Uptrenda: A Peer-to-Peer Cryptocurrency Exchange

Matthew Roberts
matthew@roberts.pm
www.uptrenda.com

Abstract. A peer-to-peer cryptocurrency exchange would allow for cryptocurrencies to be traded directly between users without the security risks present in centralised exchange. Smart contracts provide part of the solution but suffer from transaction malleability. In this paper, I propose a simple solution to the malleability problem by introducing a dispute system that is both practical and secure. The dispute system allows a third-party to mediate with minimal involvement and for trades to be verified against an unspent quantity. When a trade has been verified, a server is used to partially unlock the coins and allocate them between contracts which are carried out directly using incremental multi-signature transactions. This process gives the owners full control over their coins and third-party mediation is only required if a contract is interrupted or if the participants are unable to reach consensus.

1. Introduction

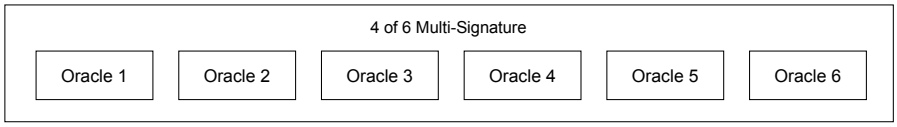
Since the release of the Bitcoin software in 2009, the popularity of Bitcoin has exploded, giving rise to a proliferation of new currencies and innovation [1]. Today, we use Bitcoin to pay for products and services, but increasingly, merchants are accepting so-called alternative currencies or “alt-coins” as a form of payment [2]. With such a huge increase in the use of cryptocurrencies, the number of currency exchanges that support them is also increasing, offering consumers new choices in currency exchange.

Currency exchange plays an import role in the world cryptocurrencies: it allows miners to sell their freshly minted coins and helps new people gain access to cryptocurrencies without having to mine them. It should be no surprise that some of these exchanges process a lot of money. As of the time of this writing, the world's largest exchanges process millions of dollars’ worth of cryptocurrencies each month, and the demand for new currencies in 2015 is only going to rise [3-4].

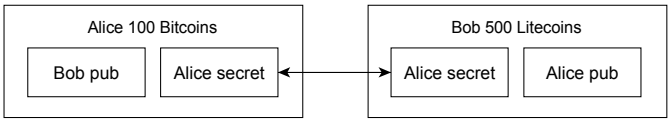
This is worrying. With so many new people hearing about Bitcoin, the number of exchanges will increase even further and so will the number of criminals targeting them [5-6]. What is needed is a new approach that accepts the unique properties of these currencies, and their flaws, as well. The model I propose is a peer-to-peer design that keeps the coins in the hands of the owners and allows a third-party to be introduced with minimal involvement. Strangers are free to trade without the direct use of a third-party by using micro-collateral and breaking down trades into a series of increasingly larger transfers. Mediation is still required; however, this is only relevant if a micro-payment channel is improperly closed, or transaction malleability has occurred, thus rendering attacks uneconomical.

2. Related Work

Early work toward building a decentralized cryptocurrency exchange involved devising networks of oracles that could be used to allocate coins between multi-signature addresses. This is the approach taken by Coinsigner [7], Multigateway [8], Metalair [9-10], Orisi [11], and Codius [12]. Most of these services rely on trusting a subset of oracles, which has the obvious disadvantages of relying on a trust-based model. For example, if the oracles were to collude, they could extort the owners for a cut. In response to this issue, distributed oracle services like Orisi sought to increase the number of oracles used to carry out contracts as well as to introduce a reputation system. This approach helps to reduce the possibility of collusion attacks, but it does nothing to solve the deeper problem of the owner having no control over how their funds are secured.



A much better solution is to use smart contracts to transfer coins directly between participants. Early work using smart contracts to solve trust problems was done independently by Noel Tierman and Mike Hearn, who focused on the use of cross-chain contracts and micro-payment channels to transfer coins between blockchains [13-14]. The biggest advantage of using smart contracts is that they require no trust. A smart contract can be used to trade coins without the need for an oracle. This is a much better alternative to the trust-based model. Unfortunately, such smart contracts suffer from transaction malleability and remain largely incompatible with the validation rules used in production networks due to their heavy reliance on non-standard transactions.

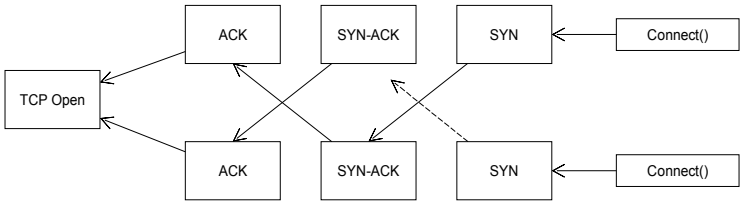


An enhanced form of cross-chain contracts was proposed by Noel Tierman on May 2014, which helped address some of these issues [15]. The new scheme only requires non-standard transactions to be used for one of the coins and thus makes it possible to trade arbitrary currency pairs by using an intermediary coin as a proxy. Additionally, the scheme also addresses the transaction malleability problem by introducing a third-party oracle who can be used by the owner to refund coins. The third-party oracle is only required to protect against transaction malleability and can be safely removed once the transaction malleability problem was fixed.

This proposal does a good job at describing a more practical exchange mechanism but is still tedious to use in practice, even with only one side requiring non-standard transactions. And worse still—there is no mention of the impact of a compromise anywhere within the proposal. If the oracle is compromised, then there is a non-trivial risk that the attacker may be able to double-spend their deposit prior to claiming the hash-locked coins of the opposite side [16]. This paper therefore presents a solution to these problems using a micro-payment channel system similar to Coinffeine's trustless exchange algorithm [17] but based on a different dispute process.

3. Network

Before nodes can communicate, there must be some way for them to find each other, a process known colloquially as “bootstrapping.” Bootstrapping is done by using a rendezvous server that keeps track of two main types of nodes: passive and simultaneous nodes; A passive node is any node that can receive inbound connections directly; A simultaneous node is a node that can't receive inbound connections from regular nodes but can receive connections from other simultaneous nodes through the use of TCP hole punching.



Hole punching works by predicting the source port that a router will map for an outgoing connection and then initialising two simultaneous connections at the exact same time to their respective source ports. This process tricks the individual routers into letting the SYN three-way handshake cross, thereby spawning a new connection. Most routers either increase (m = s + n) or preserve (m = s) the source port which makes mappings easy to guess. The port mapping can be predicted so long as they can be observed by a third-party like a remote server.

In practice around 63% of routers in the wild already have a predictable type NAT [18-19], and many of the most popular brands are even more predictable at 80% or higher [20]. These results clearly demonstrate that TCP hole punching is an effective traversal technique, which is a huge advantage for peer-to-peer networks as the direct interaction means that certain kinds of traffic don't have to be broadcast over the whole network, saving valuable bandwidth.

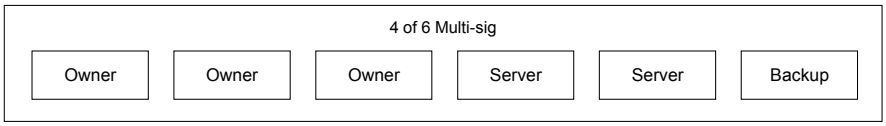
4. Order Book

The order book uses a Bitmessage-style proof-of-work for all messages that need to be broadcast over the whole network [21]. These messages currently include OPEN_ORDER messages as well as match status messages and nodes only relay them if they contain a correct proof of work. If the proof-of-work is correct, an OPEN_ORDER allows a node to store a new trade within the order book, which functions as a simple distributed, replicating database.

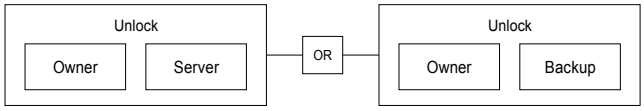
When nodes connect to the network, their neighbours are used to synchronise the order book, keeping the list of trades up to date. To protect against order book spam, each trade has an expiry and a TXID of an unspent output. The trade is also signed with the ECDSA key of the unspent output, which allows nodes to be able to independently verify that a trade hasn't been spoofed. If an open trade doesn't have a valid output, signature, proof-of-work, or timestamp; it gets pruned or rejected, and trades can be validated both before and after synchronisation.

5. Green Address

Green addresses are multi-signature addresses that can be used to prove that the coins associated with a trade are unspent and can't be moved. The primary purpose of a green address is to reduce the risk of double spend attacks prior to contract funding. This is designed to prevent malicious nodes from being able to trick other nodes into funding contracts when they don't intend to proceed with it. Thus, green addresses effectively make the time cost to open a contract the same for an attacker as for any other user. The construction of a green address is given bellow.



The three keys denoted by "Owner" belong to the owner of the coins; the server key is for micro-dispute resolution and partial matching; and the backup key is a time-lock encrypted fail-safe. Notice that in this scheme, the owner's signature is required under every possible unlock scenario. It means that even in a worst case scenario, the attacker is still missing the required leverage to be able to move funds, so the owner always has full leverage over their coins.



This is the mechanism that makes it possible to safely introduce a third-party and allows other nodes to validating a trade against an unspent output. Consequently, it is far easier to keep spam out of the order book while still providing enough leverage for a server to be able to punish nodes who break a contract as well opening up the possibility for partial matching. And green addresses are also resistant to failures: if the server key is lost, the time-lock encrypted backup still allows for the future independent recovery because the initialisation vectors for the encryption are stored directly within the client.

Theoretically, the scheme also allows for both an owner key and a server key to be lost without losing the ability to unlock the coins. This is easy to prove:

```
Unlock = Owner + Owner + Server + Backup

Unlock = Owner + Owner + Owner + Backup

Unlock = Owner + Owner + Server + Server
```

The scheme is extremely resistant to failures, which makes it naturally suitable for multi-factor authentication where a certain portion of the owner keys could be distributed between a laptop and a mobile, and the remaining key could function as a secure offline backup. The full range of unlock possibilities is given by the following equation:

```
Unlock = Owner AND (Server OR Backup OR (Server AND Backup))
```

6. Trades

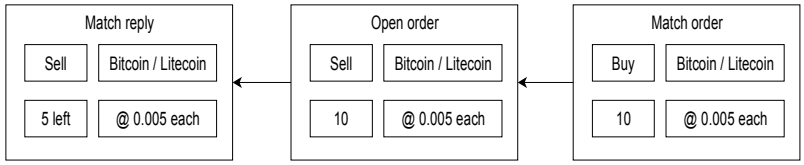
All trades have a uniquely associated green address, which is referenced by including the TXID of the green address' deposit transaction. A trade is opened on the network by forming a correct OPEN_ORDER message, which must be signed using the ECDSA key of the first owner key used in the green addresses as well as a correct proof-of-work. Each trade message includes the following information:

- Amount
- Price per coin
- Routes
- Receive address
- NTP timestamp
- Contract public keys
- Currency pair e.g. bitcoin / litecoin
- Deposit TXID, proof-of-work, and ECDSA signature

Routes is a list of ways to communicate directly with the node that opened the trade. Receive address is where they want to receive their coins. Proof-of-work is a Bitmessage-style proof-of-work. And NTP is an NTP adjusted timestamp. If all of these fields are present and correct, the trade will be allowed to enter the network's distributed order book, leaving other nodes to send compatible matches.

Matching

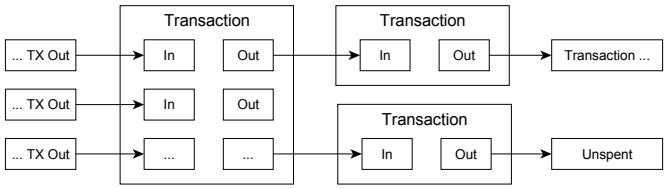
Nodes are free to make offers to any open order they like. When a node finds a compatible order, it sends a MATCH_ORDER message. The message describes how much of the trade the node is interested in filling and references an existing open order used to fund the match. In response, the recipient of the match replies with a calibrated amount, which indicates how much they can fill as portions may have already been allocated to other traders. The traders then handshake to agree on the final calibrated amount and broadcast the handshake (a new MATCH_ACCEPT) message to the network. The message tells the nodes to deduct some of the reserved coins under their individual open orders which stops double spends. A match can also be undone by broadcasting an expiry which is only accepted after a certain time has elapsed.



This unreliable nature of matching makes calculating exchange rates from open orders purely speculative. To solve this problem, a 24 hour average of past trades can be used together with a trimmed average from external sources. This process won't show the economics actually taking place within the exchange, but it at least ensures that traders won't be misquoted inaccurate prices. In any case, a trader is always free to set their own exchange rate.

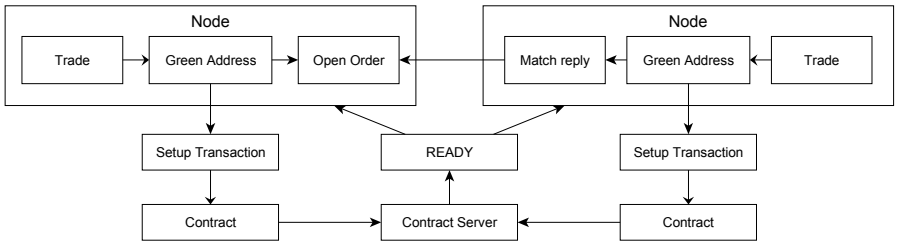
7. Contract Server

Bitcoin and Bitcoin-like currencies define transactional value as a chain of inputs and outputs [23]. Each transaction within Bitcoin must include at least one input which describes the source of the coins. The inputs come from previous transactions and must be sent in full and signed with the necessary ECDSA keys. After including and signing the inputs, the transaction must specify where and how the coins should go— this is the output. There can be multiple outputs per transaction that remain unspent until they're used as the input in some future transaction.



The transaction system works well for simple use where the inputs and outputs are unlikely to need changing, but in a peer-to-peer currency exchange that requires partial matching and atomic transfers, the situation isn't so simple. For example, if inputs must be sent in full and transactions are chains of past transactions, how do you support fast partial matching? Depending on how the outputs were split-up in your wallet, a single match for only a portion of your trade could put up to 100% of your money in a pending state until the transaction is accepted in the blockchain (usually recommended as 6 confirmations or 60 minutes between each match!)

You could solve this problem by waiting and then queueing up all relevant matches, allocating each a portion of your coins in a single transaction, but then what happens if one of those matches times out or they decide not to go through with it? There's no way to know who intends to send what, and it would be trivial for an attacker to exploit this and tie up money. The solution is to use a contract server that dynamically indicates the status of a contract's setup.

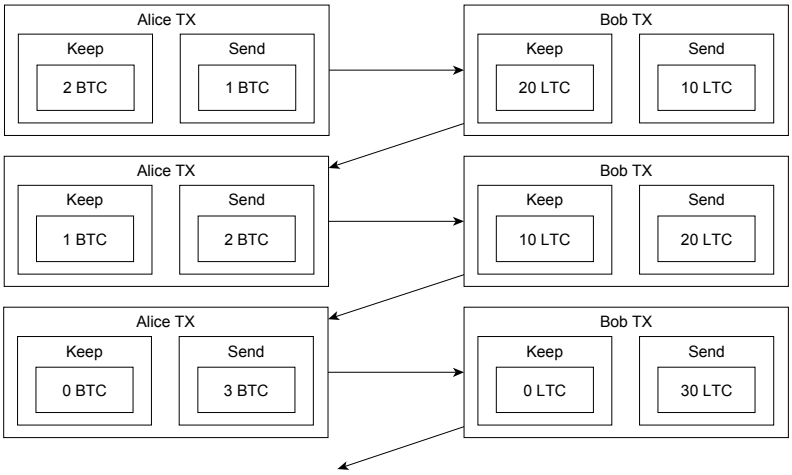


The contract server takes a setup transaction that spends from a green address and indicates to nodes in real time what the status is. It also indicates when a contract is about to expire, which gives nodes a chance to reallocate outputs. A setup transaction can have multiple outputs. Each output represent one side to a contract, and they are locked when both sides have been allocated. When all outputs are locked, the server signs the setup transactions and sends out a READY message to all affected parties. This lets the nodes know the final (signed) version of their setup transaction so they can proceed with the protocol.

8. Contracts

Coins are transferred directly between nodes using a type of smart contract known as a double-sided micro-payment channel [14]. Micro-payment channels work by breaking up a multi-signature deposit into a series of chunks that progressively serve to transfer the coins. The transfer starts out by only allocating a small portion of the total coins to the other side, which is then gradually increased in response to what they send. Each increase is therefore required to receive a higher amount, and the process is rapidly repeated until all coins have been transferred.

The beauty of using micro-payment channels is that rather than having to broadcast hundreds of individual payments, micro-payment channels work on the transaction level by creating new versions of a transaction, and the final version only needs to be broadcast once. This saves a huge amount in transaction fees and confirmation time, allowing even large amounts of coins to be transferred efficiently. An example of trading Bitcoin for Litecoin is given:



Both sides of the trade start out by creating a setup transaction that allocates coins to a multi-signature address for each of their matched contracts. The setup transaction also includes a fee section, which allocates any trade fees plus a small micro-collateral amount to the exchange. For example, if the contract requires adjusting the transaction by an additional coin each time, then the micro-collateral for that contract is 1 coin. Here is the fee output equation.

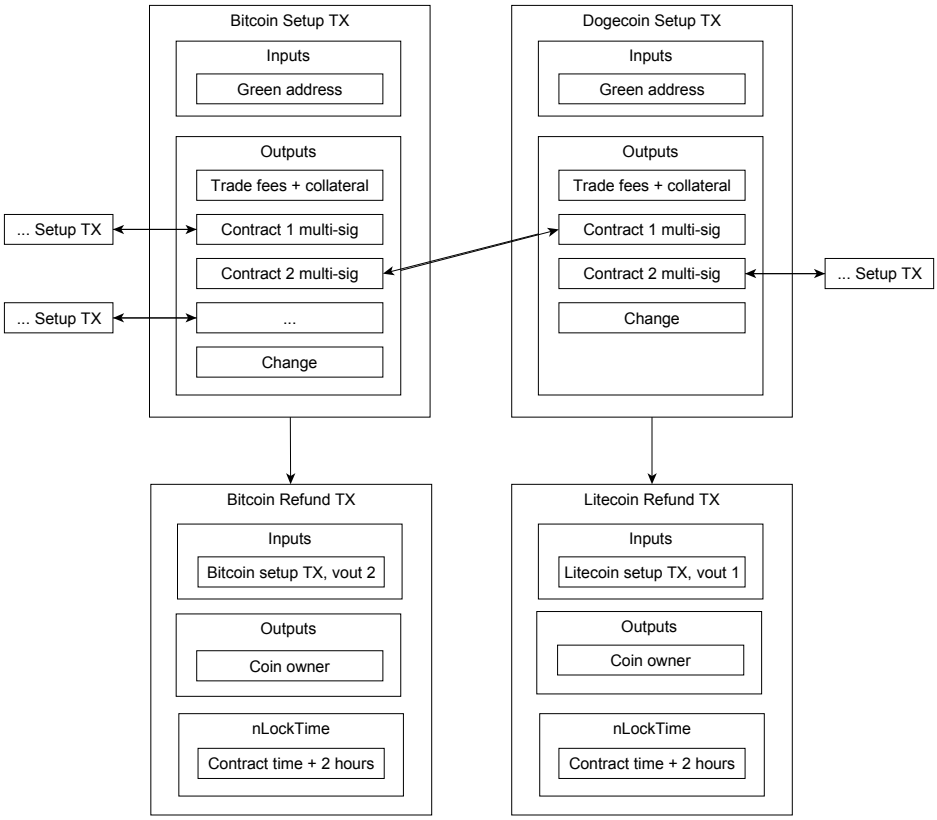
```
collateral = (contract_amount * 0.0005) + (contract_amount * 0.0005) + ...
fee_output = collateral + ((total_coins - change) * trade_fee)
```

Micro-collateral is non-refundable and factored into the fees for the exchange. In the event that a broken micro-payment channel causes a trader to lose an expected micro-increase, the micro-collateral of the other trader can be used by the contract server to reimburse the owed coins as well as to punish the side who broke the contract. Thus, this approach ensures that only a micro-portion of coins are ever in dispute in any given transfer, and the quantity is small enough that even a catastrophic hack would cause minimal damage.

9. Refunds

When the setup transaction is broadcast, it will cause coins from the green address to be locked up in a contract's multi-signature address. The address then becomes a possible attack scenario: if the coins can only be moved from the multi-signature address by using micro-payment channels, then what happens to the coins if either party decide not to comply? The coins could be locked up indefinitely, leading to a possible extortion situation: I'll sign your transaction for a cut.

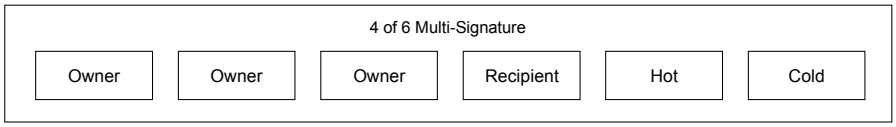
Time-locked refunds offer a trustless solution to this problem. To use time-locked refunds, a trader asks the recipient to sign a time-locked transaction, which transfers the coins from the contract multi-signature address back to the trader at a future date.



The refunds remain valid so long as the setup transaction ID hasn't been changed by a rogue miner or modified in transit before it reaches a miner's memory pool. Such an attack would be difficult to pull off on the present Bitcoin network but remains possible in theory [16]. If an attacker does manage to mutate the setup transaction, then the dispute process still offers a way out by allowing a third-party to allocate a refund. The next section looks more at this process.

10. Contract Outputs

Contract outputs are sent to a 4 of 6 multi-signature address which gives majority leverage to the coin's initial owner. Of the 6 keys required: 3 belong to the coin's owner, 1 belong to the recipient, and 2 belong to the exchange. The recipient key is an ECDSA threshold key spanning over multiple devices. The exchange keys consist of a cold and hot key. The cold key is a time-lock encrypted fail-safe, and the hot key is a special key derived by combining the public keys of multiple independent oracles with an offline ECDSA key belonging to the exchange.



The unlock possibilities can be described by the following equation:

```
Unlock = Owner AND (Recipient OR Exchange OR (Recipient AND Exchange))
```

Much like the scheme proposed for a green addresses, this scheme is able to offer a certain level of fault-tolerance in the event of a lost key. Either the owner or exchange can lose a key and still have the leverage needed to be able to allocate a refund without relying on the recipient. And much like with the green address, the hot key has a time-lock encrypted fail-safe - the cold key. These properties, together with the hot key, allow the exchange to prove that the money cannot be moved or refunded during a contract because the hot key's oracles only release their keys at a future date so the private key can only be created after the double spend window expires.

Assuming a worst case scenario where the exchange is operated by an attacker and so are the contract's owner keys—the ECC addition of multiple external third-parties' public keys (like RealityKeys [25]) with the exchange's offline ECDSA key (stored using trusted computing) can be used to prove that it would be highly improbable for an attacker to double spend as they would have to control potentially hundreds of unrelated systems and hack past an air-gap. Furthermore, the window for an effective double-spend would be limited to occurring during a micro-transfer, and the protocol strictly limits the attacker to only being able to steal as much value as he currently has (assuming an absolute worse case scenario.)

Fault-tolerance

The present scheme already provides a level of fault-tolerance for the hot key in the sense that the cold key can be used as a backup, but the same level of reliability doesn't exist for the owner keys. In order to provide this level of reliability, the third owner key can be purposely constructed such that the absence of any one owner key can be derived from the remaining keys. To express how this works, a simple XOR scheme is employed (similar to a one-time pad):

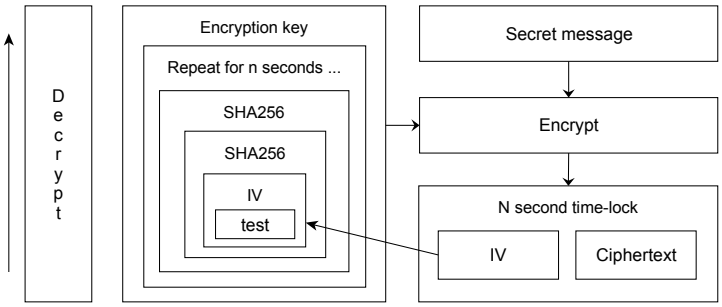
```
private_key_c = private_key_a XOR private_key_b

private_key_b == private_key_c XOR private_key_a

private_key_a == private_key_c XOR private_key_b
```

10. Time-lock Encryption

Time-lock encryption is a secure way to send messages to the future [22]. To use time-lock encryption, you start out by choosing a random value (IV) and then hash it multiple times for however long you want your time-lock to last. The final hash formed from the successive chain of hashing then becomes the key used to encrypt your message, and the initial starting value can then be given out to force an individual to have to repeat the work used to derive the key.



Since a time-lock encrypted ciphertext can be safely given out to anyone, this has extremely important consequences for reliability in decentralized smart contract systems. I spoke before about time-lock encrypted backup keys in the green address and contract output sections, and what I mean by this is that the ECDSA private key for both backup keys have been time-lock encrypted to produce a ciphertext and IV, which is then directly embedded within the client. The private key can then be discarded since there's no advantage to the exchange keeping it, and there will always be a fail-safe that can be used to recover funds even if the exchange disappears.

The process of time-lock encryption can also be outsourced to multiple third-parties in a threshold scheme to allow the oracles used in the construction of the hot key (contract output section) to have a similar level of reliability. This would be safe to do because the integrity of the hot key is a t of n scheme where all private keys are required (t = n), and the exchange stores its key offline (so there would always be that safe-guard if the time-lock providers were all compromised.)

Threshold time-lock encryption

To understand threshold time-lock encryption, consider a list of external time-lock providers `N` who publish a PGP public key that can be used to time-lock encrypt an arbitrary plaintext (the private key associated with the public key is time-lock encrypted, and when that time-lock has been broken, the resulting private key can be used to decrypt all ciphertexts created with the public key.) Now let us assume a threshold `T of N` can be trusted and that within any subset of `N` you cannot know whether the set contains some or all of the untrusted parties `N - T`.

Construction of a time-lock encryption threshold scheme then involves wrapping a plaintext in multiple layers of encryption and repeating this for every combination of time-lock provider such that the number of providers used in each repetition is always greater than `N - T`. This means that within any matryoshka doll of encrypted ciphertexts, there is always at least 1 assumed legitimate time-lock protecting the plaintext, and thus, the time-lock can never expire prematurely.

The disadvantage of this scheme is obviously the increased time-frame of decryption. If there are `N` time-lock providers, the total number of time-lock encrypted matryoshka dolls is given by:

```
matryoshka_dolls = ((N - T) + 1) * N
```

... which can be multiplied by `T` to find the total number of time-locks required. In a 3 of 4 scheme this number would be 24. Now multiply that by the individual amount of time for each time-lock —let's say 1 hour. The minimum time-lock would be `T * time_lock` (assuming the first matryoshka doll tried was valid), and so the full range is between:

```
(T * time_lock) and ((matryoshka_dolls * T) * time_lock).
```

As it happens, this range works perfectly for threshold time-lock encrypted oracles as a 1 hour time-lock can be used to give the nodes chance to complete a micro-payment channel, which would allow the oracle's private keys to be automatically recoverable within a business day if a catastrophic failure were to occur. Thus oracles with improper backup procedures would not be able to break the all or nothing scheme used to derive the hot wallet key. As for the time-frame required to encrypt the green address backup key or the contract output backup key (cold key), it would be much longer on the order of several months to a year.

The backup key is used only as a fail-safe from disasters. There are already regular backups of the server key used in the green address, and all oracles used to construct the hot key should be run by legitimate companies who presumably have backup procedures of their own. So having to rely on time-lock encrypted backups is only used as a last resort.

Other improvements to time-lock encryption:

- In addition to a randomly generated nonce for the IV the time-lock providers could also include a list of news headlines in that they hash. The IV would be able to provide an estimation of the age of the time-lock by proving that the time-lock can't have been created before a certain date.
- It is possible to add checksumming to the hash chains e.g. by hashing the result of every N and N - 1 hashes, taking the first 4 bytes of the result, and publishing a list of partial checksums along with where they occur in the chain [26]. This information provides information for error detection but more work needs to be done prove that its secure.
- It is possible to generate time-lock encryption keys in parallel by generating multiple chains simultaneously and using the final hash of chain 1 as the encryption key for chain 2, the final hash of chain 2 as the encryption key for chain 3 and so on [22]. This approach has been implemented by Peter Todd, who also used Bitcoin to incentive that the chains would be eventually unlocked by attaching Bitcoin private keys [27].
- Time-lock encrypted PGP public keys allows for an unlimited number of time-locked ciphertexts to be created without having to repeat the encryption process each time. This allows the functionality of time-lock encryption to be outsourced to a third-party which avoids having to do any specialised computations yourself, and a threshold scheme can be used to improve the overall security.

11. Conclusion

In this paper I have presented a practical system that not only reduces the amount of trust required during cryptocurrency exchange but also improves the security of the whole process. I started out by examining smart contracts and demonstrated that existing schemes are impractical, insecure, and suffer from transaction malleability. To solve this problem, I adapted micro-payment channels by adding a dispute system that doesn't rely on Script or non-standard transactions to operate. The dispute system solves the transaction malleability problem by allowing coins to be refunded by a third-party without giving them full control over coins. The dispute system also guards against time wasting by validating transactions and offering a simple way to recover from broken micro-payment channels. Finally, time-lock encryption was also introduced as a way to ensure that third-parties can disappear with minimal consequences to existing contracts.

12. Future Work

- Future advances in cryptography will allow most of the functions currently entrusted to third-parties to be conceptualised as decentralised systems requiring little trust. For example, advances in indistinguishability obfuscation could allow an exchange to be able to encrypt a program that signs refunds (complete with an embedded private key) and give the resulting ciphertext to its customers [28]. The customers could then verify that the program worked before making a deposit, and the encrypted function would presumably mean that the private key could never be recovered.

- There is currently no standard for the universal, decentralised transfer of cryptographic assets between unrelated systems— a problem leading to a proliferation of new assets that can't be directly traded. I would suggest that a solution like the one presented in this paper would work well for trading divisible assets whereas any trade using indivisible assets would be best left to a distributed oracle system—make it happen! If software has an interface to support the above ideas, then universal trading would be possible.

- External time-lock encryption services currently don't exist, and their presence could improve the security of countless unrelated systems. There is also no implementation of parallel time-lock encryption for GPUs, which could be used to efficiently generate a lengthy time-lock in a fraction of the time that it would require to actually open it.

- Threshold ECDSA has only recently been invented, and the only implementation available is in Java [29]. Since such a scheme can be used to considerably improve the security of Bitcoin transactions, it would be useful to have some implementations in other languages. Additionally, it would be useful to have a threshold scheme based on the additive properties of ECDSA public keys. It is currently very easy to do: pub_a + pub_b = pub_c as a simple way to derive a compound key without a trusted dealer; however, the approach requires all the private keys to be revealed, which is insecure and unreliable.

- In the future the design in this paper could be fully decentralised which would effectively reduce any central responsibility to that of maintaining the software and swapping out expiring time-lock encrypted backups with new locks.

12. References

[1] Coinmarketcap.com,. 2015. 'Crypto-Currency Market Capitalizations'. http://coinmarketcap.com/.

[2] Bitcointalk.org,. 2015. 'ALT Coin Stores And Services List'. https://bitcointalk.org/index.php?topic=273148.0.

[3] Hajdarbegovic, Nermin. 2014. 'Bitpay Now Processing $1 Million In Bitcoin Payments Every Day'. Coindesk. http://www.coindesk.com/bitpay-now-processing-1-million-bitcoin-payments-every-day/.

[4] Blockchain.info,. 2015. 'Bitcoin Number Of Transactions Per Day'. https://blockchain.info/charts/n-transactions.

[5] Moore, Tyler, and Nicolas Christin. 2013. 'Beware The Middleman: Empirical Analysis Of Bitcoin-Exchange Risk'. Springer, 25--33.

[6] Bitcointalk.org,. 2015. 'List Of Major Bitcoin Heists, Thefts, Hacks, Scams, And Losses [Old]'. https://bitcointalk.org/index.php?topic=83794.0.

[7] Coinsigner.com,. 2015. 'How It Works'. http://www.coinsigner.com/Howitworks.html.

[8] Multigateway.org,. 2015. 'Multigateway'. http://multigateway.org/.

[9] Dixon, Julia. 2013. 'True Peer-To-Peer Currency Exchange? - DGC'. DGC. http://www.dgcmagazine.com/true-peer-to-peer-currency-exchange/.

[10] reddit,. 2013. 'Proposal: Fully Decentralised Exchange Mechanism For All Cryptocurrencies And Fiat. • /R/Bitcoin'. http://www.reddit.com/r/Bitcoin/comments/1f7p7z/proposal_fully_decentralised_exchange_mechanism/.

[11] GitHub,. 2014. 'Orisi/Wiki'. https://github.com/orisi/wiki/wiki/Orisi-White-Paper.

[12] GitHub,. 2014. 'Codius/Codius'. https://github.com/codius/codius/wiki/Smart-Oracles:-A-Simple,-Powerful-Approach-to-Smart-Contracts.

[13] En.bitcoin.it,. 2015. 'Contracts - Bitcoin'. https://en.bitcoin.it/wiki/Contracts.

[14] Bitcoin.org,. 2015. 'Developer Guide - Bitcoin'. https://bitcoin.org/en/developer-guide#micropayment-channel.

[15] Tiernan, Noel. 2014. 'Tiernolan/Bips'. Github. https://github.com/TierNolan/bips/blob/bip4x/bip-atom.mediawiki.

[16] Karame, Ghassan, Elli Androulaki, and Srdjan Capkun. 'Two Bitcoins At The Price Of One? Double-Spending Attacks On Fast Payments In Bitcoin.'.

[17] GitHub,. 2014. 'Coinffeine/Coinffeine'. https://github.com/Coinffeine/coinffeine/wiki/Exchange-algorithm.[18] Guha, Saikat, and Paul

Francis. 2005. 'Characterization And Measurement Of TCP Traversal Through Nats And Firewalls'. In , 18--18.

[19] Lai, Yi-Wen, and KuangFu Lai. 'Implementing Nat Traversal On Bittorrent'.

[20] Ford, Bryan, Pyda Srisuresh, and Dan Kegel. 'Peer-To-Peer Communication Across Network Address Translators.'. In .

[21] Warren, Jonathan. 2015. 'Bitmessage'. Bitmessage.Org. https://bitmessage.org/bitmessage.pdf.

[22] Gwern.net,. 2015. 'Time-Lock Encryption - Gwern.Net'. http://www.gwern.net/Self-decrypting%20files.

[23] Nakamoto, Satoshi. 2008. https://bitcoin.org/bitcoin.pdf.

[24] Comments.gmane.org,. 2014. 'Bitcoin Developers' Mailing List ()'. http://comments.gmane.org/gmane.comp.bitcoin.devel/4116.

[25] Realitykeys.com,. 2015. 'Reality Keys - Facts About The Future, Cryptographic Proof When They Come True'. https://www.realitykeys.com/.

[26] News.ycombinator.com,. 2015. 'I Like The Parallelized Hash Chain Construction Idea; I've Never Seen That Befor... | Hacker News'. https://news.ycombinator.com/item?id=6509688.

[27] Todd, Peter. 2014. 'Petertodd/Timelock'. Github. https://github.com/petertodd/timelock.

[28] Garg, Sanjam, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. 2013. 'Candidate Indistinguishability Obfuscation And Functional Encryption For All Circuits'. In , 40--49.

[29] Goldfeder, Steven, Joseph Bonneau, Edward W Felten, Joshua A Kroll, and Arvind Narayanan. 'Securing Bitcoin Wallets Via Threshold Signatures'.

Version 2 published on 05/04/15.