Imperial College
London

InterCrypto Cryptocurrency Conversion Contract

Author:

Jack Tanner

jnt16@ic.ac.uk

Supervisor:

William Knottenbelt

wjk@doc.ic.ac.uk

Abstract

This project investigated mechanisms for on-blockchain cryptocurrency conversion. Current and in development smart contract and interoperable technologies were reviewed before a mechanism was chosen.

An Ethereum smart contract that uses a centralized oracle and a centralized exchange was developed and deployed to the Ethereum mainnet and testnet. Administration and recovery features were implemented as well as an interface contract that uses ENS to resolve the address of the InterCrypto contract. This report documents the design of the contract system, implementation details and challenges overcome in the developing and deploying the contract. This includes details of security vulnerability mitigation and gas optimization.

The InterCrypto service was evaluated to be of use to the current Ethereum ecosystem, and ready to use. The transaction cost of the smart contract and dependancies on centralized service providers Oracalize and Shapeshift present barriers for use and undesired failure methods. Planned Ethereum improvements and interledger networks will hopefully overcome these limitations in the future, making InterCrypto redundant at this time.

Acknowledgements

Contents

1 Introduction

The recent arrival of blockchain technology has made available new and unexplored ways to build software to facilitate economic and social systems. While the technology was only first conceived in 2008 with the release of the Bitcoin white paper[1] [1], it has gained significant momentum and the technology sector is now rapidly in need of tools to support blockchain software development to support this rapid growth.

Two very interesting areas of this technology are the diverse landscape of cryptocurrencies as well as the availability of smart contracts. The aim of this project is to develop a new on-blockchain mechanism that allows client smart contracts to execute cryptocurrency conversions.

1.1 Motivation and Objectives

Cryptocurrencies are the unit value of transfer used in blockchains. They are used as a feature to incentivise people to run the blockchain, which may only exist to support the cryptocurrency but may offer other services. Cryptocurrencies are transferable between users and provide a secure decentralized mechanism for exchange of value. While a lot of research is being done to create decentralized mechanisms to link blockchains together, it is currently not possible for smart contracts to send other cryptocurrencies to recipients on other blockchains. The ability to create transactions on other blockchains is of value to other smart contracts so that they can automatically pay out in different cryptocurrencies to the contracts users.

The primary objective of this project was to develop a new on-blockchain mechanism that would support a cryptocurrency conversion. Through this development process, other technologies that could assist this mechanism were to be investigated and compared. This report contains the following:

Section 1 Introduces the project's motivations, objectives and contributions.

Section 2 Explains how blockchain technology works by looking at Bitcoin and Ethereum.

Section 3 Reports the current state of the smart contract technology landscape.

Section 4 Reports the current state of in research, development and operational interoperable technologies.

Section 5 Explains how the developed cryptocurrency conversion mechanism, called the InterCrypto[2] service, works and the design rational.

Section 6 Details how the smart contract system was developed and tested.

Section 7 Explains key challenges overcome to develop and deploy the contract, including security vulnerability mitigation and gas optimization.

Section 8 Exhibits how the InterCrypto service is used through a client contract and explains the web application interface to interact with InterCrypto.

Section 9 Evaluates the success of the InterCrypto service's function, security and efficiency.

---

[1] https://bitcoin.org/
[2] https://intercrypto.org/

Section 10   Summarises the contributions of the project, suggests improvements to InterCrypto and speculates on the future of cryptocurrency conversions.

1.2   Contributions

During this project, an Ethereum[3] smart contract that uses a centralized oracle and a centralized exchange was developed. This smart contract provides a valuable commission-free service that can be used by other smart contracts to convert Ether to other cryptocurrencies. The following artefacts were produced as part of the InterCrypto service:

1. The Ethereum InterCrypto smart contract that converts Ether to other cryptocurrencies, deployed to the Ethereum mainnet and Rinkeby testnet.
2. The InterCrypto interface contract, that allows client contracts to easily connect to and use the InterCrypto smart contract.
3. The InterCrypto Wallet smart contract which exhibits an example use-case of the InterCrypto smart contract, and how to connect using the interface contract.
4. The InterCrypto web application that allow users to easily view and interact with the InterCrypto smart contract and Wallet. The web application can be found here with links to the InterCrypto source code and deployed contracts.

https://intercrypto.org

In addition to the development of the InterCrypto service, this project contributes valuable insights into smart contract development methods, programming challenges and security vulnerability mitigation techniques.

Two surveys were conducted to summarise the state of current and future generation of blockchain capabilities. These surveys provide a valuable central snapshot in August 2017 and had not been previously published:

1. Survey of developed smart contract technology.
2. Survey of interoperable technology, both in research and development phase.

Project milestones can be seen in Appendix E.

1.3   Terminology

The following terms are used throughout the report and are described here for the reader's reference:

Atomic - The action can happen completely or not at all.

Byzantine-fault Tolerance "BFT" - The ability of a network to tolerate a particular type of failures classified as part of "The Byzantine Generals Problem". Generally, byzantine failures are due to the nodes in the network's failure to reach an agreement required by the network, either due to technical failure or malicious behaviour.

---

[3] https://ethereum.org/

Consensus protocol / algorithm - The way in which the network comes to an agreement of the order of transactions in a blockchain ledger.

Cryptocurrency - The primary unit of value that can be transferred between blockchain users. In this report, it refers both to the native blockchain cryptocurrency that is used to reward miners and is the primary unit of transfer, as well as token systems built on top of blockchains such as Ethereum ERC 20 tokens.

Distributed Ledger Technology "DLT" - Is a ledger type data structure that is distributed over a network over several nodes. No central administrator or centralized data storage node exists. Blockchains are a subset of DLT in which the access to host or change the ledger is permissionless due to the Proof of Work consensus protocol.

Ledger - A data structure that stores a chronological set of transactions which can store arbitrary data. In the context of economics, a ledger stores a list of financial transactions.

Inter-blockchain - Refers to behaviour between two or more different blockchains such as Bitcoin and Ethereum.

Interledger - A ledger (blockchain or not) that acts as a bridge between two other ledger systems to facilitate interoperable behaviour.

Interoperable - Describes the ability of a system to communicate and interact with another system. In the context of this report, it is focused on the ability of a blockchain ecosystem of many different blockchains being able to interact with each other in a decentralized way.

Node - A computer running blockchain software. This could be a mining node that runs the full blockchain ledger or an extremely light node that only reads and executes a small amount of transactions.

Payment channels - Also called "state channels" or "micropayment channels", are a mechanism to allow for multiple secure transaction to be made without requiring that all the transactions are added to the blockchain ledger. The purpose of such channels is to reduce cost and increase the scalability of cryptocurrency transactions.

Turing complete - In the context of blockchain smart contracts, a turing complete smart contract environment is able to execute any programmed algorithm while non-turing complete algorithms cannot, usually due of not supporting programming loops.

2. Blockchain Technology

A blockchain is a distributed ledger that is used to maintain and update a growing list of transactions. The ledger is stored, updated and verified in a decentralized network of nodes (computers). The first successful blockchain application was released in 2009 called Bitcoin. This blockchain network stored a record of financial transactions of the world's first successful decentralized cryptocurrency [1].

Since this initial proof-of-concept, newer generations of blockchain technology have allowed a more flexible definition of what can be stored as well as providing a more complex set of features that the blockchain network can provide.

2.1   Blockchain Technical

Blockchain technology comes from the technical challenges of storing and updating information on a peer-to-peer (P2P) decentralized network in which peers do not know the identity of other nodes and thus do not necessarily trust one another to act honestly. The principal challenge is to allow the network to come to a consensus of the current "state" of the information, as well as incentivising participants to run the network. This section of the report will briefly explain how blockchain technology works to provide context for future sections of this report.

Bitcoin will be explained first as an introduction to blockchain, after which the generalized abstraction and smart contract capabilities that the Ethereum network provides will be explained.

2.1.1   Bitcoin Transactions

The Bitcoin blockchain is a ledger of financial transactions, stating simply the *to* and *from* Bitcoin addresses and an *amount* to transfer. This can be thought of similarly to a transaction stored by a bank in a centralized bank ledger. The difference in Bitcoin is that addresses are anonymous 26-35 alphanumeric characters [2] as opposed to bank numbers.

Users of the Bitcoin network generate a public-private keypair to access the network and the public key is written to the transaction to identify the user. For transactions to be valid and accepted, the transaction creator cryptographically signs the transaction with their private key. In this report, the alphanumeric keys of transaction addresses will often be hidden and instead shown as common names such as Alice and Bob, as seen in Figure 1 below:
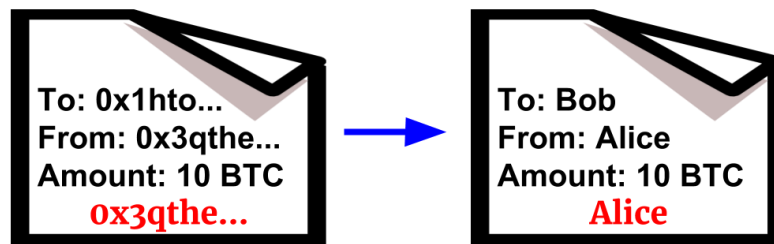
Figure 1: Signed Bitcoin transaction

After a transaction is created and signed by the author, it is then sent out to the Bitcoin network, a P2P network of nodes that run the ledger, using a gossip protocol [3]. The nodes in the Bitcoin network receive transactions and determine if they are valid by checking that the address has sufficient balance plus that the transaction signature matches the from address. If a transaction is valid the node will then broadcast it to other nodes around them.

2.1.2 Block Mining

Unlike a centralized server that runs a ledger (such as a bank), maintaining a ledger on a set of computers on a decentralized global network faces a serious problem unsolved until Bitcoin:
How do all of the computers reach a consensus on the order of the transactions that are added to the ledger?

The two issues to overcome in such a network are the lack of a universally synchronized time source, and the fact that propagation delays mean that transactions are received at different times by different nodes.

Bitcoin solves this problem by grouping transactions together into groups called blocks. Blocks of transactions submitted to the ledger at intervals as opposed to as they are received (real-time), by one or only a few nodes in the network (as opposed to all of them).

A process called mining is used to reach a consensus of which node is the one to submit the next block to the ledger. Each mining node in the network collects transactions into groups and adds them to a potential block. The data from all of the transactions is concatenated (appended one after the other) along with a random number generated by the mining node, called the nonce. This data is then hashed to produce a final hash. The mining node continues to try different nonces until the final hash has a certain number of the O digit at the beginning. The block is considered valid when this hash condition is solved, as well as all of the transactions are valid. Figure 2 shows a depiction of this process, where the hash of the invalid left block does not have any os at the start (invalid block) and the bottom left transaction signature does not match the from address (invalid transaction). The valid right block has found a nonce that will produce a hash with four Os at the start (four os have been chosen trivially in this case for ease of display) and has only valid transactions.
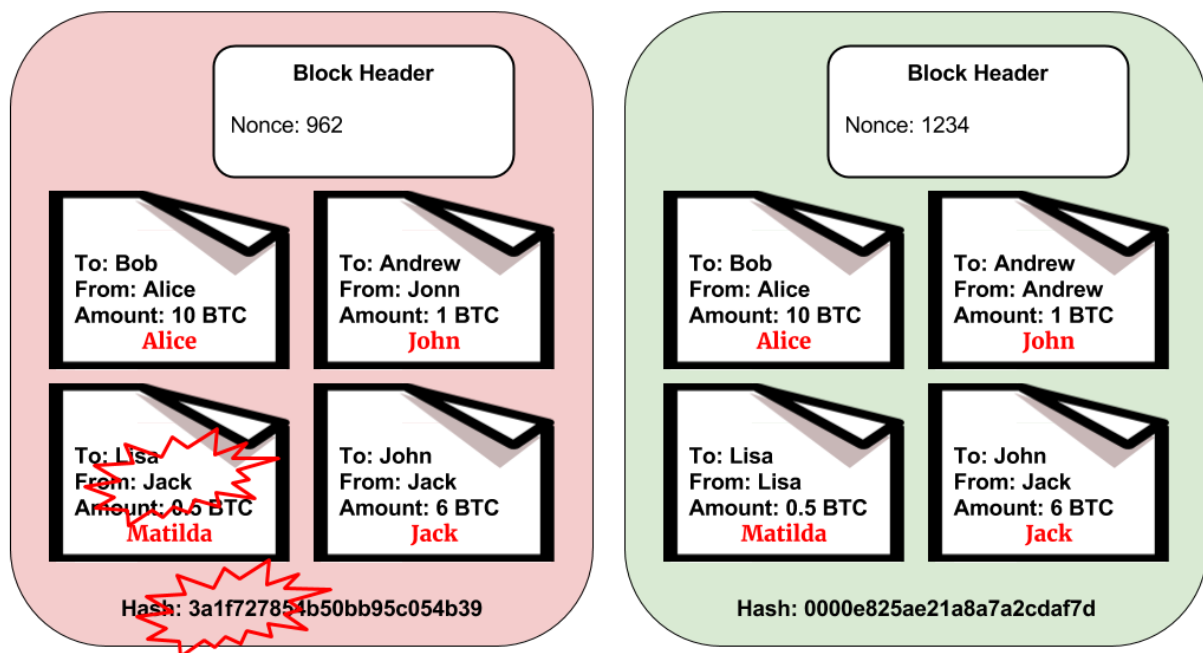
Figure 2: Invalid (left) and valid (right) blocks depend on how many 0s the resultant hash of the block has at the beginning.

The number of Os that need to be obtained in the hash is called the block difficulty, and varies automatically over time as more miners join the network and hardware capability increases. The difficulty is changed so that the network of nodes who are all trying to create a valid block will process one new block on average every 10 minutes. When the mining node successfully creates a valid block, it broadcast the block out to the network using a gossip protocol. Due to the asymmetric nature of hashing algorithms, other nodes can very quickly validate that the block hash has the correct number of Os, despite the fact that mining the correct block took much longer.

To summarise the "mining" process, nodes in the network create groups of transactions called blocks and then enter a race to solve a cryptographic problem that is hard to compute but easy to verify. Once one of the nodes generates a valid block they broadcast it to the rest of the network who checks its validity. It is important to note that not all nodes enter the mining race, some simply act as verifiers and relays for broadcasts of transactions and blocks.

2.1.3   Chains of Blocks

Each block stores the transaction data as well as a set of metadata called the block header. The nonce is part of this header along with the block number, a timestamp and the final hash of the previous block as seen in Figure 3. The mining process requires miners to hash all of this information with a correctly guessed nonce number to generate a valid block. Because the final hash of the previous block is part of the next block, this provides a unique pointer back to the previously mined block and not another mined block.
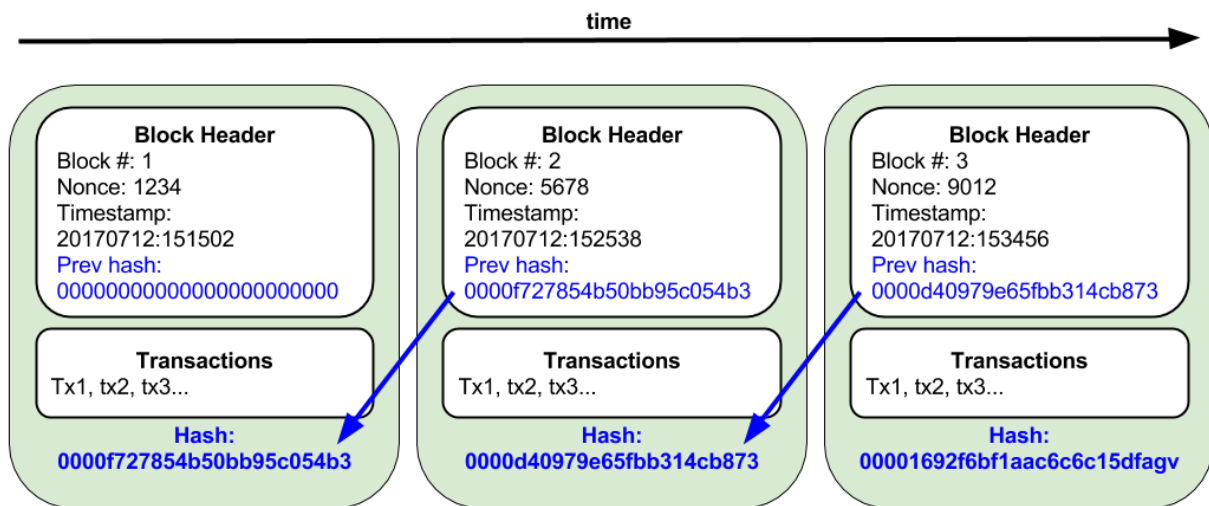
Figure 3: A chain of blocks where each block points to the hash of the previous block

The inclusion of the previous final hash is an important security feature in blockchains. Due to this feature, if the transaction data in a previous block is changed, the final hash of the next will be changed and will become invalid (as its final hash is now no longer correct based on the block's data). Not only will that block become invalid but all blocks after will then be invalid. This security feature prevents malicious miners from changing the blockchain as changing any part of a block will require that the modified block, plus all future blocks, need to be mined again in order to generate another valid blockchain. Due to the computation difficulty of mining, this creates a higher economic incentive for miners to mine the next block, than for them to try to change the prior blockchain data for their own personal gain.

2.1.4 Decentralization

The data structure of the blockchain can be summarized as a linked list of groups of transactions, linked together cryptographically through the hashes of the data in the block. This allows for the secure decentralization of this particular data structure.

As explained above in Section 2.1.3, changing of any data in a block will require all future blocks to be re-mined to create a new valid blockchain with different information. In a blockchain ledger hosted on a P2P network, in the case that a malicious miner does manage to re-mine all of these blocks then other peers are able to quickly check that the malicious chain has been changed simply by checking the hashes of the final block in the chain as seen in Figure 4.
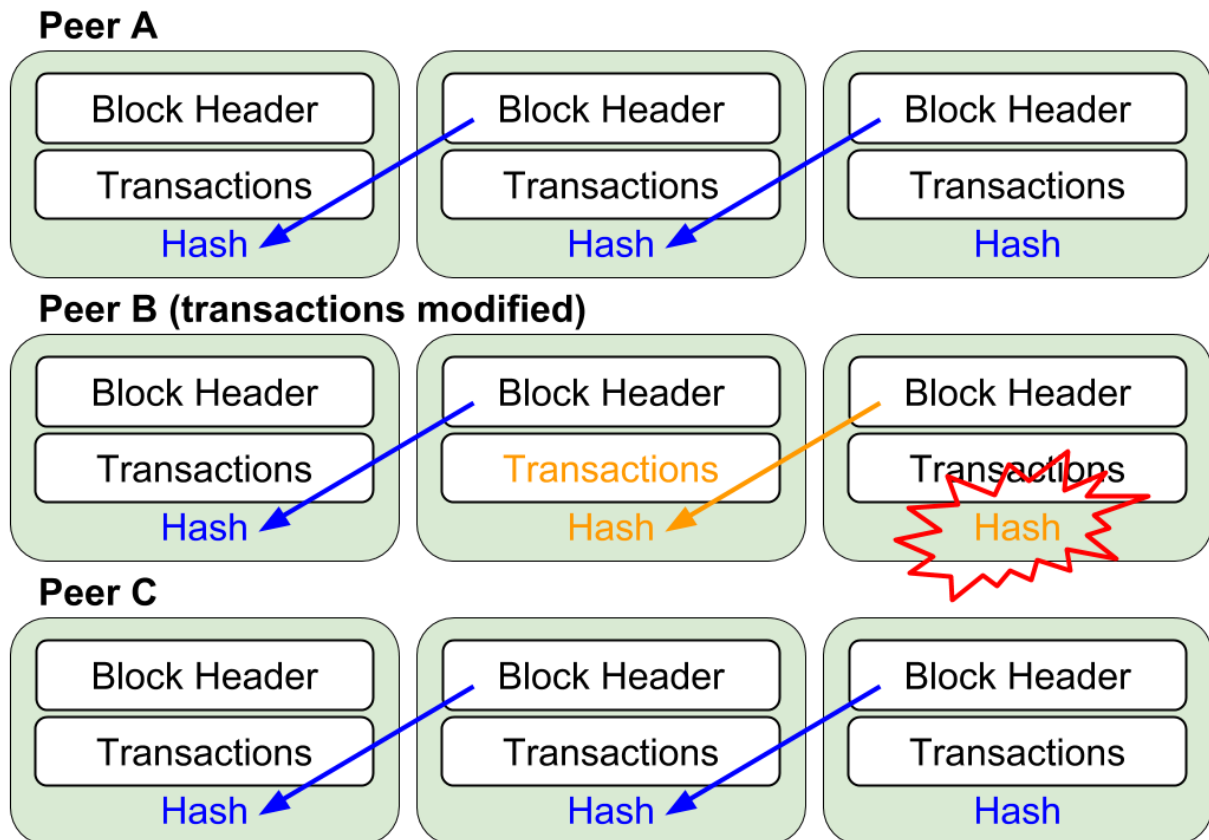
Figure 4: A peer-to-peer network of blockchains where peer B has modified the transactions of block two

In the case seen in Figure 4 above, the peers on the network come to consensus simply by using the most popular longest valid chain (in this case chain A and C). Through this mechanism, an attacking party needs to be able to mine the next valid block for as long as they need the malicious transaction to be considered valid (at least one block). To conduct this attack requires the miner to commit more computation power than at least 50% of the networks. To put this in perspective, such an attack would cost approximately £140 million per hour to overcome the Bitcoin network at the current Bitcoin hashing power, using modern mining hardware and at British electricity costs (see Appendix B.).

This very important property of transaction immutability allows Bitcoin users to get a statistical guarantee that when a transaction has been added to the blockchain, that it cannot be changed.

This chaining mechanism also allows the network to come to a consensus when two different miners solve a block at roughly the same time. In this situation, the two nodes broadcast their block to the rest of the network and other nodes will begin to work on the next block after. Consensus of the blockchain is reached again when the next block is found at which point the rest of the network switches to using the longest valid chain again.

2.1.5 Incentives

To generate currency, and to incentivize nodes to run the blockchain, a special transaction is included in each block called the coinbase transaction. This transaction creates new Bitcoins and the miner is able to send these coins to themselves as a reward for mining the block as seen in Figure 5. The amount of Bitcoin generated in this transaction started at 50 BTC when the Bitcoin network first started, and halves approximately every 4 years.
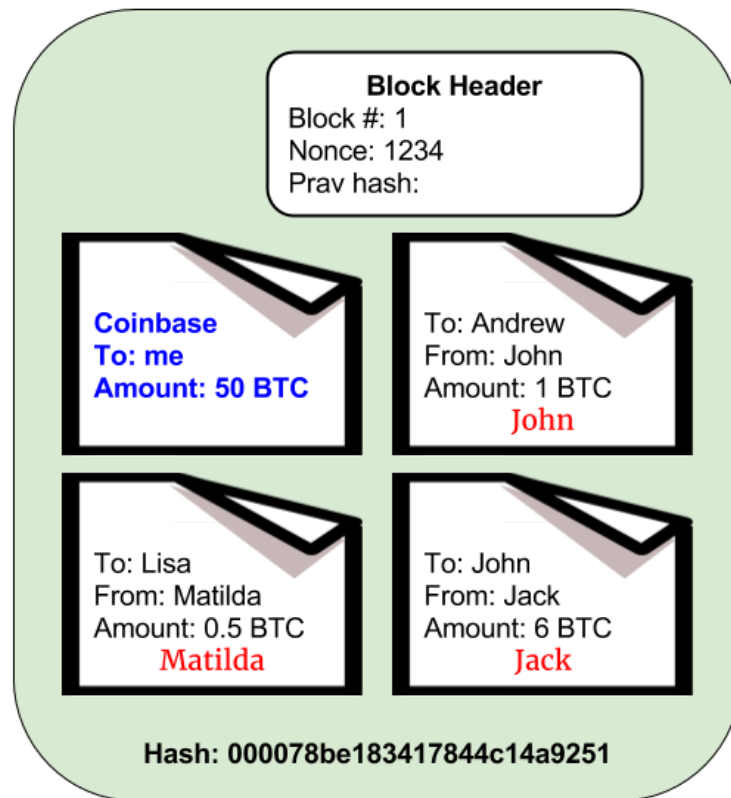

Figure 5: Block showing a coinbase transaction

In addition to the coinbase transaction, each individual transaction offers a fee, available to the miner. These fees act as the primary incentive for miners when coinbase incentives decrease.

2.1.6 Ethereum and Smart Contracts

The Ethereum blockchain extended the concept of Bitcoin by making the distinction between the Bitcoin cryptocurrency and the Bitcoin network. Ethereum provides a mechanism, called smart contracts, that allow users to easily deploy a program of arbitrary logic to the blockchain that can later be executed. In this way, the Ethereum blockchain provides a platform that allows users to create a Bitcoin token system, amongst many other interesting and yet-to-be discovered contract systems. Figure 6 shows the analogy of how the Ethereum platform abstracts the Bitcoin and other blockchains to create an interoperable internal blockchain system.
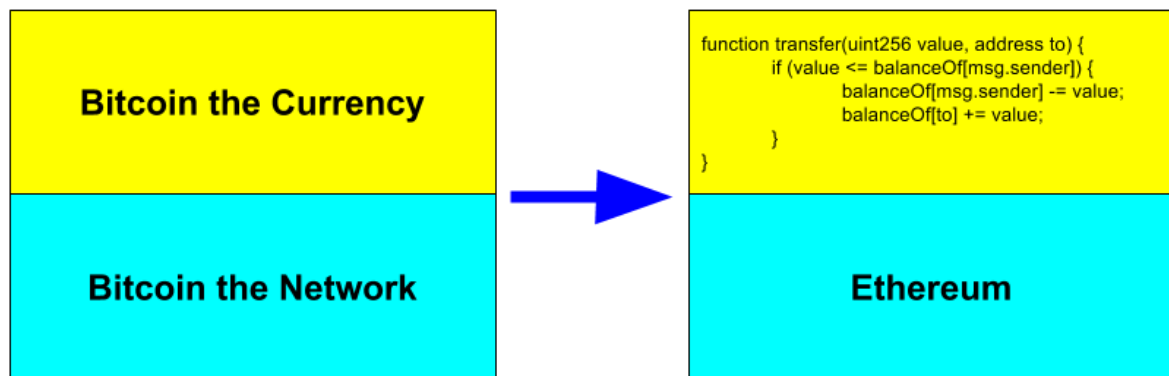
Figure 6: Ethereum evolution from Bitcoin

To enable this, all Ethereum nodes run the Ethereum virtual machine (EVM) that runs a turing complete set of opcodes [4] which can execute arbitrary programs. Users are able to write programs that will be compiled to the EVM bytecode.

To deploy a smart contract, this bytecode is sent in a transaction to an unused address, which the network will process and store in the blockchain ledger like any other transaction, as seen in Figure 7. This address is similar to Bitcoin except that no private key exists to control the address, whose behaviour is instead controlled by the program code that was deployed there. This address can then be later referenced by other contracts or addresses using the contracts Application Binary Interface (ABI) to call the functions that the smart contract exposes. Contracts are analogous to objects in Object Oriented Programming, they have a state and a set of functions that can modify the state.
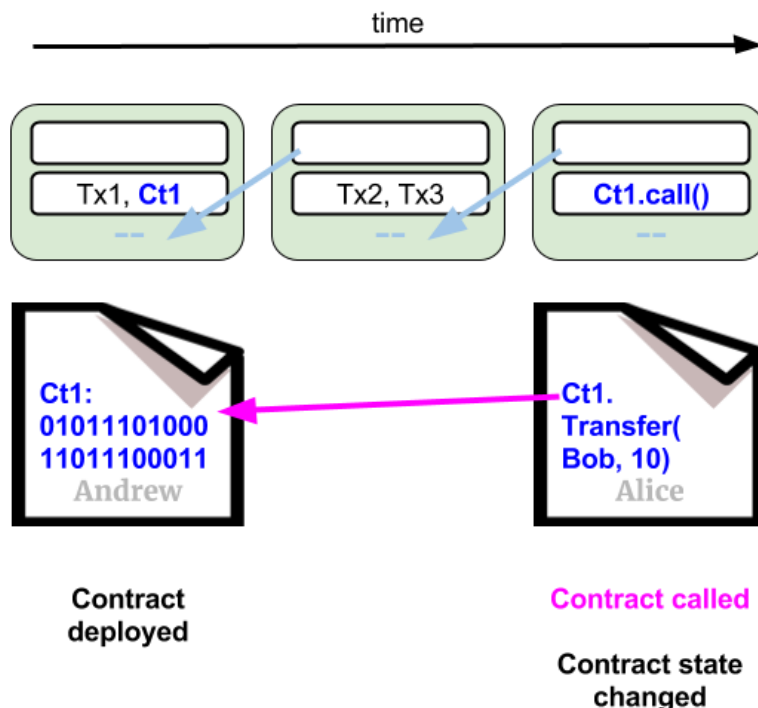

Figure 7: Contract deployment and calling mechanism

Figure 7 shows a visualization of a transaction in which a contract "Ct1" is deployed to the blockchain and a later transaction references that contract in a function call. As in Bitcoin, a transaction is executed by every node in the network to verify that the correct state change occurred. This requirement leads to important limitations to the determinism of the EVM language and are described in more detail in Section 5.2.

2.2 Testnets

Networks that run a blockchain in which all users pre-agree that the native cryptocurrency has no value are called testnets. One or more testnets usually exist for each blockchain to allow development of new infrastructure in an environment that does not have real world costs.

2.3 Blockchain Characteristics

Important characteristics of a blockchains are:
- Information stored on the blockchain cannot be changed after it has been stored.
- A transaction is an atomic action - it is either processed completely or not at all.
- No single point of failure exists in the network.
- Nodes (computers or devices) are able to join and leave the network at will, without affecting the availability of the blockchain to others.
- Transactions are added to the blockchain by one miner, but are verified by all nodes on the network.

2.3.1 Permissioned vs Permissionless

Bitcoin, Ethereum and most current blockchain technologies are permissionless such that the ledger can be accessed and updated by any node. Some blockchain technologies have finer privacy functionality and are referred to as permissioned blockchains. In permissioned blockchains, new nodes (both miners and non-miners) may need to be approved by one or more other nodes before they can participate in the network. Permissioned and permissionless blockchain technologies have several important differences that can be seen in Table 1 [5] [6].

	Permissionless	Permissioned
Access	Creation of accounts is unrestricted.	Creation of accounts is managed by those who have permission to grant accounts.
Visibility	Nodes can verify and audit all transactions.	Nodes can verify and audit transactions for which they have permissions.
Governance	The blockchain network is not controlled in a decentralized way.	The blockchain network is controlled by permissions that may or may not be granted in a decentralized way.
Attacks	The blockchain network can only be attacked by someone who controls more than 50% of the processing power of the entire network.	The blockchain network can only be attacked by someone who is granted full ledger permissions and has more than 50% of the processing power of the network that has permissions to maintain the full ledger.
Efficiency	The computational efficiency is low due to the need of a low-trust consensus mechanism such as Proof of Work.	The computational efficiency can be much higher due to a trust based consensus mechanism such as Proof of Authority.
Crypto currency	Incentive in the form of cryptocurrency tokens must be given to mining nodes.	Due to different consensus models, it is uncommon for permissioned blockchains to have a default cryptocurrency.

Table 1: Characteristics of permissionless vs permissioned blockchains

2.3.2 Public, Consortium and Private Blockchains

Using permissioned blockchains allows the for the distinction of public blockchains, in which anybody be can create and run a node, and private or consortium blockchains. In a private blockchain only authorized used can create and run a node and a consortium blockchain is a hybrid of the two. Blockchain technology currently allows for projects to choose what type of blockchain is suitable with trade-offs in the anonymity, immutability, transparency and efficiency characteristics [7]. As such, enterperprise projects are currently favouring private and controlled blockchains and global projects are favouring public blockchains.

2.4 Consensus Protocols

To reach an agreement on the current state of the blockchain ledger, the blockchain nodes use a consensus protocol to determine who will be the next node to update the ledger. In permissionless blockchains, where nodes do not inherently trust each other, a Proof of Work consensus protocol is used as described in Section 2.1.2.

Permissioned blockchains work differently. Because mining nodes in the network are granted, the identity of these nodes is known by at least the permission granter. In this case, nodes have built in trust and alternate consensus protocols can be used to reach the agreement of the current state of the blockchain. These consensus protocols do not require large computational work to be done to solve a puzzle and as

such do not provide a token based cryptocurrency incentive to miners [8] [9]. A full survey of the different consensus protocols used in blockchain technologies was not within the scope of this project.

3. Smart Contract Technology

Smart contracts are programs stored on a ledger and executed by the network. The first generation of smart contract offered by Bitcoin are not turing complete and as such offer limited capabilities. Second generation blockchain technologies now offer richer sets of execution environments that allow for greater functionality.

Smart contracts have several characteristics that are different from programs that are not executed on a blockchain network [10]:

- Like all blockchain transactions, the program of a smart contract (a piece of data representing the execution instruction set) cannot be changed after it has been added to the blockchain in a transaction (deployment).
- They are self-executing by the blockchain network and deterministic.
- The execution of the contract is verifiable by all nodes at any time (which is why they have to be deterministic).
- They cannot access data that is not on the blockchain which hosts them. This limitation is explained further in Section 5.2.

Several prominent smart contract capable blockchain technologies were investigated in this report. A summary of the characteristics and smart contract capabilities can be seen in the below sections [11]. Figure 8 shows the landscape of prominent smart contract technologies in August 2017.
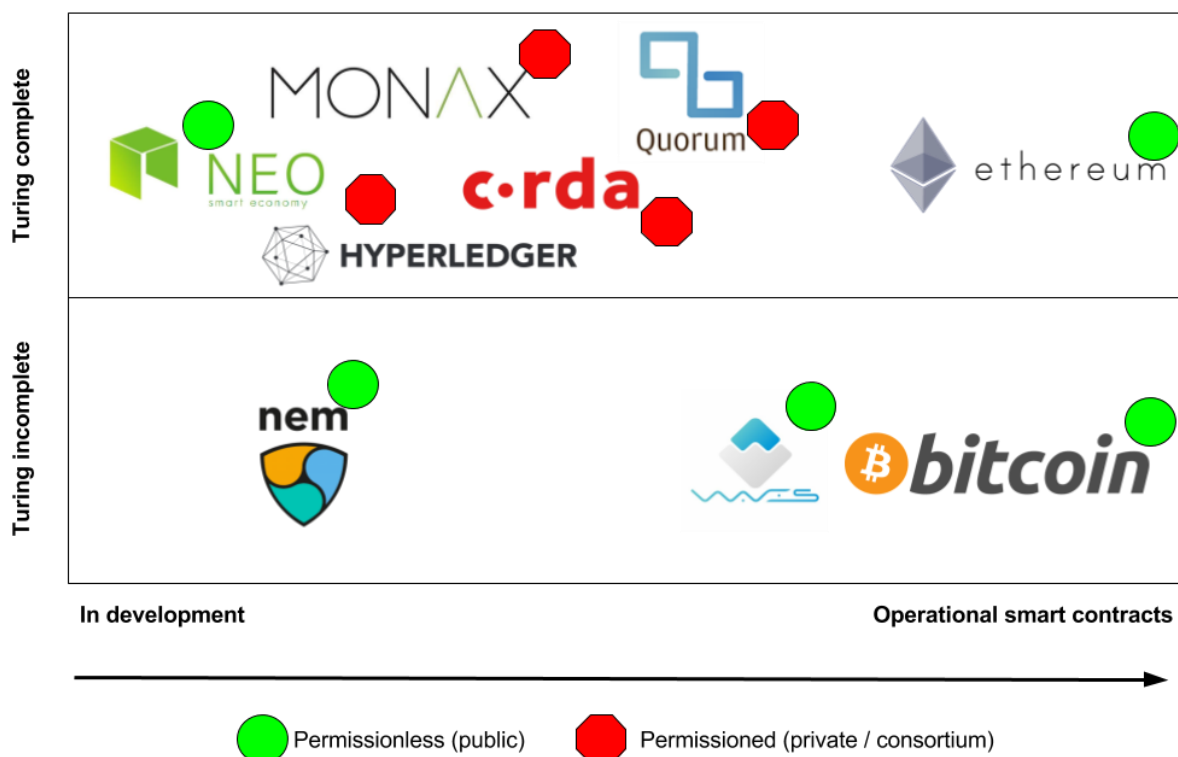

Figure 8: Different smart contract blockchain technologies

14

All of the projects are open source and are under active development. As open source projects, networks running the different ledger platforms can be deployed by anyone, and these can be given whatever permissions are available through that platform. All projects did have at least one testnet and mainnet (planned or operational), however, additional private networks could exist that were not identified in this report.

There were a few common differences between the permissioned and permissionless smart contract technologies:

- The permissionless networks implement platforms that use distributed ledger technology (of which blockchain is a subset). Depending on the platform, deployment of a network may create a blockchain network or a more general distributed ledger platform.
- None of the permissionless networks had a native cryptocurrency, as explained in Section 2.3.1.

To get a better understanding of the level of development in these technologies, the recent and total GitHub activity was compared in Table 2. This information was obtained on September 5 from the primary node repository of the projects. The numbers numbers should not be interpreted to rank project by level of development, but merely to get a rough idea of level of development.

	Contributors	Commits (4 weeks)	Commits (total)
Bitcoin	462	70	14,751
Ethereum	159	49	8,854
Quorum	110	7	7,881
Fabric	92	3	4,209
Corda	50	200	4,229
Waves	27	49	4,973
Monax	18	0	1,918
NEO	9	19	298
Rootstock	7	0	135
NEM	3	0	631

Table 2: Git commits and contributors for different smart contract platforms

3.1 Bitcoin

Bitcoin was the first successfully operating blockchain technology. Since its initial release in 2009 it has grown to become the most used blockchain with the highest confirmed transactions per day in July 2017 [12] [13]. The Bitcoin ledger was built primarily to support a ledger of Bitcoin cryptocurrency transfers (e.g. Bob sends Alice 10 Bitcoins).

Smart Contract Capabilities:
The Bitcoin network protocol supports a scripting language that allows users to change the conditions under which a Bitcoin transfer is valid (e.g. requires multiple signatures, or no signatures). It supports a stack-based non-turing complete language that does not support loops [14]. Many of the original features

of the language have since been disabled and Bitcoin lacks the ability to extend its smart contact capabilities due to security challenges [14] and governance issues. A large variety of Bitcoin scripts were in use in August 2017.

3.2   Corda

Corda[4] is a decentralized ledger platform currently in development. The platform is aimed at serving the financial sector but is applicable to other areas. Permissions for the R3 test network and planned r3net are granted by R3, the founding and developing organization [15]. R3 is a consortium of many global financial organization.

Smart Contract Capabilities:
Corda runs a state transition machine similar in principle to Ethereum that allows actors to record and manage agreements between themselves. Unlike Ethereum, contracts and the states that they manage are private between the actors while retaining the other properties of immutability, security and availability. As of August 2017, these were examples of Corda smart contracts being used for production trials but not in business operations [16].

Contracts are composed of two parts: the contract code written in unrestricted Java, and a legal clause to impose legal significance to the meaning of the contract code. Contracts are run on a sandboxed Java Virtual Machine (JVM) by all permissioned peers. Contracts can be written in any JVM language such as Java or Kotlin. While the full JVM language is available for writing, the Corda JVM sandbox runs in a deterministic way by executing only deterministic functions. The deterministic limitations reflect the same determinism issues as in Ethereum (discussed in Section 5.2).

3.3   Ethereum

Ethereum is a decentralized blockchain platform that runs general-purpose smart contracts [17]. This blockchain technology addresses some of the long-standing issues with Bitcoin by decreasing the block time from 10 minutes to 15 seconds, using a flexible block size (compared to Bitcoin's 1 Mb maximum block size) while adding new features that would allow for the deployment of smart contracts.

Several testnet networks, as well as several privately deployed networks are in use. These networks use a variety of different consensus protocols. The Ethereum mainnet is the public production network in which Ether has a real-world value. The Ethereum mainnet protocol is permissionless and the mining network is incentivised by the Ether cryptocurrency. Unless otherwise stated, reference to Ethereum in this report will always refer to the Ethereum mainnet.

---

[4] https://www.corda.net/

Smart Contract Capabilities:

The execution environment provided by the Ethereum network, also called the Ethereum virtual machine (EVM), provides a turing complete language allowing for arbitrary logic programs. The principal development language, Solidity, is a high-level language similar to a mixture of JavaScript and C. Solidity supports object oriented features such as inheritance and polymorphism [18]. High-level languages such as Solidity are compiled to EVM bytecode which can be deployed to the blockchain as self-executing programs that can be called by Ethereum account holders [19].

The turing complete nature of the EVM makes it possible to build token systems such as Bitcoin and many other Bitcoin sidechain and derivatives as well as develop standalone decentralized apps (DAPPs) with arbitrary functionality. In August 2017, large amounts of new smart contract applications were being deployed on a monthly basis.

3.4 Fabric

Fabric[5] is a blockchain framework that allows for easy modular deployment of blockchains. Developed through Linux's Hyperledger project, Fabric allows for selection of different components such as consensus or privacy models as needed to create the desired public or private distributed ledger or blockchain to fit the application [20].

Smart Contract Capabilities:

The Fabric smart contracting module provides contracting capabilities called chaincodes that can be written in Go, with plans to allow support for Java in the near future [21]. Unlike Ethereum, chaincodes are private to themselves only, but can be permissioned explicitly to interact with other chaincodes.

3.5 Monax

Eris Industries launched the Monax[6] (formerly called Eris) blockchain, the first permissioned blockchain platform, forking it from Ethereum. The blockchain is targeted for use by banks and has several features that provide additional services for private companies. The principal addition is an access control layer which controls the permission and governance of the network.

Smart Contract Capabilities:

In Monax, due to its relationship with Ethereum, smart code is written in any EVM language such as Solidity and has the same contract capabilities as Ethereum, which are discussed in more detail in Section 5.2.

---

[5] https://www.hyperledger.org/projects/fabric
[6] https://monax.io/

3.6 NEM

NEM[7] is a permissionless blockchain platform that claims to have improvements and add additional features on top of the Bitcoin protocol. Using a Proof of Importance consensus algorithm, the platform claims to have drastically lower costs, as well as having inbuilt features such as namespaces (an internal DNS-like naming system), multi-signature contracts and enhanced security. In addition to the traditional node / miner types of Bitcoin-like blockchains, NEM uses "harvester" and "supernodes" as part of its governance model. The NEM cryptocurrency is used to incentivise nodes (miners, harvesters and supernodes) to run the network [22].

Smart Contract Capabilities:
NEM supports scripting that controls account access such as creating multi-signature accounts. While these capabilities are faster and easier to use than the scripting language in Bitcoin, they are also more limited.

3.7 NEO

The NEO blockchain[8] (recently rebranded from Antshares) is China's first open source public blockchain. It claims to support cross-chain assets swap and distributed transaction protocols, quantum resistant cryptographic, parallelism through sharding and unlimited scalability [23]. The NEO blockchain is run using a Byzantine fault tolerant consensus algorithm and the NEO cryptocurrency incentivises miners to run the network.

Smart Contract Capabilities:
NEO nodes run a stack-based turing complete virtual machine and smart contract development is supported in most high-level languages, with initial support for C#, VB.Net, F#, Java, and Kotlin [24].

3.8 Quorum

Quorum[9] is a consortium distributed ledger platform, forked from the official Ethereum software [25]. The Quorum software adds additional permissioning, consensus, privacy and performance features, while staying updated with major Ethereum software changes. The development of the software is led by leading UK financial institution J.P. Morgan. Support for the Quorum project as part of external framworks and projects was evident in August 2017 [26] [27].

Smart Contract Capabilities:
Due to its relationship with Ethereum, smart code in Quorum is written in any EVM language such as Solidity[10] and has the same contract capabilities as Ethereum, which are discussed in detail in Section 5.2.

---

[7] https://nem.io/
[8] https://neo.org/
[9] https://www.jpmorgan.com/quorum
[10] https://solidity.readthedocs.io/en/develop/

Unlike Ethereum, contract state and function access are controlled by the contract deployer [28]. No production smart contract could not be found in August 2017.

3.9 Rootstock Bitcoin Merge-Mining (RSK)

The Rootstock[11] project aims to extend the limited capability of the Bitcoin network by adding a more complex set of features through a process called merge-mining. The project states that it's goal is to enable smart contracts near instant payments and higher scalability compared to the robust yet limited Bitcoin network [29]. The process of merge-mining rewards miners running the Rootstock network in the Bitcoin cryptocurrency. Rootstock is a sidechain of Bitcoin and does not require any changes to the Bitcoin protocol (see Section 4.1.1 for more details on sidechains).

Smart Contract Capabilities:
Detailed technical documentation was not available in August 2017 and as such the smart contract capabilities are unknown. The Rootstock white paper alludes that the capabilities will be turing complete and comparable with the Ethereum smart contract capabilities [30].

3.10 Waves

The Waves platform[12] is a high throughput blockchain aimed at creating easy and safe to build token economies. The Waves platform has an inbuilt decentralized exchanged of Wave asset tokens which is bridged through a centralized exchange to the Bitcoin cryptocurrency [31]. The Waves cryptocurrency is used to incentivise miners to run the network.

Smart Contract Capabilities:
The non-turing complete contracting capabilities allow for the easy and safer developments of tokens however does not allow for arbitrary logic contracts [32]. Several token systems had already been developed with Initial Coin Offerings (ICOs) however, in August 2017, there was no evidence that these tokens were in operation for their final intended purpose.

---

[11] http://www.rsk.co/
[12] https://wavesplatform.com/

4. Interoperable Technology

Several technologies were investigated that offer different services that support interoperable blockchain capabilities that can be used for cryptocurrency conversion. The goal of interoperable technologies is to overcome the inherent property that a blockchain is a closed internal system that cannot access or interact with information not on that blockchain.

Table 3 shows a summary of these technologies, how mature the technology is with respect to its interoperable service, how well it solves interoperability and its governance model.

	Maturity / Popularity	Governance model	Interoperability
Pegged Sidechain	Medium	Decentralized	One-to-one
Centralized notary	High	Centralized	High
Ad-hoc notary	High	Centralized	High Interoperability very difficult
Decentralized notary	Medium	Decentralized	High Interoperability very difficult
Relay contracts	Medium	Decentralized	One-to-one
Hash-locking	Medium	Decentralized	One-to-one
Interledger Networks	Low	Decentralized	High
ERC20	High	Decentralized	Within Ethereum ecosystem only
Centralized exchange	High	Centralized	High
Decentralized exchange	Medium	Decentralized	Within Ethereum ecosystem only

Table 3: Summary of interoperable technologies

4.1 Interoperable Technologies Currently In-Use

Several operational interoperable technologies exist that are already in use [33].

4.1.1 Pegged Sidechains

A pegged sidechain allows for low-trust bidirectional cryptocurrency conversion from a parent blockchain to a sidechain and back. As the most development in sidechains is occurring in Bitcoin with the Blockstream[13] and Lightning Network[14] projects, it shall be referred to as the parent chain.

---

[13] https://blockstream.com/
[14] https://lightning.network/

A Bitcoin is sent to the desired sidechain network by sending it to a specific address on the Bitcoin network. At this address the Bitcoins are locked until a specific release condition is met. At this point, the Bitcoin owner can provide proof that this transaction occurred to the sidechain, which will trigger the creation of an equal amount of coin tokens on that sidechain. The owner of the new sidechain coins is the same as in Bitcoin, controlled by the same public-private keypair. The sidechain can then implement extensions to Bitcoin such as new opcodes, confidential transactions or new crypto-assets [34]. The coins can be sent back to Bitcoin by sending them to a specific address on the sidechain network which will lock the sidechain coins. Upon proving this, the equivalent number of Bitcoins will be unlocked on the Bitcoin network from the original locked address [35].

While this mechanism provides a low-trust way to convert coins across two blockchains, the sidechain network must be pre-configured to be a sidechain of Bitcoin. The Segregated Witness (Segwit) soft fork in the parent chain is not required but greatly assists the efficiency and reduces the complexity of configuring the parent peg to a sidechain. The Segwit fork has not been implemented Bitcoin yet, but has been in several popular Bitcoin derived blockchains such as Litecoin[15].

4.1.2   Notaries

The technologically simplest and currently most popular way to facilitate inter-blockchain transaction is through a centralized agent or group of agents that are trusted to be intermediaries between different blockchains.

In such a mechanism, the agent is trusted to make a claim to blockchain A that an event on blockchain B took places. Oracles, as discussed in Section 4.5, are a form of inter-blockchain notaries.

4.1.3   Relays

A relay is a smart contract that lives on one chain that authenticates transactions on another chain by validating block header information. Block headers are provided by a decentralized set of sources "relays".

The smart contract maintains a mini-version of the other blockchain ledger and the transactions are validated cryptographically. Transactions are validated using the protocol of the other chain, granting a reward to the relay for providing the block header information. The mechanism is low-trust as incorrect relay data is automatically discarded. The solution is scalable and a working contract, set up as the BTC Relay[16] contract in Ethereum can currently be used to validate transactions in Bitcoin. The BTC Relay is limited due to the Bitcoin scripting language to being a one-way information source (Ethereum can learn about Bitcoin only) [36]. Two-way information streams are only possible if both blockchain technologies have appropriate smart contracting capabilities.

---

[15] https://litecoin.com/
[16] http://btcrelay.org/

4.1.4 Hash-locking

Hash-locking is a mechanism in which cross chain atomic transactions (CCATs) can be achieved. The on-blockchain low-trust process is conducted over several time sensative transactions in which parties lock funds in a contract or scripted transaction. The funds can then be unlocked on the timely provision of certain hashed data from the two parties. Multisignatures, hashing and time-locking functionality must be present in both blockchains for hash-locking to be possible, making it a viable solution for all current popular blockchains including Bitcoin [33].

This mechanism is reversible but requires both parties to act in a timely manner. Hash locking contracts need to be created on a blockchain-pair basis. For example, the Ethereum-Bitcoin hash-lock contract pair would not be valid to use for an Ethereum-NEO cryptocurrency conversion.

4.2 ERC20 Standard

Within the Ethereum network, in which tokens systems such as Bitcoins can be created as a smart contract, the ERC20 standard [37] provides a standardized smart contract interface to facilitate the easy exchange of these tokens on internal markets. An outcome of the standard is the ability to created fully decentralized token exchanges. In August 2017, hundreds of token contracts had been deployed to the Ethereum blockchain with new token systems being deployed frequently.

This standard provides an efficient method for accessing and communicating between tokens within the Ethereum network, however does not provide any mechanism that functions between blockchains.

4.3 Blockchain Assets Exchanges

Due to the growing demands of different cryptocurrencies and blockchain tokens (namely ERC20 standard Ethereum tokens), many projects exist that allow cryptocurrency and token management and exchange.

4.3.1 Centralized Exchanges

Early entrants to the exchange space have been traditional companies, with a centralized server architecture to manage and trade between its particular users in a safe way. Large exchanges trade in several fiat currencies and more than a dozen cryptocurrencies. As with any financial trading, to remain Know Your Client and Anti-Money Laundering regulation compliant, all popular global exchanges and most other exchanges require users to undergo stringent identification checks to use their service.

As the trading space has even further evolved, trading directly between currencies (such as Ether from Bitcoin) has become common. By building on top of other exchanges and not handling any fiat currency of

users, the ShapeShift[17] and Changelly[18] platforms have been identified that allow for crypto-crypto conversions without the stringent identification checks.

4.3.2 Decentralized Exchanges

To overcome the issue of trust required in using a centrally managed exchange such as the ones mentioned in Section 4.3.1, demand has grown for the use of decentralized exchanges. This model does not depend on the reputation of the service provider and instead on the underlying decentralized technology that the platform uses. The decentralized exchange model space is still very immature and several recently operational in development projects were investigated to gain further understanding about interoperable capabilities.

Thus far, two operational exchanges, EtherDelta[19] and oxProject[20] exist that provide a platform for exchanging ERC20 tokens in a decentralized way. Both projects implement Ethereum smart contracts [38] which are supported by payment channels to reduce trade management costs. Payment channels are an off-blockchain mechanism that allow for higher scalability and lower transaction costs, but their current application is limited. While the decentralized exchanges function correctly, both projects have yet to reach the high market liquidity found on popular centralized exchanges.

The operational ox "oxProject" exchange [39] and the in-development SWAP[21] [40] project both use or propose open-source exchange protocols that can be used by others to increase the liquidity in the decentralized exchange market.

The under development OpenAnex[22] project [41] proposes to build a decentralized exchange facilitating ERC20 token exchange as well as cryptocurrency and fiat currency exchange. The not-for-profit organization, which is run on a transparent Decentralized Autonomous Organization (DAO) contract, proposes to build the exchange infrastructure on top of Ethereum's primary payment channel mechanism, the Raiden network, the ERC20 standard and the ox and SWAP protocols. It also proposes to offer decentralized dispute resolution, run through their DAO contract. The final objectives are to create a decentralized exchange market with much higher liquidity than previous projects.

The Bitsquare application is a stand-alone application, available on several operating systems, that facilitates a P2P network of traders. The application allows for exchange between several cryptocurrencies and fiat currencies. The application traffic runs through the Tor[23] network using the onion protocol to hide

---

[17] https://shapeshift.io/
[18] https://changelly.com/
[19] https://etherdelta.github.io/
[20] https://oxproject.com/otc
[21] https://swap.tech/
[22] https://www.openanx.org/
[23] https://www.torproject.org/

user IP addresses and protect user privacy. The application is in beta phase with over 100 cryptocurrencies supported and limitations on user trading volumes.

4.4   Interledger Networks

There are currently significant amounts of research being applied to the idea of creating an inter-blockchain network that acts as a bridge between all other blockchains. Unlike the solutions discussed previously, the objectives of such a network would be to provide a *generalized and decentralized* platform for on-blockchain communication between blockchains.

Several prominent projects with the objective of creating such a network were surveyed and are discussed below. In general, the solutions share several common objectives:

- Facilitate a decentralized mechanism of communication between blockchains with no single point of failure.
- Allow for cross-chain atomic transfers (CCATs) between blockchains such that a transaction occurs on both blockchains, or neither.
- Maintain the security properties of each individual blockchain.

Each solution described below proposes a mechanism in which two or more blockchains are connected through a network of facilitator nodes that act as relays. Most propose using a ledger that records inter-blockchain interactions.

All of the reviewed material gave a strong impression of the high level of difficulty of the creating a general network to connect the diverse ecosystem of blockchains, each with different purposes, protocols, consensus mechanisms, privacy and governance models. Table 4 shows a summary of the maturity and the level of interoperability that the surveyed solutions provide.

	Maturity	Bridge network	Connects between
Interledger Protocol (ILP)	In development and testing stage	ILP with ad-hoc notaries (in use). Ripple blockchain (live) as intermediator ledger that uses ILP.	Public and private blockchains and traditional financial ledgers such as stocks and banks
Polkadot	Research stage	Parity Polkadot Platform (not released)	A variety (and undefined) of data structures including public and private blockchains including inter-smart contract communication
Cosmos	Development stage	Cosmos Hub (not released)	Interledger networks and blockchains
Strong Federations	Research stage	Liquid (not released)	Parent chain to sidechain blockchains only
COMIT	Research stage	COMIT (not released)	Blockchains with payment channels and traditional banking systems

Table 4: Maturity and support provided by current interledger network projects

Payment channels or a similar mechanism play an important role in these solutions. As explained in Section 4.3.2, payment channels in the context of a blockchain are a mechanism set up to facilitate off-blockchain transactions with the objective of increasing scalability and reducing costs. Hash-locking as discussed in Section 4.1.4 is also proposed as a primary feature to facilitate trust-less CCATs. This creates an important limitation in which the CCAT initiator needs to stay available throughout the process of the transaction.

To get a better understanding of the level of development in the interledger networks surveyed, the recent and total GitHub activity was compared in Table 5. This information was obtained on September 5 from the primary node repository of the projects. The numbers numbers should not be interpreted to rank project by level of development, but merely to get a rough idea of level of development.

	Contributors	Commits (4 weeks)	Commits (total)
Interledger Protocol	14	6	355
Polkadot	0	0	0
Cosmos	9	1	489
Strong Federations	0	0	0
Comit	0	0	0

Table 5: GitHub activity of interledger networks

4.4.1 Interledger Protocol

The Interledger Protocol[24] (ILP) [42] is a protocol specification to facilitate generalized interledger transfers. The protocol covers both blockchain ledgers as well as any other type of ledger system including private or consortium blockchains, bank ledgers and stock ledgers.

The protocol operates on two modes of operation. The first "atomic" mode requires users of the connected ledgers (such as Bitcoin and Ethereum, or Ripple and Lloyds bank) to transfer funds to an escrow account in the first phase of a two-phase atomic process. An ad-hoc group of notaries then execute the transfer transaction on both ledgers in a low-trust way, or abort the transaction. The notaries are compelled by a Byzantine Fault Tolerant consensus algorithm to reach an agreement on the outcome of the transaction.

In the "universal" mode the incentives of rational participants are relied on to remove the need for external coordination with a group of notaries. This is done by executing the transfer in a specific order and way to align all participants incentives such that all execute the transfer correctly.

ILP does not require the use of a blockchain or any ledger between the connected ledgers. However, the protocol is currently being trailed with the Ripple blockchain[25] as an intermediary between other ledgers, including banking ledgers. In this instance, the Ripple blockchain is providing an extra layer of liquidity that the pure ILP cannot provide [43]. To be clear, the Ripple blockchain is not an implementation of the ILP itself but is a blockchain aimed at connecting other blockchains and financial systems using the ILP. The Ripple blockchain is currently operating a centralized model to govern the blockchain mining, which has created controversial opinions by the public as to its use of a blockchain [44] [45].

4.4.2 Polkadot

Polkadot[26] is a research proposal for an inter-blockchain protocol. The project suggests a relay mechanism that allows for communication from one blockchain to another. This is done through an intermediary blockchain via a Proof of Stake consensus algorithm. A token system allows Polkadot actors to participate in the relay mechanism through a system of voting with appropriate incentives and punishments. Four different types of actors are identified that play different roles in the relay mechanism [46].

The protocol alludes to the ability to facilitate CCATs but also allows for a more general swap of data. This would be a more general solution than ILP to allow for communication between different smart contract systems on different blockchains. The Polkadot protocol is also generalized such that a large variety of data structures could be supported, including but not limited to blockchain and ledger structures. It wasn't clear as to the exact extent of this interoperability scope and the white paper mostly focuses on public blockchain examples.

---

[24] https://interledger.org/
[25] https://ripple.com/
[26] https://polkadot.io/

4.4.3 Cosmos Network

The Cosmos network[27] is developing a protocol to facilitate inter-blockchain communication, using an intermediate blockchain [47]. The intermediary blockchain uses a Proof of Stake consensus protocol powered ledger to keep track of the total assets in each connected blockchain.

The cosmos network uses a graph connection structure that allows for connection between blockchains and also other interledger networks such as the ones described in this section (Section 4.4).

4.4.4 Strong Federations

The Strong Federation network[28] facilitates CCATs between a parent chain and its sidechains [48] (see Section 4.1.1 about sidechains). In this way, Strong Federations is not a general solution to allow communication between blockchains such as Bitcoin and Ethereum.

The mechanism requires two different types of actors in the relay mechanism that operate hash-locked contracts on the connected chains. The Liquid network is the proposed first implementation of Strong Federations.

4.4.5 COMIT Network

The Cryptographically-secure Off-chain Multi-asset Instant Transaction "COMIT"[29] network is a general network connecting blockchains supporting instant transactions [49]. This is through the use of off-chain smart contracts and already operational payment channels. The network proposes the Comit Routing Protocol (CMP) in which actors manage the payment channels and hashed time-locked contracts to allow for low-trust inter-blockchain transactions. The intermediary ledger uses a Tendermint BFT consensus algorithm. It also claims that the network will be able to integrate with the traditional banking system.

The white paper identifies several key features that the connecting chain are required to meet:
- Double-spend protection
- Basic hash functions
- Time-locks
- Payment channel support (such as the Lightning Network in Bitcoin and Raiden Network[30] in Ethereum)

4.5 Oracles

Oracles are a form of notary as discussed in Section 4.1.2 and provide external information that is not normally accessible from within the blockchain. In the context of smart contracts, oracles act as data

---

[27] https://cosmos.network/
[28] https://blockstream.com/2017/01/16/strong-federations-paper-released-liquid.html
[29] http://www.comit.network/
[30] http://raiden.network/

providers for information that is off the blockchain including other blockchains, APIs and other internet data. Oracles provide a way for one blockchain to contact and interact with another, and hence are considered an interoperable service.
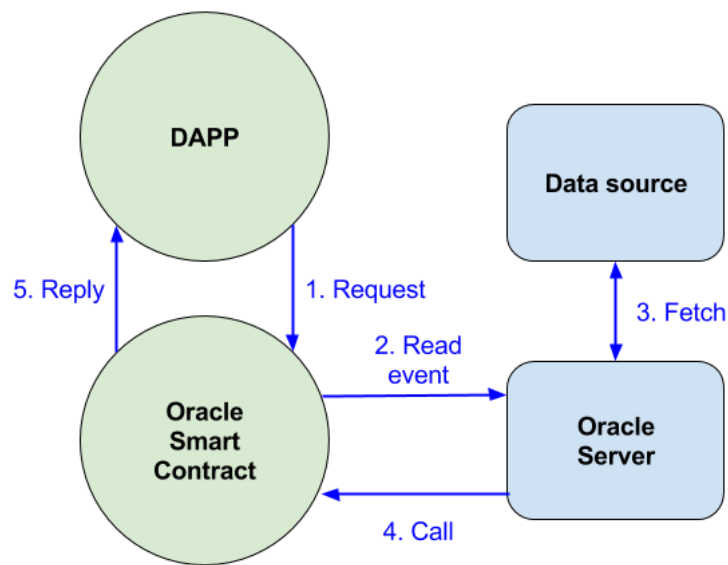

Figure 9: Blockchain oracle system

As seen in Figure 9, oracles provide an interface contract on the blockchain from which information can be requested. The requesting smart contract makes a function call to the oracle smart contract which emits an event when the information is requested. The oracle provider also runs a traditional server (not on the blockchain) that watches for events, which trigger the oracle server to fetch the requested information. This information is then delivered to the original requesting smart contract.

As data stored on the blockchain has a significantly higher price than traditional data storage (see Appendix A.), often only the most essential piece of information is returned by an oracle.

5. System Design

The final cryptocurrency conversion system as well as the reasons for choosing this mechanism are explained in this section.

5.1 InterCrypto Mechanism

After conducting the investigation into the technologies available that offer cryptocurrency conversion services as outlined in Section 4, several mechanisms were considered to build an on-blockchain cryptocurrency conversion service as seen in Table 6. In August 2017, this table is considered to be a very thorough overview of the potential solutions available.

	Governance model	Conversion capability	Considerations
Use a relay contract	Decentralized	1-to-1 unidirectional or bidirectional depending on contract capabilities	Only Ethereum to Bitcoin relay exists in production
Build a relay contract	Decentralized	1-to-1 unidirectional or bidirectional depending on contract capabilities	Only one way from a smart contract blockchain is easily implementable
Use an interledger network	Decentralized	High	No suitable operation interoperable network existed at the beginning of the project
Build an interledger network	Decentralized	High	Extremely difficult and not within scope of this project
Use an oracle with a centralized exchange	Centralized	1-N unidirectional	Both oracle and exchange are single points of failure
Use an oracle with a decentralized exchange	Centralized	1-N unidirectional	Decentralized exchanges are not mature enough to currently use and an oracle is a single point of failure
Use a decentralized oracle with decentralized exchange	Decentralized	1-N unidirectional	Decentralized oracles and exchanges are not mature enough to currently use
Build an ERC20 exchange	Decentralized	N-N within Ethereum ERC 20 ecosystem only	Several other projects already exist

Table 6: Different mechanisms that facilitate a cryptocurrency conversion contract

After consideration of many of the above mechanisms, the following three were considered suitable:

1. Use a relay contract
2. Build a relay contract
3. Use an oracle with a centralized exchange

The final mechanism (number 3) was chosen as it has the greatest potential for conversion with the most number of other cryptocurrencies. As discussed in Section 7.1, the availability of the ILP had evolved throughout the project such that by the end of the project it may have been mature enough to attempt to create the project using this mechanism. As it was not in such a mature stage at the beginning of the project, it was not considered at this time.

5.2   Ethereum

After conducting the investigation of smart contract enabled blockchain technologies as seen in Section 3, Ethereum was identified to be the most appropriate blockchain for the purpose of this project because:

- It offers turing complete capabilities.
- It is permissionless so access could be easily obtained.
- The technology is currently in production, with a large support community and documentation set.

The smart contract programming Solidity was selected to be used for the project due to its significantly greater development support compared to alternative EVM languages Serpent[31] and LLL[32].

As mentioned in Section 2.3, due to the need for every node in the network to validate every transaction, all smart contracts must execute in a strictly deterministic fashion which imposes severe limitations on current Ethereum smart contract capabilities:

- Contracts cannot access information that is not in the Ethereum blockchain. This also means that external internet sources such as websites or APIs and other blockchain data cannot be accessed, which is the primary driver for the use of oracles. This characteristic was the primary development obstacle that this project needed to overcome to provide an on-blockchain cryptocurrency conversion contract.
- Contracts have no timed trigger functions, as well as no concurrent execution capabilities.
- Random numbers and other functions dependant on non-deterministic sources (such as human time) are not available.

Several important Ethereum features are discussed below to give context to the InterCrypto smart contract.

---

[31] https://github.com/ethereum/wiki/wiki/Serpent
[32] https://github.com/ethereum/cpp-ethereum/wiki/LLL-PoC-6/7a575cf91c4572734a83f95e970e9e7ed64849ce

5.2.1 Gas

The use of smart contracts involves the executions of the EVM opcodes, which is paid for in transaction fees. The opcodes themselves are not directly mapped to Ethereum prices but instead have a "gas" cost and each transaction is transmitted with a gas price. This is done so that miners can order transactions by gas price without knowledge of the duration of execution of a transaction in advanced. Opcode costs range from 0 gas (eg RETURN, STOP) to 32000 (eg CREATE) gas. The final amount of gas used over all executed opcodes is multiplied by the gas price to get the transaction cost in Ether. Due to this feature, careful design can be employed to minimize Ether costs of smart contract function calls.

Transaction fees in any Proof of Work blockchain such as Ethereum are orders of magnitude above those in traditional centralized architectures. As an example of this, in August 2017 it was found that the cost of storage in Ethereum was 575000 times that of an AWS S3 storage instance as seen in Appendix A.

Unlike Bitcoin's 1Mb block size limit, Ethereum blocks are limited by the total amount of gas spent and are flexible depending on the mining market. In August 2017, this was over 6.7 million gas. Transactions that exceeded this gas limit would be recorded to the ledger and the gas would be given to the miner, but no state changes would be recorded.

5.2.2 Events

The Ethereum logging feature is important to the InterCrypto contract and additionally provides a good example of the EVM gas mechanism.

A variable used in InterCrypto is the address (32 bytes) *depositAddress* which stores the address to send Ether to start a ShapeShift conversion. To allow the user to see this address two options were available:
1. Store the value of the deposit address in permanent storage. This implied storing it in the contract state and would require the execution of the STORAGEADD opcode at a cost of 20000 gas.
2. Emit the value of the deposit address in an event. This implied no permanent storage of the value and the execution of the GLOGDATA opcode costing a total of 256 gas at 8 gas per byte.

The EVM logging feature, accessed by the Solidity "event" keyword, does not store any data in storage. Instead, the block header stores a bloom filter of all transactions that emit logs in the block which allows applications to quickly scan headers to see if events were triggered. An application can then re-execute the transactions to obtain the logged data [50]. Event log data is not accessible by other contracts but can serve as a useful tool not only to reduce gas costs but to efficiently emit information to external applications.

5.2.3 Functions

Function calls to contracts can be of two types:

- Calls that modify the state of the blockchain. These must be submitted to the network as transactions and paid for in gas.
- Calls that only read data from the blockchain. These can be safely called without any need to pay gas fees and the "constant" solidity keyword can be used to enforce read only privileges to a function.

Additionally, functions can have one of four different visibilities:
- External functions form part of the contract interface and could only be called through a transaction. External functions can not be called internally.
- Public functions form part of the contract interface and could be called through a transaction or internally by members of its own contract.
- Internal functions can only be called internally from its own contract or derived contracts.
- Private functions can only be called internally from its own contract but not from derived contracts.

In the UML diagram in Figure 13, external and public functions are marked with a "+", and internal and private functions with a "-".

5.3 Oraclize

After the investigation of interoperable technologies detailed in Section 4, the use of a centralized oracle service was identified to be the most appropriate way to build a smart contract that allowed for cryptocurrency conversion as:
- Oracles are simple to use and are currently the most popular form of inter-chain and off-chain data exchange.
- Several oracles exist on the Ethereum blockchain that provide a rich set of oracle functions such as GET and PUSH requests to APIs.
- Smart contracts using oracles can be reasonably simple and require no cryptographic knowledge to use the oracle, unlike relays or hash-locking.

Oraclize[33], an oracle provider, offers a low-trust service by providing a cryptographic proof of the service that they performed (proof that the information was requested and the result was not tampered with). This oracle provider was chosen as the most suitable oracle for this project due to its high reputation and its technical approach to reduce required trust in the oracle.

The Oraclize smart contract can be found at the following address on the blockchain:
- Ethereum mainnet: 0x6f28b146804dba2d6f944c03528a8fdbc673df2c
- Rinkeby testnet: 0x61048b56d6e4fca6a1f6b5dac76255a413f37f4c

---

[33] http://www.oraclize.it/

The source code can be verified and viewed through the Etherscan blockchain explorer or on their GitHub account, where the API can also be found [51]. Figure 10 shows the UML diagram of the relevant Oraclize functions that were used by the InterCrypto smart contract.

Figure 10: Oraclize smart contract UML Diagram showing relevant public functions

5.4   Shapeshift

The ShapeShift platform provides a crypto-crypto exchange service. Importantly, the user does not have to create a ShapeShift account to use this service. The platform charges a flat fee of 0.65% on all transactions in August 2017. ShapeShifter has a well-developed API to access its service.

ShapeShift was chosen as the primary exchange service to be used in the project, for its sound market reputation, based on a traditional business trust model. Figure 11 shows the UML diagram displaying the API interface that the ShapeShift server provides that were of use to the InterCrypto smart contract [52].
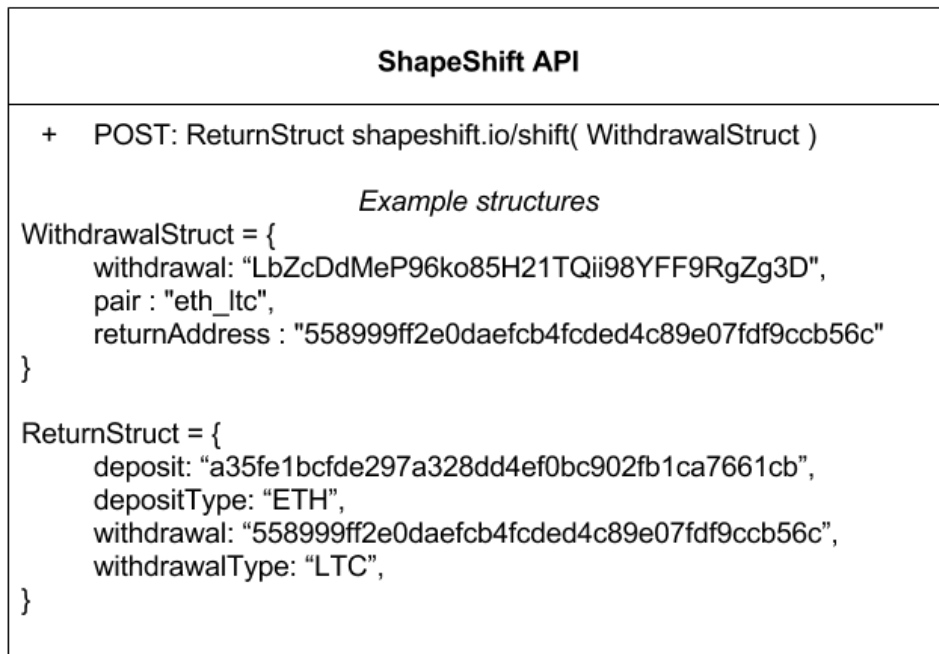
```
                              ShapeShift API
  +    POST: ReturnStruct shapeshift.io/shift( WithdrawalStruct )

                              Example structures
WithdrawalStruct = {
      withdrawal: "LbZcDdMeP96ko85H21TQii98YFF9RgZg3D",
      pair : "eth_ltc",
      returnAddress : "558999ff2e0daefcb4fcded4c89e07fdf9ccb56c"
}

ReturnStruct = {
      deposit: "a35fe1bcfde297a328dd4ef0bc902fb1ca7661cb",
      depositType: "ETH",
      withdrawal: "558999ff2e0daefcb4fcded4c89e07fdf9ccb56c",
      withdrawalType: "LTC",
}
```

Figure 11: Shapeshift UML Diagram showing relevant API functions

5.5   InterCrypto

After consideration of all of the technologies available, a smart contract on the Ethereum blockchain using Oraclize to interact with the ShapeShift exchange was chosen to build the cryptocurrency conversion contract.

The final system can be seen in Figure 12. The InterCrypto smart contract, interface, demo client Wallet and website were all developed as part of this project. The Ethereum Name Service[34] system of contracts was configured to be used by the InterCrypto interface.
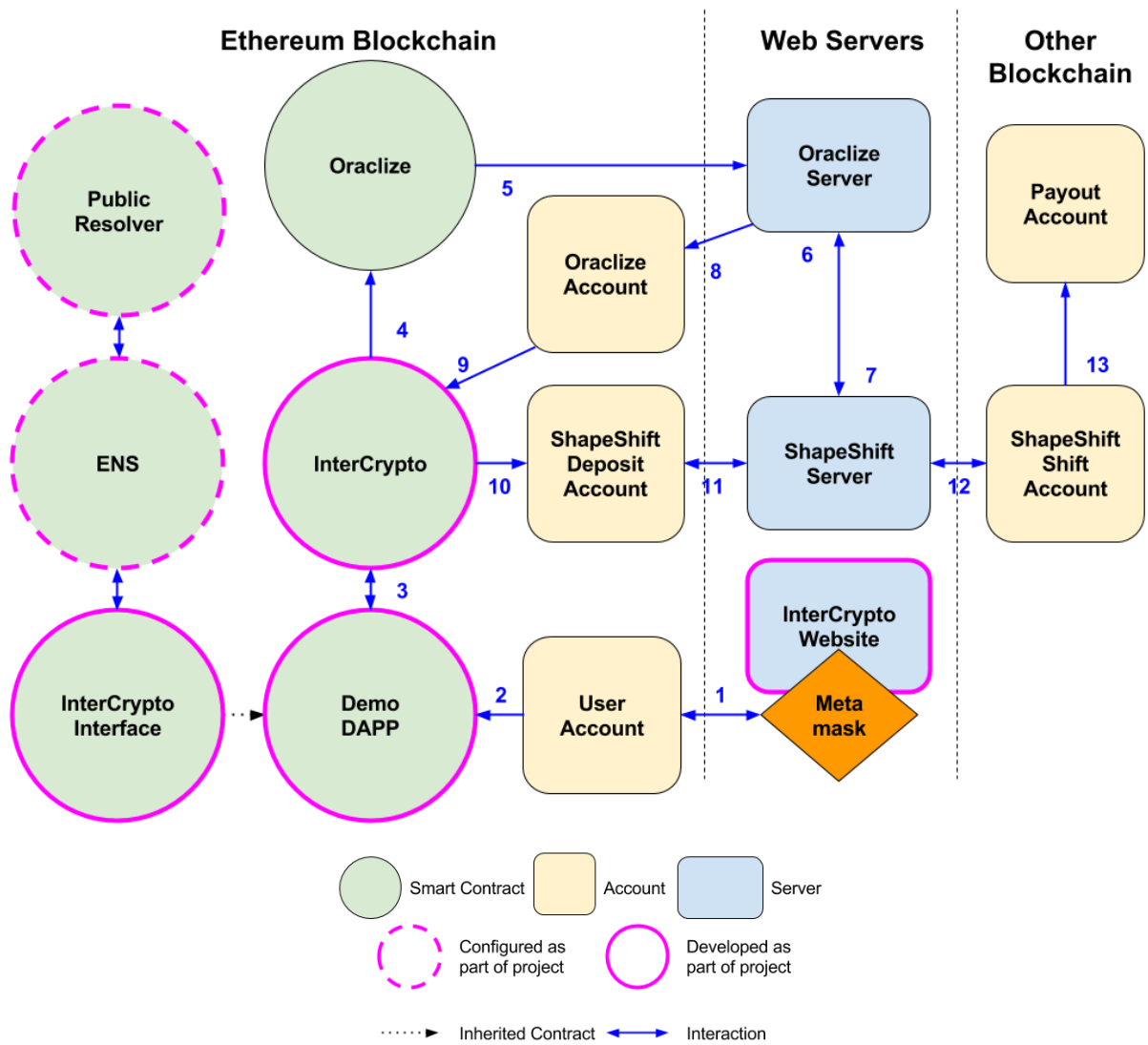
---

[34] https://ens.domains/

34

Figure 12: InterCrypto service diagram

Figure 12 shows a diagram of the system and interactions between components, which are described below.

1. The Metamask extension creates a function call transaction using the web application user's Ethereum account.

2. The function call is sent to the demo client smart contract.

3. The demo client smart contract makes a function call with some Ether to InterCrypto calling *convert*() with parameters telling it to which blockchain and to what payout address the converted cryptocurrency should arrive.

4. InterCrypto makes a function call to Oraclize calling *query2*() with the parameters of a POST request to be made to the ShapeShift server.

5. The Oraclize server reads these parameters from the Oraclize smart contract.

6. The Oraclize server makes a POST request to the ShapeShift server to set up a new cryptocurrency conversion.

7. ShapeShifter replies to the Oraclize server with the Ethereum deposit address which it will use to create the cryptocurrency conversion.

35

8. Oraclize creates a function call transaction using the Oraclize's reply Ethereum account.

9. InterCrypto's __*callback*() function is called in the transaction from Oraclize telling InterCrypto the address of the ShapeShift deposit account.

10. InterCrypto sends the original Ether minus the amount paid to Oraclize to the ShapeShift deposit account.

11. ShapeShift recognizes that InterCrypto made the deposit.

12. ShapeShift create a transfer transaction using ShapeShift's shift account on the other blockchain.

13. The converted cryptocurrency is sent to the final payout address on the other blockchain.

Figure 13 shows the UML diagram showing the fields and methods of the InterCrypto smart contract. The InterCrypto smart contract can be found at the following addresses:

- Ethereum mainnet: [0xc876B7545c2b6bFAC4cD82dB1EbD33F70D738277](#)
- Rinkeby testnet: [0x29D4dd056EF94028569558b0ABf6E6693330ccfE](#)

Figure 13 also shows the three event logs used to cheaply store historical conversion information and act as triggers for front end applications. The contract's conversion function can be used by calling external functions *convert1*() or *convert2*(). For the rest of the report *convert*() will refer to both of these function, which have very similar behaviour as explained in Section 7.3.3.

InterCrypto Smart Contract

+   address owner
+   map (uint => Conversion) conversions
+   map (address => uint) recoverable
+   uint oraclize_gaslimit
-     uint conversionCount
-     map (bytes32 => uint) oraclizeMyId2conversionID
 	
struct Conversion {
      address returnAddress;
      uint amount;
}

event ConversionStarted(uint conversionID) event ConversionSentToShapeShift(uint conversionID, address depositAddress) event ConversionAborted(uint conversionID, string reason) event Recovered(address recoveredTo, uint amount);
+   uint convert1(string _coinSymbol, string _toAddress) +   uint convert2(string _coinSymbol, string _toAddress, address _returnAddress) +   void __callback(bytes32 myid, string result) +   uint getInterCryptoPrice() +   void cancelConversion(uint conversionID) +   void recover() +   void kill() +   void update_oraclize() +   transferOwnership(address newOwner) -   uint engine(string _coinSymbol, string _toAddress, address _returnAddress) -   bytes addressToBytes(address _address) -   bytes concatBytes(bytes b1, bytes b2, bytes b3, bytes b4, bytes b5, bytes b6, bytes b7) -   string createShapeShiftConversionPost(string _coinSymbol, string _toAddress)

Figure 13: InterCrypto smart contract UML diagram showing all fields and functions

Several administration features were also implemented in the contract. The following functions can only be executed by the contract owner, which is initialized as the contract deployer when the contract is deployed:

- *transferOwnership*(*newOwner*) - This function allows for transferal of the ownership of the contact.
- *updateOraclize*() - In the event that Oraclize update their smart contract, this function can be used to update InterCrypto to point to the new contract. No input parameter is needed as the Oraclize address resolver is used to get the new Oraclize address.
- *setGasLimit*(*_newLimit*) - This can be used to adjust to the Oraclize *gasLimit*.
- *kill*() - This can be used to execute the SUICIDE opcode, making the contract unusable and returning all funds to the contract owner.

The *getInterCryptoPrice*() public function allows for users to inspect the current Oraclize price so that they can correctly determine the cost of using InterCrypto.

5.6   InterCrypto Interface

The InterCrypto Interface contract can be used to easily allow client contracts to connect and use InterCrypto. Smart contracts that wish to use InterCrypto need to import and inherit the interface as shown below. To convert Ethereum to another cryptocurrency, the client smart contract can then call InterCrypto with appropriate parameters.

```
import "github.com/ugmoo4/inter-crypto/contracts/InterCrypto_Interface.sol";

contract MySmartContract is usingInterCrypto {
        function myConvert(_coinSymbol, _toAddress) payable external {
                uint conversionID = intercrypto_convert(msg.value, _coinSymbol, _toAddress);
        }
}
```

Figure 14 shows the functions that are exposed to the derived client contract.

InterCrypto Interface Contract
+   address owner +   bytes32 ResolverNode +   bytes32 InterCryptoNode
event ConversionStarted(uint conversionID) event ConversionSentToShapeShift(uint conversionID, address depositAddress) event ConversionAborted(uint conversionID, string reason) event Recovered(address recoveredTo, uint amount);
-     setNetwork() -   updateResolver() -   updateInterCrypto() -   updateInterCryptonode(bytes32 newNodeName) -   uint getCodeSize(address _addr) -   uint intercrypto_convert(uint amount, string _coinSymbol, string _toAddress) -   uint intercrypto_convert(uint amount, string _coinSymbol, string _toAddress, address _returnAddress) -   transferOwnership(address newOwner)

Figure 14: InterCrypto interface contract UML diagram

5.6.1   Ethereum Name Service

To resolve the address where the InterCrypto smart contract exists, the Ethereum Name Service (ENS) has been used. ENS is a smart contract on Ethereum that acts as a decentralized human readable name to Ethereum address resolver. Like the internet's DNS, ENS provides a hierarchical naming system with top level names of .eth and .test available depending on which blockchain is used.

The ENS smart contract can be found at the following addresses:
- Ethereum mainnet: 0x314159265dd8dbb310642f98f50c066173c1259b
- Rinkeby testnet: 0xe7410170f87102DF0055eB195163A03B7F2Bff4A

Using ENS, a domain was created to return the current address of the InterCrypto smart contract as seen in Table 7. The advantage of using an address resolver instead of hardcoding the address into the InterCrypto interface is that the InterCrypto contract can be updated easily for feature upgrades or bug fixes without needing to redeploy any client smart contracts. By using ENS, the resolver is complying to a standard mechanism that the rest of the Ethereum ecosystem is using, which improves forward compatibility, something not possible until the existence of ENS.

	Ethereum mainnet	Rinkeby testnet
InterCrypto	intercrypto.jacksplace.eth	intercrypto.jackdomain.test
Demo client Wallet	wallet.intercrypto.jacksplace.eth	wallet.intercrypto.jackdomain.test

Table 7: ENS domains used to resolve InterCrypto smart contracts

The ENS node resolver was then pointed to a pre-deployed public resolver at ENS address "resolver.eth" which would resolve the current InterCrypto contract address. The process of obtaining the mainnet .eth domains is detailed in Appendix D.

6. Implementation Details

The way in which the Ethereum network (testnets and mainnet) was connected to and interacted with as well as the different programming environments is discussed in this section.

6.1   Interacting with Ethereum

Several methods were used throughout the project to connect to the public Ethereum blockchain and run virtual nodes. Each of the methods below creates the web3 JavaScript object which can then be access through a rpccors connection locally on the machine in an application, or through the browser if configured correctly. The web3 object is the official interface to the Ethereum blockchain and offers a wide set of features such as creating accounts, contracts and watching events [53].

Each Ethereum connection method had different advantages and disadvantages as summarised in Table 8. As such, different connection methods were used at different phases of the project.

	Geth	Metamask	TestRPC	JavaVM
Connect to public blockchain	Yes	Yes	No	No
Connect to public smart contracts	Yes	Yes	No	No
Node type	Any	Light only	Virtual	Virtual
CLI	Yes	No	Yes	No
GUI	No	Yes	No	No
web3 features	All	All	All	Limited
Mining	Yes	No	Virtual	No
Speed	Normal	Normal	Instant	Instant
Ease of configuration	Medium	Simple	Medium	Simple

Table 8: Features offered by different methods of connecting to Ethereum

6.1.1   Geth

The Geth[35] software is the official Ethereum client software. It is open source and maintained by the Ethereum Foundation. The software is able to connect to the Ethereum mainnet, any testnet or private Ethereum blockchains, or create a new private blockchain.

---

[35] https://github.com/ethereum/go-ethereum/wiki/geth

Geth can create a full node, with the entire blockchain downloaded, a light node with only block headers downloaded or hybrid nodes in the middle depending on the application requirements. Geth runs a command line interface only. The console can be used to access and call functions to any Ethereum account [54].

Geth was used in the InterCrypto project to connect to the Truffle framework, and also used to mine testnet Ether for testing. The command line could be used to quickly inspect accounts transactions in the blockchain.

6.1.2 Metamask

Metamask[36] is a Google Chrome browser extension that runs an extremely light node. Metamask can connect to Ethereum and all of the popular public testnets. Metamask provides a simple user-interface that removes most of the complication of connecting to Ethereum and managing accounts. The extension provides the web3 object to website and will allow for transactions and function calls to be made with the approval of the browser's user.

The function of Metamask through the GUI is limited to interacting only with its own accounts and those provided by web applications. Metamask cannot be used for mining [55].

Metamask was used in InterCrypto to allow users to connect to the InterCrypto app using a graphical user interface. It was also used to connect to the online Remix compiler, which was found to be much easier to connect with than using Geth.

6.1.3 TestRPC and the JavaVM

TestRPC[37] is a software that creates a virtual Ethereum node. The node has all the features of a normal node, except that it is not actually connected to an Ethereum blockchain. The TestRPC software runs transactions instantly, as opposed to having to wait for the transaction to be mined by the public network. This is a significant advantage when running testing suits with several transactions as it takes seconds instead of minutes to run tests. Unfortunately, as it is not connected to a public blockchain, any interaction with other smart contracts on public blockchains is impossible. TestRPC offers a full web3 object and a command line interface to interact with the virtual blockchain.

The TestRPC client was originally used with the Truffle framework[38] to test the InterCrypto contract before it started interacting with the public Oraclize or ENS contracts [56].

---

[36] https://metamask.io/
[37] https://github.com/ethereumjs/testrpc
[38] http://truffleframework.com/

The online Remix compiler[39] [57] also offers a virtual EVM environment that acts in a similar way to TestRPC. The JavaVM node is only accessible via the Remix compiler and was even more limited than Metamask. The JavaVM node was used occasionally for quick tests of smart contract features that did not interact with Oraclize or ENS.

6.2 Programming and Testing

Several programming environments were used throughout the project. Each environment had different pro's and con's as seen in Table 9. In hindsight, it was considered that using Remix to develop the final smart contract and then using Truffle to build an automated testing suite would have been the optimal development strategy, albeit the smart contract was actually developed in the reverse order.

	Truffle	Custom framework	Remix
Deployment overhead	High	Normal	Normal
Automated testing	Yes	Yes	No
Graphical debugging	No	No	Yes

Table 9: Different programming environments used in the project

6.2.1 Truffle Framework

The Truffle framework is currently the most popular Ethereum smart contract development framework. Truffle is offered as an easy-to-install NodeJS package. The environment offers several convenient functions that can be used to compile, deploy and test solidity smart contracts. The environment does not include an Ethereum node so a Geth and TestRPC node were configured to connect to Truffle.

The framework also set up an extra level of abstraction on top of the contract ABIs that allowed for several particularly convenient contract features. The framework deployed an extra contract and made at least two transactions to set up this contract abstraction which came at the expense of computation time and gas.

The greatest advantage of Truffle is a facility to easily create an automated smart contract testing suite, including having these tests integrate with web application testing. The testing suite is an abstraction provided on top of the popular mocha testing suite package in NodeJS. The test suite came bundled with the same overheads as the contract abstraction and was found very useful while using a virtual node but became cumbersome when using a public Ethereum blockchain.

Truffle was used to develop the first several versions of the InterCrypto smart contract however was found to be slow and expensive as soon as interactions with other smart contracts were needed (Oraclize and

---

[39] http://remix.ethereum.org/

ENS). A full test suite was developed to test the initial prototypes and this was especially useful while the solidity language was being experimented with.

6.2.2 Custom Framework

To compensate for the contract deployment overhead that Truffle framework requires, a custom contract framework was built in NodeJS. The custom framework used the default web3 contract abstractions and used the mocha testing pack to develop an automated testing suite.

6.2.3 Remix Compiler

The Ethereum foundation officially support an online Integrated Development Environment (IDE) called Remix. This environment can be easily accessed at https://remix.ethereum.org/ and offers the virtual JavaVM environment, or can connect to Metamask or a locally run node such as Geth or TestRPC.

The environment provides a rich IDE with many different solidity compilers, a graphical debugger, a file explorer and URL file imports (example importing code from GitHub files such as show in Section 5.6).

New contracts can be compiled and deployed without any additional overheads and previously deployed contracts can be connected to provided that the user provides the interface contract code. Remix provides a graphical interface to easily interact with smart contracts and has a step debugger to step through contract function calls. It also provides a useful set of compiled information such as the bytecode, ABI, contract deploy code and gas estimates.

It was found that the Remix IDE was the most convenient and easy to use environment for fast development.

7. Challenges and Smart Contract Optimisation

Throughout the project, a number of challenges were overcome to conduct research and build the InterCrypto contract. The final deployment of the contract required several stages of iteration to be financially optimized. These challenges and optimizations are detailed in this section.

7.1 Blockchain Industry Freshness

Due to the freshness of the blockchain industry, combined with its recent gain in popularity and visibility, the blockchain space (now referred to as an industry) outpaced the speed of this academic project. Such was the acceleration in blockchain development that several new prominent blockchains (NEM, Wave and NEO), several new decentralized cryptocurrency exchanges and several new interledger projects were discovered at the end of the project that were not found during the initial survey. At the end of this project, large amounts of work were being undertaken to release interoperable blockchains Ripple, Polkadot, Comit and Cosmos that would potentially evolve the whole blockchain from a "Smart Contract" to an "Interoperable" generation of blockchain usage.

Had this project been started in September (the end of the project) instead of in July, then the InterCrypto mechanism chosen to solve the on-blockchain cryptocurrency conversion problem as summarised in Table 6 may have been quite different.

7.2 Testnet Stability

Throughout development the Rinkeby and Ropsten testnet were used with different characteristics. The Ropsten network was initially used for this project but was switched to using Rinkeby due to improved speed as discussed below.

7.2.1 Ropsten

The Ropsten testnet is the oldest operational testnet. It uses a Proof of Work consensus protocol as described in Section 2.1.2, the same as the Ethereum mainnet.

As testnet Ether does not have any value (as agreed by all users), very little incentive exists for miners to contribute their computational power. To put this in perspective, in April 2017 the amount of computational power that was used on the Ropsten network was 0.000069% [58] [59] of that used on the Ethereum mainnet. Due to the decreased amount of network power, it is quite computationally feasible to conduct a denial of service (DoS) or 51% attack against the network. Such spamming attacks have slowed down block times significantly since February 2017 and the network remains slow and sometimes difficult to synchronize to [60].

7.2.2 Rinkeby

To address issues that the development community had been experiencing with Ropsten, the Ethereum Foundation set up the Rinkeby testnet which runs a of Proof of Authority consensus protocol. Unlike Proof of Work, in which miners are required to solve a complex mathematical puzzle to get the privilege of adding a block to the blockchain, the permission to add blocks is granted explicitly [61]. In this way, the Rinkeby network has a private mining governance model. This effectively disables attacker's ability to conduct many of the DoS and 51% attacks that are seen on Ropsten simply because attackers do not have permission to mine.

The difference in the consensus model did not affect the development of InterCrypto, but may be relevant for other projects. The connection to the Rinkeby network did depend on the availability of the ropsten.io web server and occasionally was observed to be unavailable. Despite the decrease in availability, the Rinkeby network had much more stable performance and was used for most of the project.

7.3 Smart Contract Challenges

Several interesting Ethereum specific programming challenges were encountered in the project that are worth mentioning.

7.3.1 Test Criteria

Testing the connection to an Ethereum blockchain and the execution of a transaction present a difficult question: *what do we compare it against?* The solution used was to compare it to another node in the network. The API supplied by infura.io[40] gave access to a reasonable set of data that could be effectively used by the testing suite test against.

7.3.2 Function Local Variable Limit

An interesting Solidity limitation was encountered when programming the *concatBytes*() and *createShapeShiftConversionPost*() functions in the InterCrypto program. Due to the EVM stack depth limit, a maximum of 16 local variables (including parameters and return parameters) can be declared per function [62]. Due to this limitation, the *concatBytes*() function underwent several iterations and optimizations to remain simple and efficient while conforming to this limit.

7.3.3 Function Call Overloading

While the solidity language allows for overloading of contracts, low level function calls that forward on custom amounts of gas or Ether using the direct call feature as shown below could not distinguish between such overloaded functions.

> *return interCrypto.convert2.value(amount)(_coinSymbol, _toAddress, _returnAddress);*

---

[40] https://infura.io/

It was for this reason that the convert function in InterCrypto was renamed *convert1*() and *convert2*(), to allow the Interface contract to successful distinguish them and eventually provide the overloaded interface function below:

function intercrypto_convert(uint amount, string _coinSymbol, string _toAddress)

function intercrypto_convert(uint amount, string _coinSymbol, string _toAddress, address _returnAddress)

7.3.4 Execution Output for Debugging

Solidity does not have an equivalent output stream or console output as in most high-level languages. This is due to the fact that contract code is executed in the EVM, a decentralized world computing network and not on a computer under the control of the contract author. Due to this, smart contract debugging becomes challenging due to the inability to check variables during execution. A common and practical solution to this [63] is to use the Solidity event feature to output an event with desired variable values. This solution obviously creates extra undesired gas costs and so was used for the first phase of smart contract development, after which all debugging events were removed from code as reported in Section 7.5.

As an example, the following event was defined and then used in InterCrypto. By watching the InterCrypto contract for this event when calling the *recover*() function, the value of the *amount* variable could be viewed.

```
event consoleLog(string message, uint value);
function recover() external {
        uint amount = recoverable[msg.sender];
        consoleLog("recover().amount", amount");
        recoverable[msg.sender] = 0;
        if (msg.sender.send(amount)) {
                Recovered(msg.sender, amount);
        }
        else {
                recoverable[msg.sender] = amount;
        }
}
```

7.3.5 Recovery

Several important conditions were identified that would cause InterCrypto conversion function *convert*() to fail as seen in Figure 15:

1. **Bad user inputs** could be supplied by the user for the *_coinSymbol* and/or *_toAddress* parameters. These strings had to be in a specific format as defined in the ShapeShift API. To reduce this risk, examples are supplied in the public code repository and the application website has an address checker that checks that the user's parameters are valid according to ShapeShift. The *isValidString*() function does some checks for obviously incorrect arguments that are not alphanumeric characters or are too long, however, it is still within the user's control to ensure that parameters are correct.

2. **ShapeShift trades become unavailable for particular cryptocurrencies** due to events such as the Bitcoin Cash hard fork or other technical issues that ShapeShift may have with their service.

3. **ShapeShift service becomes unavailable** due to server failure or other technical failure.

4. **Oraclize service becomes unavailable** due to server failure or other technical failure.

5. **ShapeShift fails to conduct the currency conversion** due to the deposit amount exceeding the ShapeShift limit, ShapeShift not recognising the deposit to the Ethereum deposit address or some other technical failure.
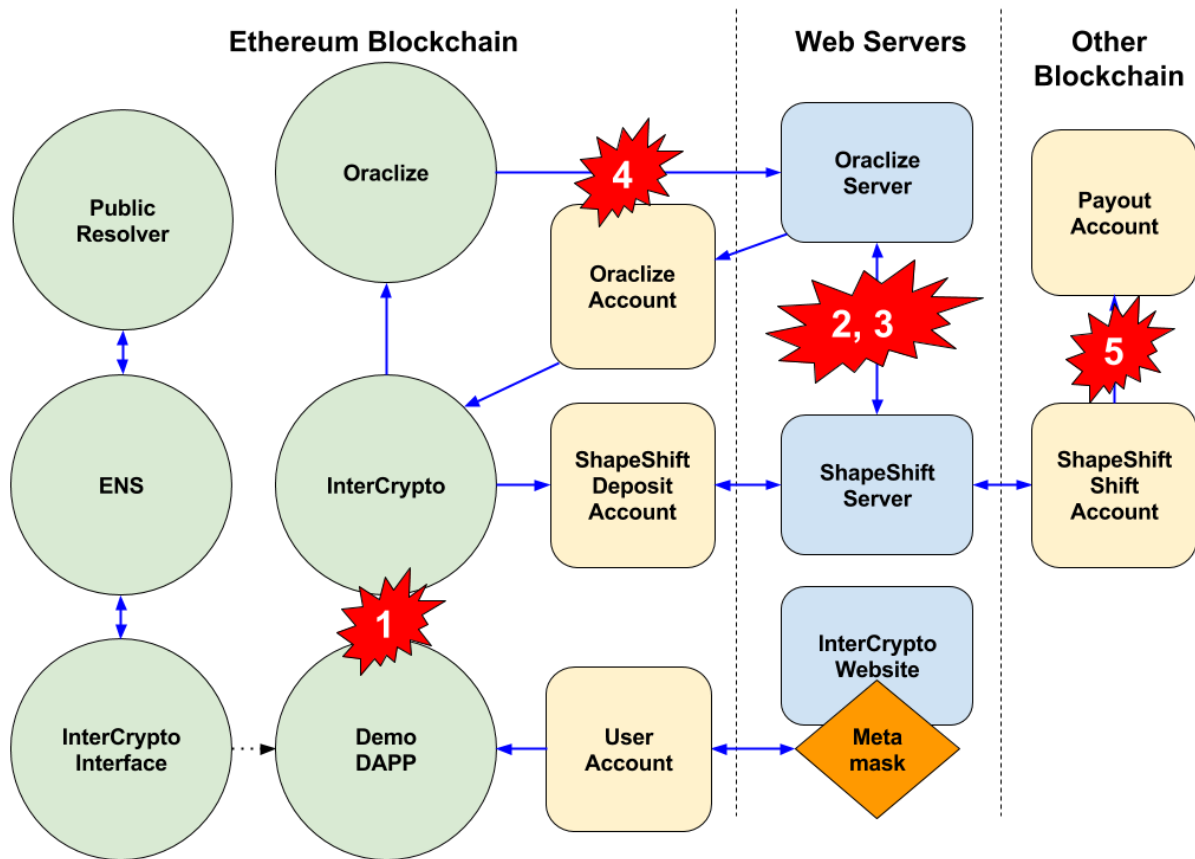

Figure 15: Failure points in the InterCrypto service

To mitigate the risk that Ether is forever lost, a recovery feature that deals with cases one to four above was built into InterCrypto as follows:

- The *ConversionAborted*(uint *conversionID*, string *reason*) is emitted whenever a failure condition is detected by InterCrypto that also states a message with the reason for the conversion failure. Application can watch for this event to check if the cryptocurrency conversion failed.

- If a conversion fails, the *recoverable* mapping variable is incremented by the amount of Ether that the conversion failed to convert, minus any fees paid to Oraclize.

- In the case that the Oraclize service fails (case four), there is no way for the InterCrypto contract to know about this automatically. This can only be discovered by observing the InterCrypto smart contract and checking that the *ConversionSentToShapeShift* event is not emitted. In this case, the *cancelConversion*() can be called to recover these funds which will increment the *recoverable* mapping accordingly.

46

- Users can inspect the public *recoverable* variable for any funds and call the *recover*() function to withdraw them using the safe withdrawal format [64].

Case five is handled off-blockchain by ShapeShift. In the case that ShapeShift is unable to execute the cryptocurrency conversion, they will send the deposited Ether back to an address that InterCrypto provides. By default, the InterCrypto function *convert1*() configures this return address to be that of the function caller (the InterCrypto client). If this is a contract then the fallback function should accept Ether (using the "payable" keyword) at a minimal gas cost. Alternatively, the InterCrypto client can set the return address to be a custom address using *convert2*(), which may be appropriate when the InterCrypto client smart contract should not accept Ether payments.

The client smart contract and application will need to implement the use of the InterCrypto recovery feature on a case-by-case basis, depending on the client smart contract application.

7.3.6   Security Considerations

Several important major and minor security risks have been identified and mitigated throughout the development of the InterCrypto smart contract. Several security checklists [64] [65] as well as general auditing was done to reduce technical and logic flaws. The following details the major risks to the contract that were mitigated.

Gas Limit Stalling

Contract calls can be locked when their execution gas price rises above the gas price limit set by the network (6.7 million gas in August 2017). When this happens, a contract is considered fully or partially "locked" depending on the role of the unusable function. The gas price of a function can increase over time for a number of reasons such as iterating over a dynamic array that increases in size. If the contents of such an array can be added to by users then this array can grow until iterating over it costs more gas than is allowed in a block.

This risk was mitigated by:
- Not using dynamic arrays, only mapping, so iteration over arrays was impossible.
- Limiting the size of user defined input strings. This was done within the function *isValidString*(string *_string*, uint *maxSize*).

Re-Entrancy

An important attack was conducted on the DAO smart contract in June 2016, irreversibly draining 3.6 million Ether from the contract (worth almost £50 million at the time). This attack did not hack the Ethereum blockchain itself but exploited a vulnerability in the DAO smart contract code.

The re-entrancy vulnerability is present when a contract transfers Ether to an address at which an attacker can deploy a contract. The attacker deploys this contract so that when it is sent Ether (through the fallback

function "()"), it makes a function call back to the victim contract, creating a recursive loop that will drain all Ether from the target, as depicted in Figure 16.
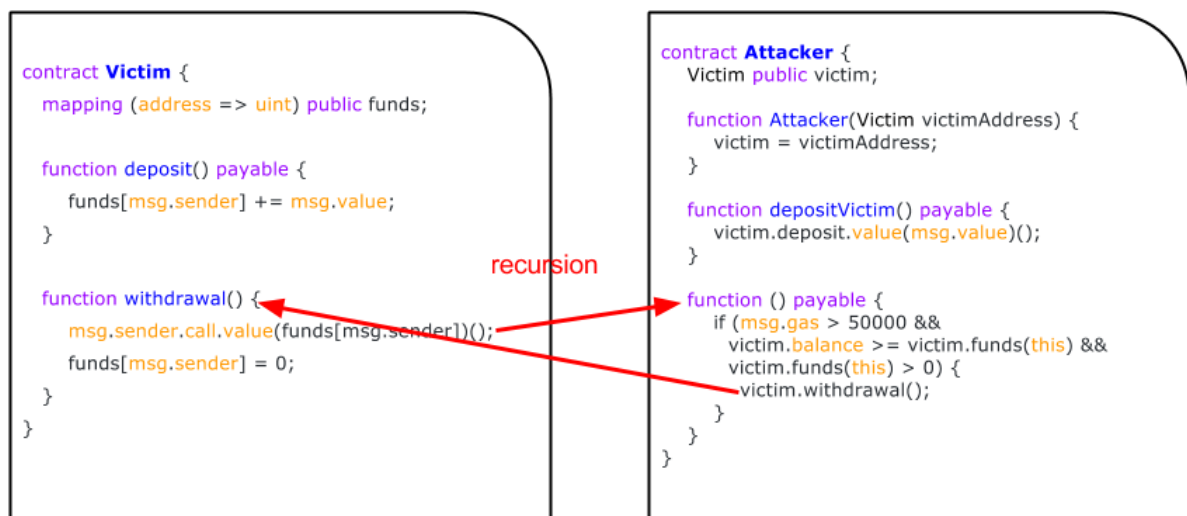

Figure 16: Illustration of DAO re-entrancy exploit

The re-entrancy risk was mitigated by:
- Using the safe withdrawal format as suggested by the Ethereum Foundation [64]. This was the format used in function *recover*().

In InterCrypto's *__callback*() function, the safe withdrawal format could not be applied when sending Ether to the ShapeShift deposit account and the following mitigations were applied:
- The following check at the start of the function was added to ensure that only Oraclize could execute the function:
  if (msg.sender != oraclize.cbAddress()) revert();
- To protect against the slim possibility that Oraclize themselves decided to exploit the re-entrancy vulnerability, transfers to their own account were disabled with the following line:
  require(depositAddress != msg.sender);
  The two checks in tandem meant that no one but Oraclize could call *__callback*() and Oraclize could not set up a recursive re-entrancy attack.

Integer Arithmetic Overflow

Solidity does not have any in-built arithmetic overflow protection. Bugs due to arithmetic overflow can be easy to miss and present security vulnerabilities and/or cause a contract to be locked. This risk was mitigated by:
- Using mapping structures instead of iterable arrays so that the smart contract required considerably less integer arithmetic.
- Not using any user defined function arguments in any integer arithmetic.
- Using suitably large variable sizes (256-bit unsigned integers) for storage that would easily be larger than any feasibly required variable size.

- Limiting the size of user defined input strings. This was done within the function *isValidString*(string *_string*, uint *maxSize*).

Poison Data

Contracts that store user defined inputs that stores and/or expose them to other users (on-blockchain or off-blockchain through external contract variable reads) create potential vulnerabilities that may lock the contract or create security flaws. An example of this would be emitting an event with a user defined string variable that is watched by a web application. Such a string could be constructed with JavaScript code so that it would execute malicious code on a browser page that compromises the users with the web application open. This is a classic example of a command injection attack.

This data poisoning risk was mitigated by:
- Limiting the size of user defined input strings. This was done within the function *isValidString*(string *_string*, uint *maxSize*).
- Limiting the characters allowed for used defined input strings to only numbers and characters. This was done within the function *isValidString*(string *_string*, uint *maxSize*).
- Not exposing any user defined input strings in events.

Contract Complexity

As in all programming, the greater the code complexity, the greater the susceptibility for coding errors that could create logic and/or security vulnerabilities. For this reason, no extra function other than absolutely necessary was added to reduce the contract complexity.

7.4   Smart Contract Upgrades and Bug Fixes

By using ENS, new versions of the InterCrypto contract can be deployed and used by clients of the previous contract without the need for them to redeploy their contract. This allows InterCrypto to be upgraded if bugs or new features are added. The new InterCrypto contract will need to comply to the same public interface to remain backwards compatible, though some administration functions could be added safely.

7.5   Mainnet Deployment and Gas Optimization

The ShapeShift exchange operates using only mainnet blockchain network (as opposed to any testnets). This condition meant the InterCrypto smart contract would need to run on the Ethereum mainnet to be fully functional. Specifically, the ShapeShift deposit account as seen in Figure 12 that was watched for deposits needed to be on the mainnet.

This implied using mainnet Ether, which had a real-word cost of approximately £230 per Ether in August 2017. To reduce the cost of this academic project, as well as to reduce costs of other users of InterCrypto in the future, a set of gas optimization steps were applied to the InterCrypto contract as seen in Figure 17 and Figure 18.

The gas changes shown below were summed up over several iterations that may have had marginal overlap. The numbers should be considered approximate only.

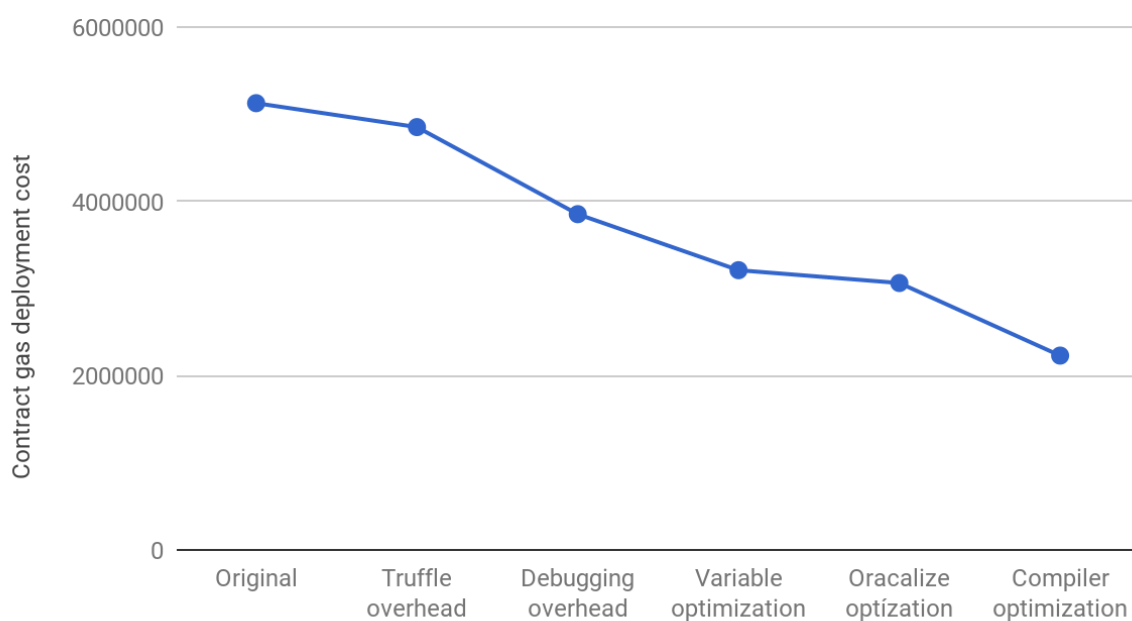**Effect on InterCrypto deploymet cost with smart contract optimization**

Figure 17: Deployment cost gas optimization

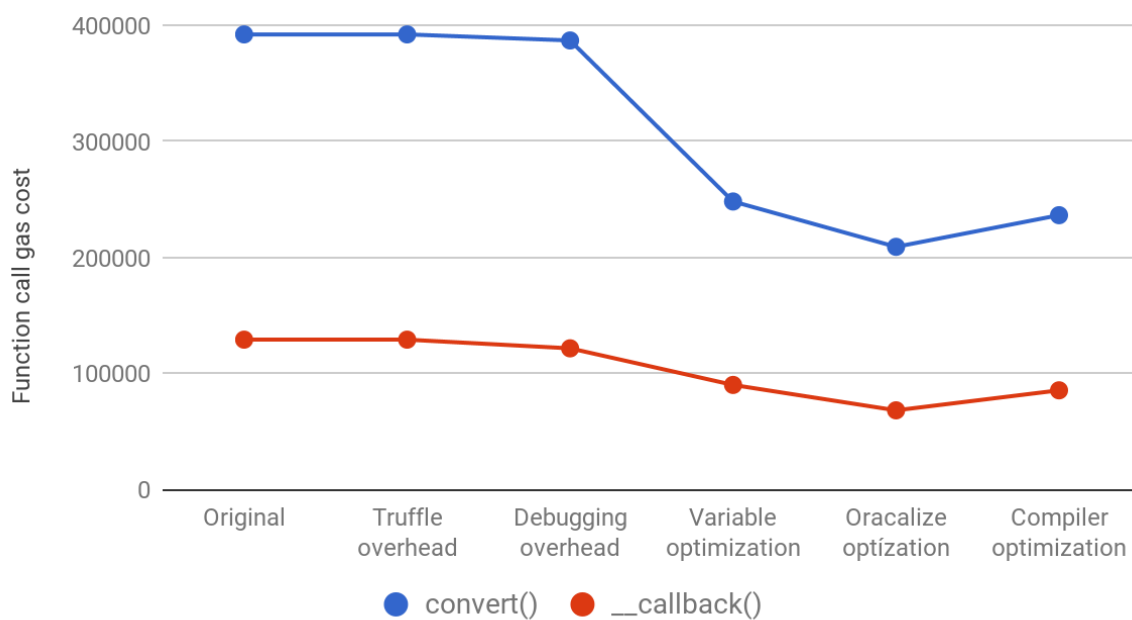**Effect on function call cost with smart contract optimization**

Figure 18: Function call cost gas optimization

The optimizations undertaken were:

1. **Truffle overhead** was the contract deployment overheads produced by the Truffle framework contract abstraction and deployment mechanism as discussed in Section 6.2.1_. These overheads made no change to function gas cost but reduced deployment cost by ~5%.

2. **Debugging overhead** was due to the events added to the code during development to allow for the inspection of variable values throughout execution as described in Section 7.3.4. The removal of these events had a minimal effect on function cost but reduced deployment cost by ~20%.

3. **Variable optimization** was conducted to minimize any unnecessary variable costs, especially storage variable (see Section 5.2.1) which came at an extremely high gas price. The main variable optimized was the *Conversion* struct which initially had six storage variables instead of the final two. This optimization reduced deployment cost by 17% and function calls *convert*() and *__callback*() by 35% and 26% respectively. The following strategies were applied:

   a. Reducing storage variables
   b. Using memory variables
   c. Using temporary pointer variables when references to mapping values when they are used multiple times in functions.
   d. Using private (not public) variables when adequate
   e. Indexing event parameters

4. **Oraclize overhead** was the unnecessary extra cost that the Oraclize API produced. A new, simpler and leaner custom interface was created for this project that marginally reduced deployment cost but further reduced cost of function calls *convert*() and *__callback*() by 16% and 24% respectively.

5. **Compiler optimization** was done by changing from Solidity compiler version 0.4.7 to version 0.4.15 as well as enabling the compiler optimization. Interestingly, it was found that this reduced the deployment cost by ~27% but increased cost of function calls *convert*() and *__callback*() by 13% and 25% respectively.

In addition to the above gas optimization, the cost of using Oraclize (they charge a small fee) was reduced by minimizing the *gasLimit* argument. This meant that less Ether was sent to the Oraclize smart contract, increasing the amount of Ether that was finally converted as seen in Table 10.

	Default	Final	
Oracalize *gasLimit*	20000	12000	Gas
Oracalize cost	0.0040355	0.0024426	Ether
	£0.93	£0.56	Pounds

Table 10: Oraclize cost reduction by minimizing the gasLimit *argument*

8. Exhibition Application

To exhibit the InterCrypto smart contract graphically and to create a realistic use-case client contract, several additional artefacts were developed which are described in this section of the report.

8.1 InterCrypto Wallet

A demo client smart contract was developed to exhibit and give an example of how and why the InterCrypto smart contract is used.

The demo client smart contract is an Ethereum contract wallet. Such wallets are used by users to store Ether offering extra security and other functionality that normal addresses do not. Often these smart contract wallets are built to only allow withdrawal if several actors, each with their own separately controlled Ethereum accounts, authorizes it. This is a useful technique to mitigate risk of Ether being stolen or misused by several members.

The demo smart contract, called the InterCrypto Wallet:
- Allows users to deposit Ether, and withdraw these funds in Ether or in another cryptocurrency of their choosing
- Uses a mapping of addresses to funds, so is safely usable by anyone to store Ether, not just the contract owner
- Can call InterCrypto.*recover*() to recover any funds that were lost by making incorrect calls to *convert*() or due to failure as discussed in Section 7.3.5.

The InterCrypto Wallet smart contract can be found at the following address on the blockchain:
- Ethereum mainnet: [0x34F61fF92Cd0F49A358A9fdedDe62898b9d70901](#)
- Rinkeby testnet: [0xD1C5f9a2A876927FCd23fAd29090EBE89d347c18](#)

8.2 InterCrypto Web Application

https://intercrypto.org

A web application was built at the above address that connects to the InterCrypto smart contract and the InterCrypto Wallet contract. Users are able to interact with either of these contracts by using the Metamask extension or the Mist browser. Figure 19 shows a screenshot of the InterCrypto website where Metamask is being used to interact with the InterCrypto smart contract.
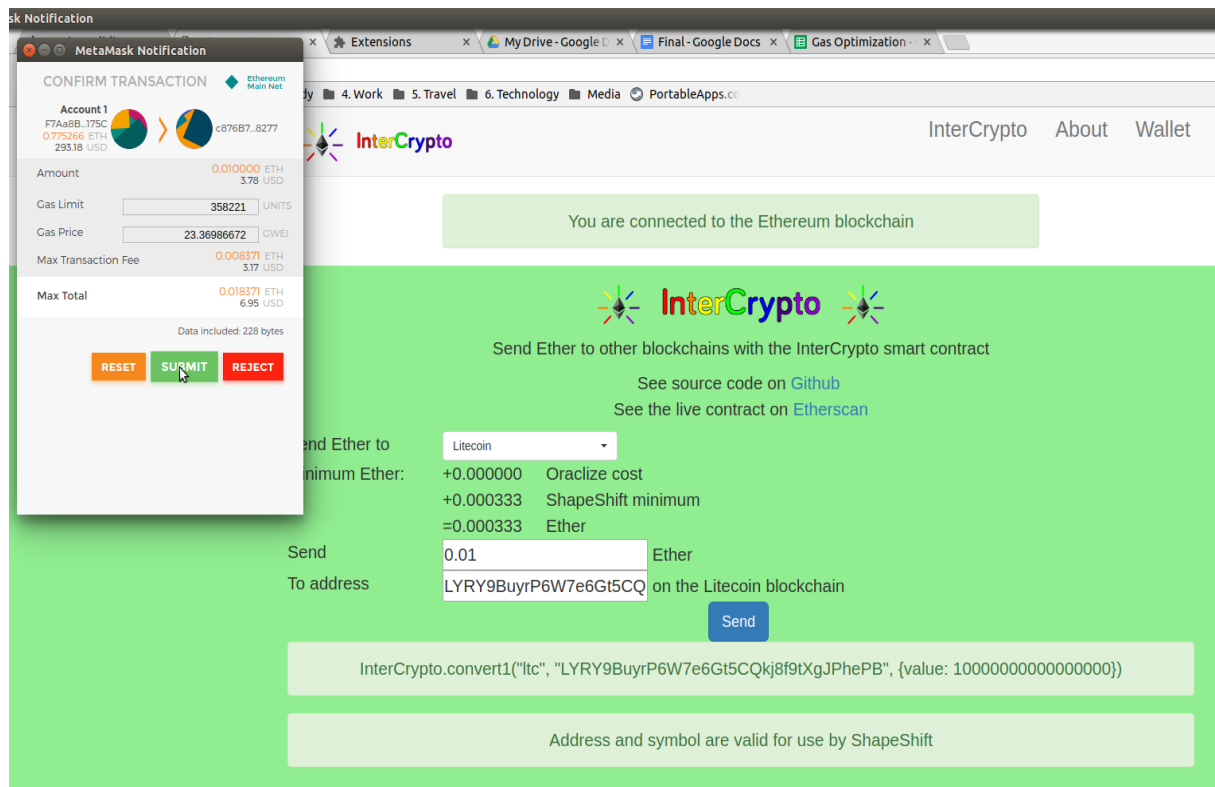
Figure 19: Screenshot of the intercrypto.org web portal with an imminent transaction using Metamask

The website is run on an Amazon Web Server elastic beanstalk instance and uses NodeJS for the back end. The application runs client-side only, the server merely acting to serve a static HTML page and supporting files such as JavaScript code, images and CSS style sheet. By running client-side only, users can be assured that their Ethereum keys are never sent to or used by intercrypto.org.

The web application resolves all Ethereum addresses using the ENS smart contract using the same domain names as the InterCrypto Interface contract, as explained in Section 5.6.

9. Evaluation

The InterCrypto smart contract was evaluated by testing and reviewing its final function, its capabilities to execute on-blockchain cryptocurrency conversions, it's security and the efficiency and cost of deployment and use. InterCrypto's final ability to provide a service to other smart contract was also evaluated.

9.1   Function

The InterCrypto smart contract successfully converts Ether to other cryptocurrencies.

Figure 20 shows the different states that the InterCrypto contract can have during one call to the *convert*() function with the following summary:

1. Three bad final states could be achieved by bad user input. This is not considered failure of the InterCrypto service as this is due to a user error.
2. Three bad final states could be produced through failure of the Oraclize or ShapeShift service. These were considered to be failures of the InterCrypto service.
3. The InterCrypto conversion service was successful if the final cryptocurrency conversion was completed leading to good final state 13.
4. The InterCrypto conversion service was successful if the user cancels the conversion before Oraclize responds to the query leading to good final state 6.
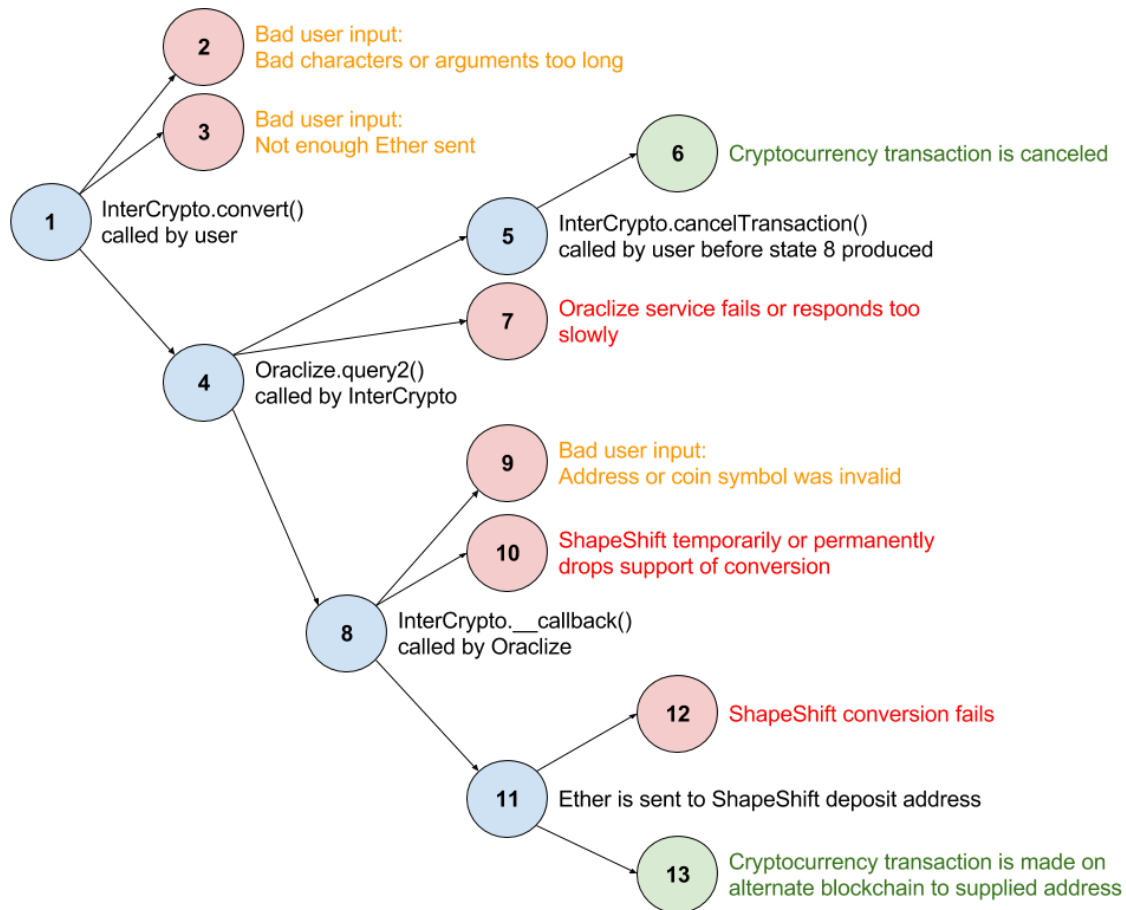
Figure 20: State transition diagram for InterCrypto call to convert*()*

In all bad final states, recovery of the Ether sent to the contract is possible, though from state 4 onwards, the amount recoverable is reduced by the Oraclize price.

All of the final states were produced except state 12. The failure of the Oraclize service (state 7) was observed once throughout the project, in which it was used hundreds of times. This is considered to be a very unlikely failure point.

The failure of setting up the ShapeShift conversion (state 10) was observed once in the project, in which it was used approximately one hundred times. This state can be reached because of an unplanned failure by ShapeShift or due to occasionally planned service stoppages such as during blockchain hard forks. This is considered to be a possible but unlikely failure point.

The failure of a ShapeShift conversion (state 12) was never observed, however this was only tested a handful of times during the project. It is considered that if ShapeShift correctly provide a deposit address and a deposit transaction to this address is made immediately after then it is unlikely that the conversion will fail. This is due to the strong reputation that ShapeShift has built based on this service. This is considered to be a possible but unlikely failure point.

55

The InterCrypto smart contract was tested automatically using the mocha testing package throughout the initial development phase of the project while the Truffle framework was being used. Testing at this stage exhaustively covered all InterCrypto states. Thorough manual but not automatic testing of the final InterCrypto contract was conducted at the end of the project.

9.2   Conversion Capabilities

The cryptocurrency conversion that InterCrypto supports is from Ether only to any cryptocurrency or token supported by ShapeShift. The number of supported coins by ShapeShift is was greater than 50 in August 2017, which covers a large range of the most popular cryptocurrencies and tokens. While this is considered to be a very successful amount of supported coins, it also exhibits the smart contracts dependence on what assets ShapeShift supports.

InterCrypto provides a one-way mechanism that allows conversion only from Ether. The solution does not provide a way conversion from other cryptocurrencies as shown in Figure 21.
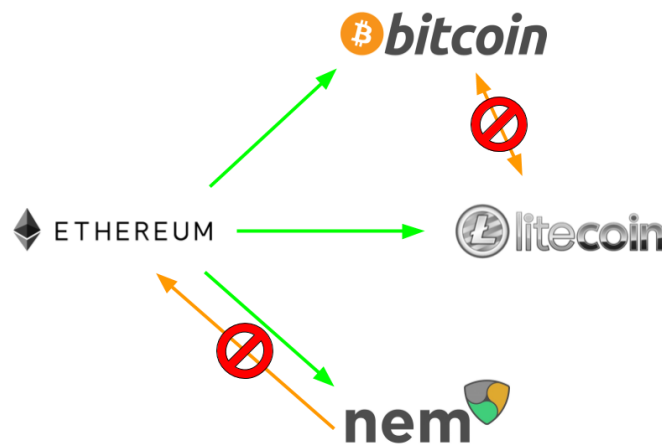
Figure 21: Graphical diagram of InterCrypto capabilities

9.3   Security

Due diligence was done to identify and mitigate bugs and security risks that are commonly found in smart contracts. While the contract is considered to be well checked, the potential for unknown issues is still present both due to the freshness of the smart contract space and due untested issues with the Ethereum blockchain and the Solidity language and compiler.

The greatest risk that the smart contract faces is that it allows for withdrawal of Ether funds without the permission of the rightful owner. This risk has been minimized by:
- Not holding Ethereum in the contract for prolonged periods of time unless the conversion fails or is cancelled.
- A safe recovery mechanism was implemented to allow for safe withdrawal of any recoverable funds in the case that conversion fails or the conversion is cancelled.

9.3.1 Code Review

To increase confidence in the InterCrypto service, smart contract expert Laurence Kirk[41] conducted a casual peer review of the InterCrypto contract, with no major bugs or security issues found. This review can be seen in Appendix C.

9.4 Efficiency

The final cost of the contract deployment and main function calls can be seen in Table 11. The conversion made was using the August 13 gas price of 20 GWei / gas and £230 Pounds per Ether.

Deploy	*convert*()	__*callback*()	
2229653	236255	85371	Gas
0.0446	0.0047	0.0017	Ether
£10.26	£1.09	£0.39	Pounds

Table 11: Final InterCrypto costs for deployment and function calls

Together with the Oraclize fee paid in Ether (see Table 10), this meant that the total cost of calling the *convert*() was 0.0040355 + 0.0046 = 0.0086 Ether (£1.98 pounds) plus the 0.65% commission that Shapeshift takes for the conversions.

The time taken for InterCrypto to set up a ShapeShift conversion is approximately 3-4 blocks (45 to 60 seconds). The time taken for ShapeShift to recognize the deposit and execute the conversion was observed to be just over two minutes. At this point, the recognition of the alternate blockchain transaction depends on the block time of that network.

9.5 Overall Service

The InterCrypto smart contract successfully provides a useful service to other smart contracts on the Ethereum blockchain that allows them to send Ether to other cryptocurrencies.

The InterCrypto interface contract allows users to connect to and use the InterCrypto contract without having to manually configure the connection. It also allows for a decentralized mechanism for the contract to be updated for bug fixes and function upgrades. The interface exposes a simpler set of functions so that client contract developers do not need to worry about all of the internal complexities of InterCrypto such as the administration functions.

---

[41] https://www.linkedin.com/in/extropylaurence/

10. Conclusion and Future Work

10.1 Summary of Achievements

This project thoroughly investigated mechanisms for on-blockchain cryptocurrency conversions. Current and in development smart contract and interoperable technologies were reviewed before a mechanism was chosen.

An Ethereum smart contract that uses a centralized oracle and a centralized exchange was developed and deployed to the Ethereum mainnet and testnet. Administration and recovery features were implemented as well as an interface contract that uses ENS to resolve the address of the InterCrypto contract. Major and minor security risks were mitigated in the contract and an external code review was done.

A demo client smart contract was built that successfully uses the InterCrypto contract and a front-end only web application was built that provides a user-friendly interface to the demo and InterCrypto smart contract and useful information and links to the open-source code and deployed contract.

The InterCrypto smart contract successfully converts Ether to a large variety of other cryptocurrencies and blockchain tokens. Failure points of the smart contract were identified as being low probability. These failure points exist due to the dependence on centralized off-blockchain services, meaning that the InterCrypto service is not fully decentralized. Recovery of Ether is possible for all failure states.

The cost of using the using the InterCrypto conversion feature is 0.0086 Ether (currently £1.98 pounds) plus 0.65% of the conversion. This non-trivial cost derives from the inefficient underlying blockchain technology itself, due to the Proof of Work consensus protocol, the cost of using an oracle as well as commissions taken by the Oraclize and Shapeshift services. Future upgrades of Ethereum to Proof of Stakes as well as other planned efficiency improvements should significantly lower these costs.

The InterCrypto mechanism does not allow for conversion to Ether, or to other cryptocurrencies from non-Ethereum blockchains. Research into interoperable technology using interledger networks is in progress and shows strong promise to allow for a more robust, truly decentralized mechanism that can provide multi-multi cryptocurrency conversion.

10.2 Improvements to InterCrypto

While the Ethereum smart contract is considered production ready, various works have been identified that would improve the InterCrypto service:
- Rebuilding the Truffle test suite to fully test the latest version of the InterCrypto service, including fuzz testing.
- Setting up bug bounties for smart contract vulnerability identification.
- A formal smart contract review, including formal verification and game theory analysis.

- Implementation of an administrative fail-safe function that can be used to check that the contract's Ether has not been accessed without permission.
- Implementation of a system that gives the InterCrypto users the ability to choose which oracle and exchange service they wish to use.
- Implement InterCrypto in a different way that uses a callback to the client contract, to allow for more automatic recovery that does not depend on watching events.
- Create an administration function that calls Oraclize function *setCustomGasPrice*(uint *_gasPrice*) to ensure that Oraclize callback will always be timely.

10.3 The Future of Inter-blockchain Technology

Ethereum smart contracts exhibited a powerful ability to execute arbitrary contract logic and to interact with other contracts. The project exposed the severe limitations of Ethereum, namely the non-trivial transaction costs of executing smart contracts and the isolation of the blockchain to access information outside of its own system.

It is hoped that planned efficiency improvements to Ethereym such as sharding and the implementation of Proof of Stake will greatly reduce these barriers for use.

The isolation of blockchains from each other creates a security vulnerability in cryptocurrency use as current exchange intermediaries run traditional centralized business structures. Technologies in the decentralized exchange space and interledger networks show strong promise to reduce such barriers and evolve the blockchain space from the current "smart contract" era to the "interoperable" era of blockchains, offering a vastly more powerful environment for new technologies and the benefits they bring to society.

11. References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 6 June 2017].

[2] Bitcoin.it, "Bitcoin Addresses," [Online]. Available: https://en.bitcoin.it/wiki/Address. [Accessed 19 July 2017].

[3] HashiCorp, "Serf Gossip Protocol," [Online]. Available: https://www.serf.io/docs/internals/gossip.html. [Accessed 17 August 2017].

[4] Ethereum Foundation, "Ethereum Yellow Paper," [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf. [Accessed 17 August 2017].

[5] D. Annamalai, "Blockchain – What is Permissioned vs Permissionless?," 10 January 2017. [Online]. Available: https://bornonjuly4.me/2017/01/10/blockchain-what-is-permissioned-vs-permissionless/. [Accessed 6 June 2017].

[6] BlockGeeks, "What is Blockchain Technology? A Step-by-Step Guide For Beginners," [Online]. Available: https://blockgeeks.com/guides/what-is-blockchain-technology/. [Accessed 31 May 2016].

[7] R. T. Smith, "Public and Private Blockchains: Enemies or Allies? Why the Enterprise Ethereum Alliance will prove the latter," [Online]. Available: https://medium.com/@rtylersmith/public-and-private-blockchains-enemies-or-allies-45f050c38fc0. [Accessed 4 June 2017].

[8] BlockGeeks, "Proof of Work vs Proof of Stake: Basic Mining Guide," [Online]. Available: https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/. [Accessed 13 June 2017].

[9] M. Vukoli, "The Quest for Scalable Blockchain Fabric:: Proof-of-Work vs. BFT Replication," 2015. [Online]. Available: http://vukolic.com/iNetSec_2015.pdf. [Accessed 20 August 2017].

[10] SmartContract, "About SmartContract," [Online]. Available: http://about.smartcontract.com/. [Accessed 5 June 2017].

[11] Blockchain Technologies, "Smart Contracts Explained: The Ultimate Guide to Understanding Blockchain Smart Contracts," [Online]. Available: http://www.blockchaintechnologies.com/blockchain-smart-contracts. [Accessed 8 June 2017].

[12] Etherscan, "Ethereum Transaction History," [Online]. Available: https://etherscan.io/chart/tx. [Accessed 6 June 2017].

[13] Blockchain Luxembourg S.A, "Bitcoin Confirmed Transactions per Day," [Online]. Available: https://blockchain.info/charts/n-transactions?timespan=all. [Accessed 6 June 2017].

[14] Bitcoin.it, "Bitcoin Script," [Online]. Available: https://en.bitcoin.it/wiki/Script. [Accessed 6 June 2017].

[15] R3 Ltd., "Corda Documentation," [Online]. Available: https://docs.corda.net/. [Accessed 12 June 2017].

[16] S. Higgins, "Mizuho Completes Blockchain Trade Finance Trial," 7 July 2017. [Online]. Available: https://www.coindesk.com/mizuho-completes-blockchain-trade-finance-trial/.

[17] Ethereum Foundation, "Ethereum Project," [Online]. Available: https://ethereum.org/. [Accessed 31 May 2017].

[18] Ethereum Foundation, "Solidity," [Online]. Available: https://solidity.readthedocs.io/. [Accessed 6 June 2017].

[19] L. Liu-Thorrold, M. Macdonald and R. Julien, "The Blockchain: A Comparison of Platforms and Their Uses Beyond Bitcoin," The University of Queensland, Brisbane, 2017.

[20] The Linux Foundation, "Hyperledger Fabric," [Online]. Available: https://www.hyperledger.org/projects/fabric. [Accessed 20 August 2017].

[21] The Linux Foundation, "Chaincode for Developers," [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode4ade.html. [Accessed 22 August 2017].

[22] NEO.io Foundation Ltd., "NEM The Smart Asset Blockchain," [Online]. Available: https://www.nem.io/. [Accessed 21 Augst 2017].

[23] NEO, "NEO Smart Economy," [Online]. Available: https://neo.org/. [Accessed 20 August 2017].

[24] NEO, "NEO Smart Contract Introduction," [Online]. Available: http://docs.neo.org/en-us/sc/introduction.html. [Accessed 20 August 2017].

[25] J.P. Morgan, "Quorum," [Online]. Available: https://www.jpmorgan.com/country/US/EN/Quorum. [Accessed 20 August 2017].

[26] L. Armasu, "J.P. Morgan's 'Quorum' Blockchain Platform Will Implement Zcash Privacy, Security Features," 22 May 2017. [Online]. Available: http://www.tomshardware.co.uk/jp-morgan-quorum-zcash-privacy,news-55680.html.

[27] Truffle, "Building DAPPs for Quorum: Private Enterprise Blockchains," [Online]. Available: http://truffleframework.com/tutorials/building-dapps-for-quorum-private-enterprise-blockchains. [Accessed 5 September 2017].

[28] Truffle, "Building DAPPs for Quorum: Private Enterprise Blockchains," [Online]. Available: http://truffleframework.com/tutorials/building-dapps-for-quorum-private-enterprise-blockchains. [Accessed 20 August 2017].

[29] RSK Labs, "About RSK," [Online]. Available: http://www.rsk.co/#about-rsk. [Accessed 7 June 2017].

[30] RSK Labs, "RSK Rootstock Platform: Bitcoin powered Smart Contracts," 19 November 2015. [Online]. Available: https://uploads.strikinglycdn.com/files/90847694-70f0-4668-ba7f-ddoc6bob00a1/RootstockWhitePaperv9-Overview.pdf. [Accessed 2017 June 2017].

[31] Waves Platform, "Waves Platform: Blockchain for the People," [Online]. Available: https://wavesplatform.com/. [Accessed 20 August 2017].

[32] S. Ivanov, "Waves Platform Whitepaper," [Online]. Available: https://wavesplatform.com/files/whitepaper_v0.pdf. [Accessed 20 August 2017].

[33] V. Buterin, "Chain Interoperability," 9 September 2016. [Online]. Available: http://www.r3cev.com/s/Chain-Interoperability-8g6f.pdf. [Accessed 19 June 2017].

[34] The Elements Project, "The Periodic Table of Elements," [Online]. Available: https://elementsproject.org/elements/. [Accessed 1 September 2017].

[35] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón and P. Wuille, "Enabling Blockchain Innovations with Pegged Sidechains," 22 October 2014. [Online]. Available: https://blockstream.com/sidechains.pdf.

[36] Ethereum Foundation, "BTC Relay: Frequently Asked Questions," [Online]. Available: https://btc-relay.readthedocs.io/en/latest/frequently-asked-questions.html. [Accessed 19 June 2017].

[37] Ethereum Foundation, "ERC20 Token Standard," [Online]. Available: https://theethereum.wiki/w/index.php/ERC20_Token_Standard. [Accessed 19 June 2017].

[38] EtheDelta, "Smart contract overview," [Online]. Available: https://www.reddit.com/r/EtherDelta/comments/6kdiyl/smart_contract_overview/. [Accessed 20 August 2017].

[39] W. Warren and A. Bandeali, "0x: An open protocol for decentralized exchange on the Ethereum blockchain," 21 February 2017. [Online]. Available: https://0xproject.com/pdfs/0x_white_paper.pdf.

[40] M. Oved and D. Mosites, "Swap: A Peer-to-Peer Protocol for Trading Ethereum Tokens," 21 June 2017. [Online]. Available: https://swap.tech/whitepaper/.

[41] H. Madden, D. Tee, L. Lunesu and L. Bussell, "openANX - Real World Application of Decentralized Exchanges," 20 June 2017. [Online]. Available: https://www.openanx.org/en/assets/whitepaper/openANX_White_Paper_ENU.pdf.

[42] S. Thomas and E. Schwartz, "A Protocol for Interledger Payments," [Online]. Available: https://interledger.org/interledger.pdf. [Accessed 19 June 2017].

[43] A. Liu, "Implementing the Interledger Protocol in Ripple," 6 October 2015. [Online]. Available: https://ripple.com/insights/implementing-the-interledger-protocol/.

[44] [. U. "Reddit Thread: Why does Ripple even need a blockchain if it's centralized?," [Online]. Available: https://www.reddit.com/r/Ripple/comments/62zoq8/why_does_ripple_even_need_a_blockchain_if_its/. [Accessed 1 September 2017].

[45] CryptoMiningGuy123, "Bitcoin Forum: Mining Ripple?," [Online]. Available: https://bitcointalk.org/index.php?topic=1393257.0. [Accessed 1 September 2017].

[46] G. Wood, "Polkadot: A vision for a heterogeneous multi-chain framework," [Online]. Available: https://github.com/polkadot-io/polkadotpaper/raw/master/PolkaDotPaper.pdf. [Accessed 15 July 2017].

[47] J. Kwon and E. Buchman, "Cosmos: A Network of Distributed Ledgers," [Online]. Available: https://cosmos.network/whitepaper. [Accessed 20 August 2017].

[48] J. Dilley, A. Poelstra, J. Wilkins, M. Piekarska, B. Gorlick and M. Friedenbach, "Strong Federations: An Interoperable Blockchain Solution to Centralized Third Party Risks," 6 January 2017. [Online]. Available: https://blockstream.com/strong-federations.pdf.

[49] J. Hosp, T. Hoenisch and P. Kittiwongsunthorn, "Cryptographically-secure Off-chain Multi-asset Instant Transaction network," [Online]. Available: http://www.comit.network/doc/COMIT white paper v1.0.2.pdf. [Accessed 1 September 2017].

[50] H. Barcelos, "StackExchange Question: How does Ethereum make use of bloom filters?," [Online]. Available: https://ethereum.stackexchange.com/questions/3418/how-does-ethereum-make-use-of-bloom-filters. [Accessed 25 August 2017].

[51] Oraclize Ltd., "oraclizeAPI_0.4.sol," Github.com, 2017.

[52] ShapeShift, "ShapeShift API," [Online]. Available: https://info.shapeshift.io/api. [Accessed 9 August 2017].

[53] Ethereum Foundation, "Web3 Javascript API," [Online]. Available: https://github.com/ethereum/wiki/wiki/JavaScript-API. [Accessed 9 August 2017].

[54] Ethereum Foundation, "Geth," [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/geth. [Accessed 9 August 2017].

[55] MetaMask, "MetaMask," [Online]. Available: https://metamask.io/. [Accessed 9 August 2017].

[56] Ethereum Foundation, "TestRPC," [Online]. Available: https://github.com/ethereumjs/testrpc. [Accessed 9 August 2017].

[57] Ethereum Foundation, "Remix IDE," [Online]. Available: http://remix.ethereum.org/.

[58] Z. "StackExchange Question: How can I see required hashrate to mine?," [Online]. Available: https://ethereum.stackexchange.com/questions/21319/how-can-i-see-required-hashrate-to-mine. [Accessed 13 August 2017].

[59] Etherscan, "Ethereum Network HashRate Growth Chart," [Online]. Available: https://etherscan.io/chart/hashrate. [Accessed 13 August 2017].

[60] J. Manning, "Ropsten To Kovan To Rinkeby: Ethereum's Testnet Troubles," 8 March 2017. [Online]. Available: https://www.ethnews.com/ropsten-to-kovan-to-rinkeby-ethereums-testnet-troubles.

[61] Parity Technologies, "Proof of Authority Chains," [Online]. Available: https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains. [Accessed 13 August 2017].

[62] J. Carlyle , "Ethereum Forum: Error: Stack too deep, try removing local variables," [Online]. Available: https://forum.ethereum.org/discussion/2400/error-stack-too-deep-try-removing-local-variables. [Accessed 13 August 2017].

[63] T. "What's the Solidity statement to print data to the console?," [Online]. Available: https://ethereum.stackexchange.com/questions/7139/whats-the-solidity-statement-to-print-data-to-the-console. [Accessed 1 July 2017].

[64] Ethereum Foundation, "Security Considerations," [Online]. Available: https://solidity.readthedocs.io/en/develop/security-considerations.html. [Accessed 20 August 2017].

[65] King of the Ether, "Contract Safety and Security Checklist," [Online]. Available: https://www.kingoftheether.com/contract-safety-checklist.html. [Accessed 15 August 2017].

[66] Ethereum Foundation, "Gas 1.0 Costs," [Online]. Available: https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid=0. [Accessed 30 May 2017].

[67] Amazon Inc., "Amazon Web Service Calculator," [Online]. Available: https://calculator.s3.amazonaws.com/index.html. [Accessed July 2017].

[68] Bitmain, "Antminer S9: World's Most Efficient Miner," [Online]. Available: https://shop.bitmain.com/specifications.htm?name=antminer_s9_asic_bitcoin_miner. [Accessed 15 July 2017].

[69] UKPower.co.uk Ltd, "Gas & Electricity Tariff Prices per kWh," [Online]. Available: https://www.ukpower.co.uk/home_energy/tariffs-per-unit-kwh. [Accessed 15 July 2017].

Appendix A.   Data Storage on Ethereum

Ethereum data storage cost		
Data cost	5	Gas per Byte
Ether cost	0.000000002	Ether per gas
GBP cost	£230	Pounds
Cost of data	£0.0000023	Pounds per Byte
	£2,300	Pounds per GByte
Amazon web server storage cost		
S3 data cost	£0.04	Pounds per GByte
Difference	5750000%	

Sources: [66] [67].

Appendix B.　　Bitcoin 51% Attack Cost

Bitcoin network hash rate (17 July 2017)	6,318,920	TH/s
51% of Bitcoin hash rate	3,222,649	TH/s
Antminer S9 hash power consumption [68]	10	GH/W
Power required to attack 51% of Bitcoin network	327	MW / s
	1177922	MW / h
Cost of electricity in England [69]	£0.12	GBP / kWh
Cost to attack 51% of Bitcoin network	£141,350,619	GBP / h

Appendix C.   InterCrypto Code Review

On August 13, Laurence Kirk, CEO and founder of Extropy.io[42] a company specialising in smart contract development, education and research, provided the following comments as a review of the InterCrypto contract. The review was casual but seemed thorough. Laurance did not perform this commercially and made no guarantee of this review.

The review is based on GitHub commit [e339cf1c8cef9214094b75a07333fbd4261ce442]():

On the whole it looks fine, no major problems I can see, though I haven't tested it.

In general, (and Oraclize should also do this more) use libraries and existing contracts, for example in many of the conversion functions have could be in a library.
You could also use the OpenZeppelin contracts as a base to inherit from, for example
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/ownership/Ownable.sol

None of the fields in your events are indexed, you may want to do that if you need to filter events, but it depends how you intend to use them.

line 31 you don't check the return value of oraclize_setNetwork();
line 84 should this function return an error value ? or will it always produce a valid address, in the calling function should you check for a zero address ?
line 148 For the currency symbols, you could put those in an enum, but you should also validate the input to that function for obviously wrong addresses.
line 163 should that be >= ?
line 171 I haven't checked what Oraclize returns as an id but you might want to check the test
if (myQueryId == 0) {
line 193 could that return say an error code, maybe you can you be more specific about a valid return length ?
line 229 I would probably also have an event here so that if the send fails the caller gets an indication rather than relying on the absence of an event

---

Appendix D.   ENS Auctions

The following steps were taken to obtain the ENS domain "jacksplace.eth". A similar in nature but much simpler process was undertaken to obtain the "jackdomain.test" domain on the Rinkeby testnet.

1. Generate node name of subdomain "jacksplace", later referred to as *JPnode*:
   web3.sha3('jacksplace') =
   "0x152abedaf0bc8c842ab939a872f1b5308fdaf64c0a4e85d2ec5eec80f9998dd4"

2. Connect to the ENS top level domain "ens"Auction Registrar contract at
   "0x0092e3d385193237637d7df64f47dda992d6a6b2"

3. Get my account address, referred to as *JackAccount*:
   "0xF7Aa8B7EfF5f640A8fF43C64CBe8eaF4c2B1175C"

4. Submit function call Registrar.shaBid(*JPnode*, *JackAccount*, "20000000000000000", "mybestsalt7") and obtain the result referred to as *sealedBid*:
   "0x60c4c844412f09f74a09f3d040dff8a65b1dda69c7a14f3437c663bfd7f48941"

5. Submit function call transaction with 0.25 Ether (used to mask the bid)
   Registrar.startAuctionsAndBid([*JPnode*], *sealedBid*).

6. Submit function call transaction Registrar.unsealBid(*JPnode*, "20000000000000000", "mybestsalt7")

7. Submit function call transaction Registrar.finalizeAuction(*JPnode*)

Appendix E.   Project Plan

This schedule was set out at the beginning of the project and had minor deviations throughout the project.

Date	Task
June 7	Investigation of Smart Contract Technology
9	Review of Ethereum Smart Contracts
14	Progress Report
21	Configure Programming Environment
24	Apply for Ethereum Funding
28	Configure Ethereum Nodes
July 5	Training for Ethereum Smart Contract Programming
8	Review Exchange Platform and Oracle APIs
19	Develop 1st draft of InterCrypto Smart Contract
30	Develop 1st draft of Demo Smart Contract
August 1	Investigation of Supporting Smart Contracts
8	Develop InterCrypto Smart Contract API
13	InterCrypto Smart Contract Audit
18	Final version of Demo Smart Contract
20	Develop demo Website
24	Final version of InterCrypto Smart Contract
September 7	Presentation of InterCrypto Project
8	Final Report