly map directly to real-world interactions.

As demonstrated in Figure 4-2, this exchange can be represented as a state transition system with a one-to-one mapping to the previously detailed states of the `SingleDataProviderEscrowContract`.

In order to initiate the execution of an algorithm against a dataset in the data exchange, a questioner initializes a new smart contract. Smart contracts are created by sending a new transaction to the `0x0000000000000000000` address with the contract's EVM code included in the data field.

Code can access the value, sender, and data of an incoming message (`msg.sender`, `msg.value`, and `msg.data` respectively), as well as block header data. In the initializer for `SingleDataProviderEscrowContract`, the deployer of the contract (`msg.sender`) specifies values for each of the four initialization parameters (`algorithmUri`, `target`, `dataId` and `amount`) in the data field, and the deployer herself is set as the `questioner`. The `algorithmUri` corresponds to the Uniform Resource Identifier (URI) where the algorithm to execute can be retrieved from; the `target` corresponds to the address of the data provider that holds the relevant data; the `dataId` identifies the specific dataset; and the `amount` represents the offered payment for executing the algorithm against the specified data. The `payable` modifier indicates that ether may be sent to the function (i.e. `msg.value` can be greater than zero).

```
function SingleDataProviderEscrowContract(string _algorithmUri, address
   _target, string _dataId, uint _amount)
   payable
{
   questioner = msg.sender;
   algorithmUri = _algorithmUri;
   target = _target;
   dataId = _dataId;
   amount = _amount;
}
```

Listing 4.8: Smart contract initialization.

Trustless decentralized escrow. Smart contracts provide a powerful mechanism for trustless and decentralized escrow. Contract accounts hold balances in the same way that externally owned accounts hold balances, so can act as provably secure escrow providers. A function in the contract code—the initializer and/or some other function(s)—can be designated as `payable`, which allows ether to be sent to the function in a transaction or message. This ether is then credited to the contract's account.

A questioner who deploys a smart contract can fund the contract's account with enough ether to cover the cost of executing an algorithm in the question-and-answer system. By simply examining the publicaly visible smart contract account associated with a given question, the data provider can ensure that payment for the algorithm has been reserved, and is therefore guaranteed upon returning an answer. If some condition must be met in order to withdraw ether from the contract account, the smart contract is providing escrow. In the `SingleDataProviderEscrowContract` the condition is an indication that the data provider has performed the computation and returned the result to the questioner. Routing the payments associated with the exchange of questions for answers through the pegged blockchain further bolsters the two-way peg.

4.3.3 Alternative state transition designs

The state transition conditions outlined are representative of a typical data exchange, yet not entirely comprehensive. Bismuth aims to be flexible and to enable participants in a question-and-answer system to specify contracts that model their anticipated real-world exchange of questions for answers. To best accommodate this, Bismuth provides multiple smart contracts that can be used to peg a particular question. The exact state transitions coded into the various available smart contract might differ slightly, though roughly all follow a similar lifecycle (starting with `Created` and ending in `Inactive`). Appendix B: `MultiDataProviderBidContract.sol` is an example of a more complex smart contract that accommodates multiple data providers and incorporates a bid-ask mechanism. The state transitions associated with this contract differ from those in `SingleDataProviderEscrowContract`, as an additional state, `Locked`, is included to represent the point at which a payment price is settled upon between the data providers and the questioner, and the funds are locked. As exhibited in Figure 4-3, additional events are emitted in this contract that correspond to behaviour involved in locking the contract, while the rest of the state transition system remains identical to that of the `SingleDataProviderEscrowContract`.

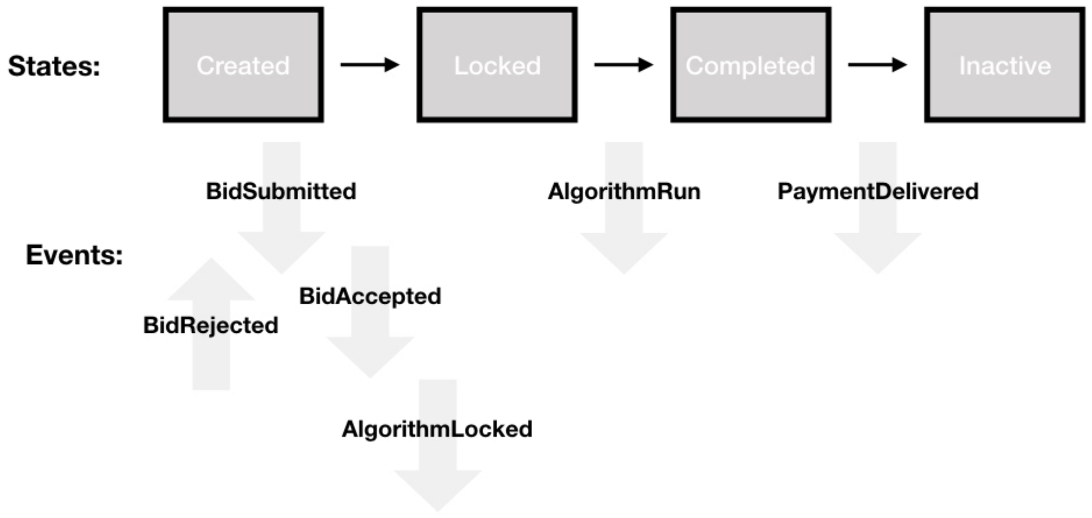

Figure 4-3: Bid-ask state transition system.

4.3.4 Blockchain integration patterns

In the design presented, Bismuth integrates with a pegged blockchain by deploying independent smart contracts for each instance of question and answer exchange. Independent smart contracts represent a fully decentralized solution, where each questioner is directly responsible for, and has full control over, the two-way peg associated with their question. Questioners can exercise this control by choosing the smart contract that they associate with their question. In the `SingleDataProvider-EscrowContract`, the contract provides escrow functionality and only one data provider is involved in the exchange. A question could instead require computations from multiple data providers, or a questioner might opt not to use the contract for escrow, preferring an alternative payment mechanism. The negative implications of this approach are (1) that the size of the bytecode associated with the smart contracts might be large—especially for more complex or feature-laden contracts—and therefore expensive to deploy, and (2) the available smart contracts must be hosted somewhere. However, independent smart contracts still provided more benefits than the alternative patterns considered.

Smart contract factory. An alternative to deploying independent smart contracts is to create a smart contract "factory", which holds the bytecode associated with different smart contracts. Instead of a questioner retrieving the smart contract bytecode that she would like to associate with her question and deploying it herself, the smart contract factory could host all of the available smart contract types, and she could simply send payment and request the factor to deploy a new contract on her behalf with specified parameters (algorithm URI, data provider address, dataset identifier, etc.). This approach solves the hosting issue, as all participants in the Ethereum network could see the available contracts as data stored on the blockchain, yet introduces a source of centralization. Someone would need to be responsible for deploying and maintaining this original factory smart contract.

Sending data with transactions. An additional option is to include the relevant open data exchange information as data in the transaction, rather than embedded within a smart contract. A transaction could be sent directly from a questioner to a data provider with the value set as the cost of the algorithm's execution, and the algorithm and dataset identifiers included as additional data. This is a smaller transaction, so would require less gas to send. However, it severely limits the potential functionality provided by Bismuth: the complexity associated with a question involving input from multiple data providers grows immensely; payment would need to be provided up-front—there could be no escrow option; adding additional functionality would be challenging, if not impossible.

4.4 Storing data and value on the blockchain

In a public blockchain, all account data (including contract code) and all transactions are visible to every participant in the network. Beginining from the genesis block, each discrete state can be replayed, and it is impossible to remove information from the blockchain's history. The visibility of information on a public blockchain greatly effects Bismuth's design.

4.4.1 Answer verification and payment

Listing 4.6 hides an important detail; to call the function which confirms execution of the requested algorithm, the function takes the URI from which the computed results can be retrieved. The URI is required to transition the smart contract from the `Created` state to the `Completed` state. The contract acts as an escrow provider, so there must be some condition which the data provider can satisfy to enable the provider to withdraw payment upon answering the question posed by the questioner. The satisfiable condition should provide a guarantee that the data provider did in fact execute the requested algorithm and send the answer to the original questioner. The difficulty arises in devising a mechanism in which no party is incentivized to act dishonestly: the data provider does not try to retrieve payment without executing the code, or without sending the results to the questioner, and the questioner does not try to prevent the delivery of payment to the data provider after receiving the answer.

The simplest solution is arguably be to send the computed answer itself as input to the function. This solution virtually guarantees that the question was answered, as the answer is recorded on the blockchain for all network participants to verify—all transactions, as well as the current and historical states and account data, are visible to every participant in the Ethereum network. Additionally, the questioner is guaranteed to receive the result, as the result is publicly available on the blockchain. However, this solution has two fundamental flaws. First, there is a cost associated with the size of a transaction and the amount of data that it adds to the blockchain. If the answer includes a large number of bytes, returning the answer might be prohibitively expensive. Second, in the context of a question-and-answer system that charges questioners for sending questions to data providers to answer, publicly providing the answer disincentivizes data providers participation. Once a data provider answer a question and reveals the data on the blockchain, no questioner will pay to ask the question again as the answer is already publicly available. Instead, only the original questioner should receive the answer.

The smart contract could instead require that the data provider send a hash of the computed results. This gives a weak guarantee that the question was answered (the guarantee is weak because an arbitrary string of characters could be provided as a hash, rather than a legitimate hash), and no guarantee that the questioner received the original computed results. However, the approach resolves the issue of the public visibility of computed answers on the blockchain (assuming the data providers uses a secure, one-way hashing algorithm), and improves upon the issue of storage size, as a hash of the results will have a smaller size than the raw results.

Bismuth utilizes neither of the aforementioned strategies, and instead requires the URI from which to retrieve the results. The URI is encrypted using the questioner's public key so that only the original questioner knows where to retrieve the answer. Although this strategy still provides only a weak guarantee that the algorithm was executed and the results were delivered to the questioner, there is a clearer tie between the delivery of results and the provided argument to the `confirmAlgorithmExecuted` function. Section 6 addresses how Bismuth provides the primitives required to incorporate a reputation management system into the question-and-answer system, which would strengthen the guarantee that the data provider did in fact make the answer available at the provided URI—questioners could report data providers that fail to make the answer available at the URI they provide. The benefits of sending an encrypted URI additionally include:

- only the original questioner can access the results, as the address from which to retrieve the answer is encrypted using the questioner's public key;
- if the URI is compromised for some reason (e.g. the hashing algorithm used is broken), the answer located at the URI can be moved;
- in the majority of anticipated cases, the size of an encrypted URI is smaller than a hash of the results, reducing the associated storage costs;[7]
- delivery of the answer is managed directly through the blockchain, rather than

[7]To further reduce storage costs, Ethereum allows events to be employed as a cheaper form of storage. When an event is emitted, the corresponding logs are written to the blockchain. These logs are inaccessible to other contracts, but can be viewed by outside participants. The logs cost only 8 gas per byte, whereas contract storage costs 20,000 gas per 32 bytes.

through some auxilirary protocol.

Of course, none of these proposed schemes ensure that the data provider answered the question correctly, whether though unintentional error or malicious deception. A method for verifying the accuracy of returned answers without access to the raw data used to generate the answers is an incredibly interesting question, yet highly complex and out of the scope of this work. The aforementioned reputation system might additionally prove beneficial in detecting data providers who consistently return questionable answers.

From answer delivery to payment. Listing 4.7 demonstrates functionality to pay data providers via the withdrawal pattern. After the contract has reached the `Completed` state, the data provider is eligible to collect payment from escrow for answering the posed question. No input is required to call this function; the `inState` modifier confirms that the contract is "completed", i.e. that the questioner has received an answer and that the data provider is therefore eligible to collect payment. The withdrawal pattern utilizes pull payments in place of push payments, protecting against malicious behavior if the data provider address is tied to a malevolent contract account. When a contract receives funds, it invokes a special function on the contract, called a fallback function. This behavior gives the contract control over the control flow of the transaction that invoked it, and could potentially allow contract code at the recipient account to act maliciously. The possibility is avoided by requiring data providers to withdraw their payment, rather than using the smart contract to send payments.

5 Integration with the OPAL server

The OPAL server is a working implementation for "sending the code to the data". OPAL is designed with a plugin architecture to allow users of the software to easily customize behavior (e.g. support for interchanging different distributed ledgers implementations to provide the two-way peg). The system is composed of document objects represented by signed JWDs, and nodes which interact with these objects. This section defines the OPAL objects and nodes, and details how they interact with one another. It concludes with a discussion of how Bismuth integrates into the OPAL server.

5.1 OPAL documents

Documents in OPAL are represented by signed JWDs—a data storage format that natively supports digital signatures used to authenticate the document contents. OPAL JWDs support multiple signing algorithms (ES256, ES384, ES512, RS256, RS384, and RS512) and multiple signers (each of which can independently select a signing algorithm with which to sign). OPAL includes four basic document types: `Question Documents`, `Smart Contract Documents`, `Query Transaction Documents`, and `Query Result Documents`, each of which can contain some subset of five common payload properties.

`uuid`	Globally unique identifier
`uri`	Location of the document
`iss`	Issuer of the document
`iat`	UNIX timestamp in seconds
`exp`	UNIX timestamp in seconds

Table 5.1: Common OPAL payload properties.

5.1.1 Question document

A `Question Document` represents a question as executable code, a plain text description, and a question identifier. The executable code can be represented in any language used for computation (C++, R, Map-Reduce, etc.); the language of the code is specified in the document. A `Question Document` can additionally specify a certain smart contract that must be used to facilitate its two-way peg, as well as pricing information for the cost of executing the question.

`qid`	REQUIRED	question identifier
`language`	REQUIRED	media-type of executable code
`code`	REQUIRED	executable code
`version`	REQUIRED	version number
`title`	RECOMMENDED	human recognizable name for the question
`description`	RECOMMENDED	explanation of the meaning of the results
`type`	RECOMMENDED	type of question (aggregate, etc)
`cid`	OPTIONAL	smart contract identifier
`ask`	OPTIONAL	asking price of the question
`cur`	OPTIONAL	currency of the asking price

Table 5.2: `Question Document` payload properties.

5.1.2 Smart contract document

A `Smart Contract Document` holds smart contract code (e.g. EVM bytecode) to deploy to a smart contract platform to provide the two-way peg. The document can be written to support any smart contract platform, and used with any smart contract language.

`cid`	REQUIRED	smart contract identifier
`platform`	REQUIRED	smart contract platforms (Ethereum, etc.)
`language`	REQUIRED	language of source code (Solidity, etc.)
`code`	REQUIRED	source code for the smart contract
`version`	REQUIRED	version number
`title`	RECOMMENDED	human recognizable name for the smart contract
`description`	RECOMMENDED	explanation of the behavior of the smart contract

Table 5.3: `Smart Contract Document` payload properties.

5.1.3 Question transaction document

A `Question Transaction Document` represents the information associated with a specific instance of asking a question.

`tx`	REQUIRED	transaction identifer
`qid`	REQUIRED	question identifier
`iss`	REQUIRED	URI or identifier of the party asking the question
`providers`	REQUIRED	array of objects describing the data providers
`cid`	REQUIRED	contract identifer
`qp`	OPTIONAL	JSON object representing question parameters
`rt`	OPTIONAL	response type requested by the questioner
`cb`	OPTIONAL	URI for callback
`total`	OPTIONAL	total value of smart contract
`cur`	OPTIONAL	currency of the 'total' value

Table 5.4: `Question Transaction Document` payload properties.

5.1.4 Question results document

A `Question Results Document` contains the computed answer to a question.

`content-type`	REQUIRED	value indicating the format of the ‘results‘
`results`	OPTIONAL	data representing the results

Table 5.5: `Question Results Document` payload properties.

5.2 OPAL nodes

A node is a computer that runs a distributed ledger client software, which validates or rejects new incoming data. OPAL supports four node types, and a client can run one or more nodes.

- **Question provider node.** A `Question Provider` node accepts question proposals, signs questions (at the request and discretion of an authorized user), and publishes all questions with search and changes endpoints. The `Question Provider` acts as a trusted party.
- **Contract provider node.** A `Contract Provider` node publishes smart contracts that can be used to facilitate the two-way beg between the exchange of questions and answers and blockchain transactions.
- **Data provider node.** A `Data Provider` node is an entity that owns or controls dataset that is available for questions to be asked of it. A `Data Provider` node signs and publishes information describing the schema of their available datasets.
- **Questioner node.** A `Questioner` node is the node that asks questions. The `Questioner` signs a document indicating the targeted data providers and dataset, question and pegged smart contract, and any additional supporting information required.

The OPAL architecture permits each type of nodes to engage in a specific type of activity. Questions can be proposed by any authorized party; a proposed question is simply an unsigned `Question Document` payload. A proposed question can optionally include proposed attributed for a `Question Provider` node to attest to.

`Question Provider` nodes act as trusted parties responsible for signing attributes associated with proposed questions. `Question Provider` nodes examine the proposed questions (which may already included suggested attributes), and can attest to one or more of the suggested attributes, or attest to a new attribute. Each attestation must carry an associated risk function (as defined in Section 4). A proposed question that is signed by a `Question Provider` node is a signed `Question Document`. As an example, a `Question Provider` node run by a US federal bank regulatory agency might examine all questions proposed by banks operating in the United States. If a bank proposes a computation to calculate leverage ratio, the `Question Provider` node run by the federal bank regulatory agency could attest that the calculation is complaint with US federal capital requirement regulations.

The number and type of signatures required for a `Question Document` to be valid to execute against a protected dataset might depend on the domain area, the particular dataset, the attributes attested to, the signers, or a number of other factors. Each `Data Provider` node determines which vetted questions it will answer, based on criteria it defines.

A `Contract Provider` node is responsible for signing smart contract code that can be used to provide a two-way peg associating question and answer interactions to blockchain records. Authorized parties propose smart contract for different smart contract platforms, represented as unsigned `Smart Contract Document` payloads, written in whatever language is supported by the particular platform. `Contract Provider` node can then verify that the smart contract code is safe and behaves as intended by signing the proposed `Smart Contract Document` payload.

In order to ask a `Data Provider` node a question, a `Questioner` node must invoke a `Question Transaction Document` with the question and other necessary information. A `Question Transaction Document` associates a `Question Document` with a `Smart Contract Document`, setting up the two-way peg mechanism, and is then sent to a `Data Provider` node to be executed. The results of the question's execution are returned to the `Questioner` in a `Question Results Document`.

5.3 OPAL interactions

In the classical instantiation of a question-and-answer system, one party acts as the questioner and a different party asks as the answerer. The inclusion of pre-vetted questions and verified attributes necessitates two additional parties: one party to propose questions and another to provide relevant attestations. These parties can be different entities, or a single entity running multiple services and acting as multiple nodes. As in the aforementioned example, a commercial bank could use a pre-vetted leverage ratio calculation to compute leverage ratios using the bank's own data, thereby acting as both a `Questioner` and a `Data Provider`.

Each permutation of entities and OPAL nodes implies a unique interaction. When a `Questioner` and a `Data Provider` are different entities, the `Questioner` is taking advantage of the data collected and stored by another entity to answer a question relevant to the `Questioner`. In this scenario, the `Questioner` benefits from OPAL in that the `Questioner` is able to answer questions for which it does not have the data necessary to arrive at an answer. The `Data Provider` benefits from OPAL by agreeing to only execute algorithms that have been pre-vetted by a `Question Provider` to include some set of verified attributes, such that the `Data Provider` is aware of the results of executing the `Questioner`'s algorithm and returning the answer—i.e. that the result is free from discrimination, privacy-preserving, or compliant with a given regulation or piece of legislation. Additionally, the OPAL system provides guarantees regarding the exchange of payments for answers and the validity of provided attestations through the use of Bismuth. Bismuth is able to provide these guarantees when provided with a smart contract, supplied by a `Contract Provider` node, capable of supporting the specific question-and-answer exchange through a two-way peg.

5.3.1 Verified computations

Another interesting permutation of entities and OPAL nodes occurs when the `Questioner` node and the `Data Provider` node represent the same underlying entity. In this scenario, the `Questioner` technically has access to the data necessary to answer a particular question (as the `Questioner` is also the `Data Provider`), seemingly implying no use for a question-and-answer system as proposed. However, OPAL remains highly relevant through its support for verified computations. Similarly to how a trusted party can attest to a question being non-discriminatory—through asserting a non-discriminatory attribute associated with a given question—a trusted party can likewise attest to the correctness of a question. In the question-and-answer system proposed, a question is represented in code as an algorithm, computation or query. A correct algorithm, computation or query is one which consistently returns a result as specified. A verified computation is therefore a computation which has been vetted for correctness. Two simple examples clearly illustrate the concept:

1. In the previous example of a commercial bank using a pre-vetted leverage ratio calculation to compute ratios using its own data, the bank is acting as both a `Questioner` and as a `Data Provider`. OPAL is useful in this scenario because it exposes the code behind the leverage ratio calculation to the outside world, enabling the code to be verified for correctness and compliancy with federal banking regulations. This process permits the bank to publicize its compliancy, and allows the correctness of the calculation to be verified by outside parties—both of which facilitate increased transparency in the banking sector.
2. An alternative scenario is one in which an entity holds the data required to answer a known question, but does not know how to go about answering the question. Consider a Fitbit user who has collected large amounts of personal health data through his/her Fitbit device. The user may wish to calculate his/her propensity for some ailment or injury based upon the collected data, but does not want to reveal the data to an outside party. With an OPAL question-and-answer system, the user can select the computation corresponding to his/her question that, for example, has been verified as correct by Harvard Medical Center scientists, and compute the answer to the question without needing to reveal his/her health data to any outside party.

5.4 Bismuth integration

Bismuth is incorporated into the OPAL server as a library providing multi-blockchain support. It includes standard smart contract implementations that are generally applicable for pegging standard OPAL exchanges, such as the `SingleDataProvider-EscrowContract.sol` and `MultiDataProviderBidContract.sol` contracts provided in appendices A and B. The Bismuth library uses the adapter pattern to support clients across multiple blockchains. These clients each implement the two-way peg interface specified by Bismuth for their respective smart contract platforms. The clients abstract out the differences between the underlying platforms, such that a given operation is only either supported or not supported by a particular platform. For example, in the case of the `EthereumClient`, the client provides a wrapper around Ethereum's standard Web3 JavaScript library to manage interactions with the Ethereum blockchain, including the ability to directly deploy the smart contracts used to peg the exchange of questions and answers. The Bismuth multi-blockchain clients further support using the same keypairs on-chain as are used for signing data within the OPAL system.

6 Open questions and future work

This thesis present a design, mechanism and program for verifying responsible data usage, yet many open questions remain. These open questions provide the basis for future work and investigation, and fall into three broad categories: (1) the use of trusted parties, (2) verifying off-chain action, and (3) cryptographic protocols and quantum computing.

6.1 Trusted parties

Particularly in the cryptography and cryptocurrency environments, trusted parties are viewed as security vulnerabilities. A primary design goal for public blockchains is achieving "trustless" cooperation and consensus. Apparent from the previous discussion of the OPAL question-and-answer system, trusted parties play a critical role in verifying the attributes associated with questions. This work acknowledges that the claim of ensuring responsible data usage through pre-vetted code designed to promote "safe" questions and answers is contingent upon the competency of the trusted parties providing attestations of attributes. Unfortunately, the challenge of verifying attributes is even more difficult than it initially appears when only considering questions in isolation.

The difficulty arises from the often unexpected and unpredictable interactions between questions. Anonymized datasets are most typically de-anonymized not be cracking the underlying anonymization scheme, but rather by pulling in additional data from other sources to add dimensionality to ostensibly "anonymized" dataset. A question asked in isolation might be accurately represented by its verified attributes, however, the original verification of the attributes might not well-account for how the question could be combined with seemingly unrelated questions, potentially invalidating the risk functions associated with the attributes. This could result in the risk of an attribute not holding true for a given dataset dramatically increasing. Even taking into consideration all the currently available, pre-vetted questions when determining the appropriate attributes and associated risk functions to assign to a proposed question is insufficient. The pool of proposed questions grows over time, so any analysis that considers the possible interactions with previously vetted code quickly becomes outdated.

Luckily, setting an upper-bound on how out of date the analysis of a question's attributes might be is straight-forward. JWDs can optionally include an `exp` field which provides an expiration time, after which the JWD is invalidated. The expiration time chosen provides the upper-bound on how long a question can remain available and valid in the context of the question-and-answer exchange before the JWD storing the question (the `Question Document`) is invalidated and the question and its attributes must be re-visited. At this time, the attributes can be re-evaluated taking into consideration newly approved questions.

6.1.1 Formal verification

The ideal solution to replace trusted parties verifying attributes of the code being executed against sensitive data is a robust formal verification system. An ideal formal verification system would be able to deterministically analyze guarantees about the results of executing a given piece of code on different dataset schema, taking into consideration all previously analyzed questions to check for undesirable interactions. Some work has been done in this area, in particular work verifying differential privacy guarantees for interactive systems, but the research has a long way to go [Tschantz et al., 2011].

6.2 Unverified actions off-chain

Hinted at in Section 4.4.1, the issue of verifying the off-chain actions taken by participants in the question-and-answer system is significantly more challenging than verifying their statements or on-chain actions. Digital signatures can provide secure, cryptographic guarantees over messages and documents, yet these represent statements rather than explicit actions. In comparison, a blockchain transaction is an action. Technically, Bismuth pegs messages (such as hashes or encrypted values) and signed JWDs to blockchain transactions, rather than pegging physical-world actions involved in exchanging questions and answers to blockchain transactions. The challenge arises in matching signed messages and documents to physical-world actions as rigorously as possible so that the corresponding blockchain transactions are accurately represent physical-world behavior.

In Section 4.4.1, a data provider is able to receive payment for computing the question's code after providing the encrypted URI specifying where the questioner can retrieve the results. However, unlike a digital signature, this does not provide a verifiable guarantee. The data provider could provide any arbitrary value; no mechanism validates that (1) the provided value is properly encrypted, (2) the provided value is in fact a URI, (3) the URI exists, (4) computed results are stored at the URI, and (5) the computed results represent the accurate answer to the original question. Some of the missing checks result from the public nature of computing on blockchains and the resulting limitations on the type of data that can be processed, while some might simply be intractable. On the other hand, guaranteeing that the questioner will pay for their question is a more tractable concern due to the escrow abilities offered by smart contracts which moves the off-chain payment action on-chain.

Answer verification presents an additional unsolved question. Ideally, upon receiving an answer the questioner should be able to verify that the answer is in fact correct. However, this check is incredibly difficult to make without access to the raw data. In the current design, the questioner must blindly accept the answer provided.

6.2.1 Reputation management system

Although not a complete solution, the concerns arising from unverified actions off-chain can be mitigated with an effective reputation management system. A reputation management system would strengthen the claims made by participants in the question-and-answer system by incorporating a participant's historical actions and behaviour. For example, if a data provider submits an invalid URI to a smart contract peg and proceeds to withdraw payment, the questioner will soon discover that the URI is invalid and the results are not available. The questioner can then report the data provider's bad behavior. Consistent use of digital signatures associated with digital identities provides the primitives required for managing reputation. Good actors are incentivized to continue using the same digital identifiers so that their positive reputations follow them, while bad actors are forced to constantly adopt new identities in an effort to shed bad reputations, resulting in a perpetually short transactional histories—a possible warning sign for other participants.

6.3 Cryptographic guarantees

The work presented in this thesis relies heavily on cryptographic digital signatures. Unfortunately, there are known flaws with the Elliptic Curve Digital Signature Algorithm (ECDSA) implemented in both Bitcoin and Ethereum, in particular with respect to resiliency against quantum computing attacks. Transactions are signed using ECDSA, and when a transaction is sent the sender's public key is revealed. Quantum computer compromise ECDSA, making it easy to derive the private key associated with a public key. It is highly possible that quantum computers will be able to work faster than transactions can be confirmed. If signatures can be broken faster than transactions can be confirmed, an attacker could sign a transaction that spends coins from an account with a pending transaction before the original transaction is ever accepted, possible draining the original sender's funds. In Metropolis, the next release of Ethereum, accounts will be able to specify their own security scheme for validating transactions. The change permits opt-in, quantum-resistant mechanisms, such as Lamport signatures, to be used to improve the security of Ethereum transactions.

7 Conclusion

The OPAL paradigm, and the extensions presented in this thesis, enable the data collected and stored by geographically disparate individuals, organizations, companies and governments to be productively utilized in a responsible manner through the verified exchange of vetted questions and answers. Each individual, organization, company and government represents a potential data provider with a unique set of data and responsibilities—stemming from legal, ethical, and/or business requirements—that the provider must maintain. Access to an open store of pre-vetted questions enables data providers to be confident that the answers they provide are safe from inaccuracy, abuse and misuse. Data providers can publicly commit to only using their data to answer questions that conform to a high ethical and legal standard, fostering an environment of responsible data usage.

The design for verified question attributes proposed in this thesis utilizes JWDs to enable multiple trusted parties to each provide signed attestations of unique attributes and/or risk functions over the same question—optionally using different signing algorithms. JWDs additionally permit questions to be invalidated after a specified expiration time, encouraging frequent re-evaluation of each question's assigned attributes. Pegging the exchange of safe questions and answers through blockchain transactions further produces immutable, time-stamped and audit-able records, which can be used to observe the attributes and associated risk levels that data providers accept when answering the questions posed to them. The Bismuth program developed as part of this work supports the aforementioned ideas through a general interface, and a specific implementation of a blockchain client for the Ethereum smart contract platform.

A paradigm that emphasizes collaborative and productive consideration of the answers revealed by data and their implications helps promote responsible data usage. The realization of an open and thoughtful question-and-answer system allows data owners to profit from the information they hold, while simultaneously encouraging the data to be used to foster data-driven decisions and to improve society.

A SingleDataProviderEscrowContract.sol

```
pragma solidity ^0.4.13;

contract SingleDataProviderEscrowContract {

   address public questioner;
   string public algorithmUri;
   address public target;
   uint public amount;
   string public resultsUri;

   enum State { Created, Completed, Inactive }
   State public state = State.Created;

   function SingleDataProviderEscrowContract(string _algorithmUri,
       address _target, uint _amount)
      payable
   {
      questioner = msg.sender;
      algorithmUri = _algorithmUri;
      target = _target;
      amount = _amount;
   }

   modifier onlyQuestioner() {
      require(msg.sender == questioner);
      _;
   }

   modifier onlyTarget() {
      require(msg.sender == target);
      _;
```

```
}

modifier inState(State _state) {
   require(state == _state);
   _;
}

event AlgorithmExecuted(address target, address questioner, string
    resultsUri);
event PaymentDelivered(address from, address to, uint amount);

function confirmAlgorithmExecuted(string _resultsUri)
   onlyTarget()
   inState(State.Created)
   public
   returns(bool success)
{
   resultsUri = _resultsUri;
   AlgorithmExecuted(msg.sender, questioner, resultsUri);
   state = State.Completed;
   return true;
}

function withdraw()
  onlyTarget()
  inState(State.Completed)
  public
  returns(bool success)
{
  uint payment = amount;
  amount = 0;
  if (msg.sender.send(payment)) {
    PaymentDelivered(questioner, msg.sender, amount);
    state = State.Inactive;
    return true;
  } else {
```

```
            amount = payment;
            return false;
        }
    }
}
```

B MultiDataProviderBidContract.sol

```
pragma solidity ^0.4.13;

contract MultiDataProviderBidContract {

   struct DataProvider {
      uint bid;
      bool isTarget;
      bool bidAccepted;
      bool bidRejected;
      bool algorithmExecuted;
      string resultsUri;
      bool paymentDelivered;
   }

   address public questioner;
   string public algorithmUri;
   mapping(address => DataProvider) public targets;
   address[] targetIdx;

   enum State { Created, Locked, Completed, Inactive }
   State public state = State.Created;

   function MultiDataProviderBidContract(string _algorithmUri) {
      questioner = msg.sender;
      algorithmUri = _algorithmUri;
   }

   modifier onlyQuestioner() {
      require(msg.sender == questioner);
      _;
   }
```

```
modifier onlyTarget() {
   require(targets[msg.sender].isTarget);
   _;
}

modifier inState(State _state) {
  require(_state == state);
   _;
}

event BidSubmitted(address dataProviderAddr, uint bid);
event BidRejected(address dataProviderAddr, uint bid);
event BidAccepted(address dataProviderAddr, uint bid);
event AlgorithmLocked();
event AlgorithmExecuted(address target, address questioner, string
    resultsUri);
event PaymentDelivered(address from, address to, uint amount);

function submitBid(address _dataProviderAddress, uint _bid)
   payable
   onlyQuestioner
   inState(State.Created)
   returns(bool success)
{
   assert(msg.value >= _bid);
   DataProvider memory target = DataProvider({
      bid: _bid,
      isTarget: true,
      bidAccepted: false,
      bidRejected: false,
      algorithmExecuted: false,
      resultsUri: "",
      paymentDelivered: false
   });
   targets[_dataProviderAddress] = target;
   targetIdx.push(_dataProviderAddress);
   BidSubmitted(_dataProviderAddress, _bid);
   return true;
}

function rejectBid()
   onlyTarget
   inState(State.Created)
   returns(bool success)
{
   DataProvider storage target = targets[msg.sender];
   assert(!target.bidAccepted);
   assert(questioner.send(target.bid));
   targets[msg.sender].bidRejected = true;
   BidRejected(msg.sender, target.bid);
   return true;
}

function acceptBid()
   onlyTarget
   inState(State.Created)
   returns(bool success)
{
   targets[msg.sender].bidAccepted = true;
   uint8 numAccepted = 0;
   for (uint i = 0; i < targetIdx.length; i++) {
      if (targets[targetIdx[i]].bidAccepted) {
         numAccepted += 1;
      }
   }
   if (numAccepted == targetIdx.length) {
      state = State.Locked;
   }
   BidAccepted(msg.sender, targets[msg.sender].bid);
   return true;
}
```

```
function lock()
   onlyQuestioner()
   inState(State.Created)
{
   uint numTargets = 0;
   for (uint i = 0; i < targetIdx.length; i++) {
      if (!targets[targetIdx[i]].bidAccepted) {
         targets[targetIdx[i]].isTarget = false;
      } else {
         numTargets += 1;
      }
   }
   if (numTargets > 0) {
      state = State.Locked;
      AlgorithmLocked();
   } else {
      state = State.Inactive;
   }
}

function cancel()
   onlyQuestioner
   inState(State.Created)
{
   state = State.Inactive;
   assert(questioner.send(this.balance));
}

function confirmAlgorithmExecuted(string _resultsUri)
   onlyTarget()
   inState(State.Locked)
   returns(bool success)
{
   targets[msg.sender].algorithmExecuted = true;
   targets[msg.sender].resultsUri = _resultsUri;
```

```
    AlgorithmExecuted(msg.sender, questioner, _resultsUri);
    uint numCompleted = 0;
    for (uint8 i = 0; i < targetIdx.length; i++) {
       if (targets[targetIdx[i]].algorithmExecuted) {
          numCompleted += 1;
       }
    }
    if (numCompleted == targetIdx.length) {
       state = State.Completed;
    }
    return true;
 }

 function withdraw()
    onlyTarget
    inState(State.Completed)
    returns(bool success)
 {
    uint amount = targets[msg.sender].bid;
    // The pending payment for this data provider must
    // be zeroed before sending to prevent reentrancy
    // attacks
    targets[msg.sender].bid = 0;
    if (msg.sender.send(amount)) {
       targets[msg.sender].paymentDelivered = true;
       PaymentDelivered(questioner, msg.sender, amount);
       uint numPaid = 0;
       for (uint8 i = 0; i < targetIdx.length; i++) {
          if (targets[targetIdx[i]].paymentDelivered) {
             numPaid += 1;
          }
       }
       if (numPaid == targetIdx.length) {
          state = State.Inactive;
       }
       return true;
```

```
        } else {
            targets[msg.sender].bid = amount;
            return false;
        }
    }

    function withdrawRemainingFunds()
        onlyQuestioner()
        inState(State.Inactive)
        returns(bool success)
    {
        assert(questioner.send(this.balance));
        return true;
    }

}
```

Bibliography

[Back et al., 2014] 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. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains.*

[Buchholtz et al., 2014] Buchholtz, S., Bukowski, M., and Śniegocki, A. (2014). Big and open data in europe: A growth engine or a missed opportunity. *Warsaw Institute for Economic Studies Report Commissioned by demosEUROPA*, 10.

[Buterin et al., 2013] Buterin, V. et al. (2013). Ethereum white paper.

[Butler, 2008] Butler, D. (2008). Web data predict flu.

[CxSci, 2017] CxSci (2017). Opal. https://github.com/CxSci/OPAL.

[de Montjoye et al., 2014] de Montjoye, Y.-A., Smoreda, Z., Trinquart, R., Ziemlicki, C., and Blondel, V. D. (2014). D4d-senegal: the second mobile phone data for development challenge. *arXiv preprint arXiv:1407.4885.*

[Foundation, 2017] Foundation, T. L. (2017). Hyperledger.

[Greenwood et al., 2014] Greenwood, D., Stopczynski, A., Sweatt, B., Hardjono, T., and Pentland, A. (2014). The new deal on data: A framework for institutional controls. *Privacy, Big Data, and the public good: Frameworks for engagement*, pages 192–200.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[Ohm, 2009] Ohm, P. (2009). Broken promises of privacy: Responding to the surprising failure of anonymization.

[Pentland, 2009] Pentland, A. (2009). Reality mining of mobile communications: Toward a new deal on data. *The Global Information Technology Report 2008–2009*, page 1981.

[Tschantz et al., 2011] Tschantz, M. C., Kaynar, D., and Datta, A. (2011). Formal verification of differential privacy for interactive systems. *Electronic Notes in Theoretical Computer Science*, 276:61–79.

[Turner et al., 2014] Turner, V., Gantz, J. F., Reinsel, D., and Minton, S. (2014). The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future.*

[Walport, 2016] Walport, M. (2016). Distributed ledger technology: beyond block chain. *UK Government Office for Science.*

[Wjst, 2010] Wjst, M. (2010). Caught you: threats to confidentiality due to the public release of large-scale genetic data sets. *BMC medical ethics*, 11(1):21.