Open-Transactions: Secure Contracts between Untrusted Parties

Chris Odom

chris@opentransactions.org

Abstract

A low-trust notary could replace conventional transaction servers and would allow users to gain access to safe, fast, inexpensive, off-chain transactions with increased functionality. We propose a solution that enables parties to contract with each other without being able to defraud each other, and to issue currencies and to prove all balances, as well as prove which instruments are valid and which transactions have closed, without storing any history except for the last signed receipt. Theft of reserves and counterfeiting are prevented by storing cryptocurrency reserves in multi-signature voting pools consisting of notaries who audit each other in real time and then vote multi-signature on the blockchain to release coins only when authorized by users' signed receipts.

1. Introduction

The Bitcoin blockchain is an ideal medium for issuing currencies and for censorship-resistant, peer-to-peer payments[6]. But the trade-off is that blockchain transactions are slow and expensive compared to using a server. Users also commonly employ server-based systems for added functionality such as market exchange and smart contracts. Unfortunately, servers in use today must be trusted to hold the funds, to accurately maintain their internal ledger, and to faithfully execute the transactions requested by their users. As a result, problems traditionally associated with third party intermediaries have crept into the Bitcoin ecosystem, including reversible transactions, verified accounts, frozen balances, and expropriated funds.

What is needed is a transaction server based on cryptographic proof instead of trust, allowing any willing parties who wish to contract with each other to enjoy the benefits of a server without needing to trust it. In this paper, we propose a solution that demotes transaction servers to mere notaries, only able to countersign contracts that have first been signed by their clients. Since only each client has access to its own private key, receipts are unforgeable. Theft of reserves and counterfeiting are prevented by storing bitcoins and colored coins in multi-signature voting pools consisting of notaries who audit each other in real time and then vote multi-signature on the blockchain to release coins only when authorized by users' signed receipts.

2. Transactions

I. Financial Instruments

We define a transaction as a group of operations on contracts capable of objectively proving balances (and changes of balance) between adversarial parties. Open-Transactions implements financial instruments as *Ricardian Contracts*, which are contracts that can be understood by humans as well as manipulated by software[3].

All transactions use the same basic structure: the parties involved sign agreements which are then countersigned by an independent notary server. Furthermore, transactions are irreversible since the receipts are always formed and signed on the client side first, before being notarized by any server. This prevents the notary from falsifying receipts, since it can't forge the client's signature.

This basic structure can be built upon to create many types of financial instruments. Those supported by Open-Transactions include:

- **Transfer.** An atomic movement of funds from one account to a different account, like a bank account-to-account transfer.

- **Cash.** Untraceable cryptographic tokens which can be securely redeemed by the recipient without revealing the sender.[1]
- **Cheque.** A payment which is not deducted from the sender's account until the recipient claims it.
- **Voucher.** A payment which is deducted from the sender's account at the time of creation.
- **Invoice.** A payment request which the recipient can opt to pay from any of his accounts.
- **Market Offer.** An open agreement to exchange a given quantity of one unit type for a given quantity of another unit type.
- **Smart Contract.** A customizable agreement between multiple parties, containing user-defined scripted clauses, hooks, and variables.

II. Destructible Receipts

In the solution we propose, each notary-signed receipt contains a copy of the original user-signed request.[2] Since that request also includes a statement of the balance, we can always prove the current balance using the most recent receipt.[3]

In addition, we can prove which instruments are still valid by including a list of open transaction numbers on each signed receipt[7].

Open-Transactions is thus able to prove the balance, as well as which instruments are valid, and which transactions have closed, without having to store any history except the last signed receipt.

Does this mean that parties *should* destroy their historical receipts? Not necessarily. But it should be noted that parties are not *forced* to keep their receipt history in order to prove the current state. Proofs which require a full history are $O(n)$, whereas Open-Transactions proofs are $O(1)$. $O(1)$ balance proofs are preferable to $O(n)$ proofs, because even though most users would choose to save their transaction history, the risk of a balance proof failing due to data loss does not grow without bound over time.

A few communication tables will be illustrative. First let's see how Alice takes ownership of some transaction numbers:

Alice	- I'm all out of transaction numbers. Please give me some more. *Signed: Alice*
Notary	- Okay, I have placed a receipt containing numbers #700 through #799 in your nymbox. *Signed: Notary*
Alice	- I hereby accept transaction numbers #700 through #799. You may remove the receipt from my nymbox. *Signed: Alice*
Notary	- Done. *Signed: Notary*

Bob needs to do the same thing:

Bob	- I'm all out of transaction numbers. Please give me some more. *Signed: Bob*
Notary	- Okay, I have placed a receipt containing numbers #800 through #899 in your nymbox. *Signed: Notary*
Bob	- I hereby accept transaction numbers #800 through #899. You may remove the receipt from my nymbox. *Signed: Bob*
Notary	- Done. *Signed: Notary*

At this point, Alice is now responsible for transaction numbers #700 through #799, and Bob is responsible for transaction numbers #800 through #899.[4]

[1] Based on work by David Chaum[2]. Open-Transactions currently includes Lucre[5] by Ben Laurie.

[2] Based on work by Ian Grigg[4].

[3] Based on work by Bill St. Clair[13].

[4] Client software should always remember the highest transaction number it has ever accepted, and should never in the future accept any number lower than that. Otherwise the notary could trick the client into accepting responsibility for a previously-used transaction number, and thereby defraud the client by processing the same instrument twice.

Alice is now able to create a cheque payable to Bob:

Cheque	#700
Amount:	65
To:	Bob
Signed:	*Alice*

So far no balances have changed yet. Alice gives the cheque to Bob who deposits it into his account:

Bob	- I'm using Txn #800 to deposit this cheque in the amount of 65 units. - My last balance was 350, so my new signed balance will be 415 units. - Transaction numbers #801 through #899 will still be signed-out to me. *Signed: Bob*
Notary	- Success. *Signed: Notary* [Note: at this point, the notary places a cheque receipt into Alice's inbox, and moves 65 units from Alice's account to Bob's.]

What does the notary verify when Bob performs his cheque deposit? Among other things:

- Did Bob sign this cheque deposit?
  (He did. Good.)
- This cheque deposit is #800. Is that transaction number currently signed out to Bob? (It is. Good.)
- Bob claims that after this deposit is complete, transaction numbers #801 through #899 will be the numbers still signed out to him. Is that true? (It is. Good.)
- Does Bob actually own the account he is depositing into? (He does. Good.)
- After the deposit is complete, will Bob's new balance actually be 415 units?
  (It will. Good.)
- Is there actually a cheque attached to this deposit? (There is. Good.)
- Is the cheque drawn on an account of the same asset type as that of the account that Bob is depositing into? (It is. Good.)
- Is the cheque signed by Alice?
  (It is, good.)
- Does Alice own the account the cheque is drawn on? (She does. Good.)
- The cheque is #700. Is that transaction number currently signed out to Alice?
  (It is. Good.)
- Does Alice already have a cheque receipt for #700 in her inbox?
  (She does not. Good.)
- Are there sufficient funds in Alice's account to cover this cheque?
  (There are. Good.)

Satisfied with these things, the notary countersigns Bob's deposit. He now has access to spend those funds.

Later, Alice checks her account's status:

Alice	- What is my current account status? *Signed: Alice*
Notary	- There is a cheque receipt #700 in the amount of 65 units in your inbox. - Therefore, though your last signed balance was 1000, your current balance is 935 units. *Signed: Notary*

Of course Alice's client software doesn't just accept this information at face value, but verifies it against her last signed receipt in a process similar to the notary's verification of Bob's deposit. After that is satisfied, she then processes her inbox:

Alice	- I'm using Txn #701 to process my inbox. - You may remove cheque receipt #700 (in the amount of 65 units) from my inbox. - My last signed balance was 1000, so my new signed balance will be 935 units. - Transaction numbers #702 through #799 will still be signed-out to me. *Signed: Alice*
Notary	- Success. *Signed: Notary*

It should be noted that the notary would not have been safe to remove the cheque receipt from Alice's inbox until this point. Only now that Alice has signed a new balance agreement can the cheque receipt be removed, since before that, the notary needed the cheque receipt in order to prove the current balance.

Now that Alice has processed her inbox, her new signed balance is 935 units and she is still responsible for transaction numbers #702 through #799. (#700 and #701 are now closed.) She can discard any previous receipts.

Meanwhile Bob's new signed balance is 415 units and he is still responsible for transaction numbers #801 through #899. (#800 is now closed.) He can also discard any previous receipts.

If Bob were to now attempt to deposit cheque #700 a second time, it would fail to verify, since Alice is no longer responsible for Txn #700. But what if he tried *before* she processed her inbox? His attempt would still fail, since there would already be a cheque receipt #700 in her inbox. In Open-Transactions you simply cannot deposit the same cheque twice.

Thus at all times throughout the process, all parties are able to prove their current balance, as well as which instruments are valid and which transaction numbers are closed, without storing any history except their last signed receipt. Alice's own signature proves these things to Alice, and Bob's own signature proves these things to Bob. The notary is unable to defraud them.

III. Recurring Transactions

While simple transactions occur as described above, recurring transactions are slightly more involved, requiring an opening transaction number for each user, and a closing transaction number for each asset account.

Consider an example where Alice places a market offer. Three transaction numbers in total must be used by Alice:

Market Offer	#702
Type:	Bid
Purchasing:	Gold
Amount:	200 grams
Into:	Alice's Gold Acct
Using Closing Txn:	#703
Maximum Price:	$40 per gram
From:	Alice's Dollar Acct
Using Closing Txn:	#704
Signed:	*Alice*

In the above market offer, #702 is the opening number for Alice (the user), #703 is the closing number for Alice's Gold Account, and #704 is the closing number for Alice's Dollar Account. Each time a trade occurs against that market offer, two *market receipts* will be created, with one placed into Alice's Gold Account's inbox, and the other placed into Alice's Dollar Account's inbox. Both market receipts will be in reference to the opening #702. Any changes in balance for those two accounts must be justified by market receipts in the same amount, and each market receipt contains a copy of Alice's original offer.

Once the market offer expires or is canceled, a *final receipt* in reference to #702 will be placed into each inbox. Final receipt #703 for Alice's Gold Account, and final receipt #704 for Alice's Dollar Account, both in reference to Alice's original market offer #702. On Alice's next transaction, whatever that may be, she must remove #702 from her statement of open transaction numbers, closing it permanently.

When Alice processes her Gold Account's inbox, she can close final receipt #703 as long as she also closes all related market receipts in her Gold Account's inbox, and signs the new gold balance. Similarly, when she processes her Dollar Account's inbox, she can close final receipt #704 as long as she also closes all related market receipts in her Dollar Account's inbox, and signs the new dollar balance.

All types of recurring transactions, including smart contracts, follow a similar process. All of this is managed "behind the scenes" by the client software and hidden from the user, with the end result that recurring transactions fit into the same security model as simple transactions.

3. Counterfeiting

Since a notary is unable to falsify Alice's receipts against her will, the only crime left is counterfeiting funds with a dummy account. That is, even without falsifying any of Alice's receipts, a notary can still create a dummy account and then sign a false receipt for that dummy account, and thus create counterfeited funds which can then be spent to Alice or Bob. Thus, counterfeiting is the only crime still possible by an Open-Transactions notary.

Fortunately, counterfeiting can be easily prevented by auditing the receipts. But while it is technically easy to construct an audit server, we still need a party who is incentivized to operate that audit server. And the solution to incentive depends on the various methods by which currencies are issued. They are:

1. The issuer directly issues his currency onto a specific notary.
2. The issuer issues his currency as a colored coin onto the blockchain.
3. The currency is a cryptocurrency (which has no issuer.)

Let's examine these in turn:

The issuer directly issues his currency onto a specific notary. If, for example, there is an issuer of a gold-based currency who is actually storing gold bars in a vault somewhere, he can act as a normal Open-Transactions user and just directly issue units of his currency onto a notary. In this case there is no blockchain involved whatsoever.

The drawbacks are:

1. The issuer can be coerced into terminating his relationship with a specific notary.
2. The issuer must operate an audit server himself in order to prevent that notary from counterfeiting.
3. His currency will only be available on that specific notary.

The issuer issues his currency as a colored coin onto the blockchain. In this case, the currency can circulate independently on the blockchain itself, outside the confines of any notary. Thus the issuer has no connection to any specific notary. Whether or not a user chooses to upload some of his coins to a notary is entirely outside of the issuer's knowledge or control. In this case the issuer does not need to audit any notary, and his only concern is that the gold in his vault matches up with the colored coins he has issued onto the blockchain.

The question remains: if a user does choose to upload some of his colored coins to a notary, how will that notary be audited to prevent counterfeiting inside Open-Transactions?

Here are the basic options:

1. The user can just trust the notary to hold the coins, and trust the notary not to be audited. (This is by far the worst option but it is the standard practice today in the Bitcoin world.)

2. Users can choose an external auditor to receive a percentage of their transaction fees. This is the solution favored by the Voucher-Safe[1] project.

3. Users can upload their colored coins into a multi-signature voting pool composed of several notaries who all audit each other in real time. In this case the auditing is effectively the same as with normal bitcoins. (Below.)

The currency is a cryptocurrency (which has no issuer.) Bitcoin, like all cryptocurrencies, does not have a real-world "issuer", but instead coins are created by the peer-to-peer network itself.

If a user chooses to transact his bitcoins off-chain using an Open-Transactions notary, he can do so by uploading those coins into a multi-signature voting pool composed of several notaries who all audit each other in real time.

This solves two problems at once:

1. Counterfeiting. Multiple parties are auditing each notary, which prevents it from counterfeiting on its internal ledger.
2. Theft. An individual notary is also incapable of stealing coins out of the pool, since a multi-signature vote is necessary to retrieve the funds. (And those votes are controlled by the auditors.)

4. Voting Pools

We propose *voting pools* as an arrangement of notaries to securely store and account for customer cryptocurrency deposits, and to redeem valid withdrawal requests even in the event the customer's notary of choice has completely disappeared. Voting pools are designed to ensure that no single person or organization can ever perform unilateral actions on deposited funds, in order to reduce the risk of loss or theft, and custodial liability[10]. Each notary in the voting pool operates its own audit server, and each auditor has a corresponding blockchain wallet. The blockchain wallet manages the multi-signature transaction generation, as well as a hierarchical and deterministic list of addresses for bitcoins[12] and colored coins[11].

When a customer deposits cryptocurrency into a voting pool, he receives corresponding units in his account on his notary of choice[8]. How? Each audit server watches the receipt stream for requests to deposit or withdraw bitcoins or colored coins from the voting pool, and then communicates with its bitcoin wallet as appropriate. The auditor independently verifies the Open-Transactions operations of all notaries in the voting pool, as well as the bitcoins held by the pool on the blockchain itself. The auditor uses this audit data to know when it should direct the wallet to create a withdrawal transaction[9], and it is also the component responsible for information sharing and achieving consensus between all members of the pool. Effectively each auditor conducts a permanent, real time proof-of-reserves audit against all of the notaries in the pool, and simultaneously enforces it. It is the auditors and the wallets who hold the keys to creating blockchain transactions at the request of the user, and the auditors must all act by consensus and with the cooperation of the wallet to create multi-signature blockchain transactions.

5. Smart Wallets

While a voting pool approaches the highest level of security that is theoretically possible on the server side, it is still not perfect: *m-of-n* notaries may still collude. In fact no server-centric solution achieves perfection. Even Bitcoin itself does not entirely eliminate risk, but rather, distributes risk effectively across a peer-to-peer network.

Ultimately, in order to achieve a practical distribution of risk, the client software must take responsibility for its own funds, keeping some safe on the blockchain, and distributing some across multiple voting pools for easy access to fast, inexpensive, high-functional transactions.

Open-Transactions is client-centric. The smart wallet of the future will be responsible for key management, interfacing with hardware keys, backups of wallet data, and most importantly, distribution of funds across the blockchain and across multiple voting pools.

6. Conclusion

We have proposed a notary server based on cryptographic proof as a replacement for transaction servers based on trust. Transaction servers traditionally have authority over user balances. To solve this, we proposed the use of triple-signed receipts in order to prevent the notary from changing a user's balance without his signature, or falsifying any of his transactions. Furthermore, we proposed the use of inboxes and transaction numbers in order to make it possible for all parties to prove which instruments are valid, and which transactions have closed, without storing any history except the last signed receipt. Counterfeiting by the notary is still a problem. To solve this, we proposed the use of auditing, and we must provide an incentive for parties to operate audit servers. In the case where an issuer has issued his currency directly onto a notary, we proposed that the issuer must directly audit that notary. Alternately, client software may pay a percentage of its transaction fees to an independent auditor. In the case of bitcoins or colored coins, we proposed the use of multisignature voting pools. These voting pools prevent the theft of funds by any single notary, and they also provide multiple notaries with incentive to audit each other, which prevents counterfeiting. While a voting pool approaches the highest level of security that is theoretically possible on the server side, *m-of-n* pool members may still collude, and so we have proposed that the smart wallet of the future must achieve a practical level of security by automating the distribution of client funds across the blockchain and multiple voting pools.

References

[1] *Voucher Safe*. `http://www.voucher-safe.org/tiki-index.php?page=System+Utility`.

[2] David Chaum. *Blind Signatures for Untraceable Payments*. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1983.

[3] Ian Grigg. *The Ricardian Contract*. In *Proceedings of IEEE Workshop on Electronic Contracting July 6*, pages 25–31, 2004.

[4] Ian Grigg. *Triple Entry Accounting*, 2005. `http://iang.org/papers/triple_entry.html`.

[5] Ben Laurie. *Lucre: Anonymous Electronic Tokens v1.8*. Technical report, 1999, 2008.

[6] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*, 2008. `http://bitcoin.org/bitcoin.pdf`.

[7] Chris Odom. *Triple Signed Receipts*, 2010. `http://opentransactions.org/wiki/index.php?title=Triple-Signed_Receipts`.

[8] Justus Ranvier. *Voting Pool Deposit Process*. `http://opentransactions.org/wiki/index.php/Voting_Pool_Deposit_Process`.

[9] Justus Ranvier. *Voting Pool Withdrawal Process*. `http://opentransactions.org/wiki/index.php/Voting_Pool_Withdrawal_Process`.

[10] Justus Ranvier. *Voting Pools*. `http://opentransactions.org/wiki/index.php/Voting_Pools`.

[11] Justus Ranvier and Jimmy Song. *Hierarchy for Colored Voting Pool Deterministic Multisig Wallets*. `https://github.com/Open-Transactions/rfc/blob/master/bips/bip-vpc01.mediawiki`.

[12] Justus Ranvier and Jimmy Song. *Hierarchy for Non-Colored Voting Pool Deterministic Multisig Wallets*. `https://github.com/Open-Transactions/rfc/blob/master/bips/bip-vpb01.mediawiki`.

[13] Bill St. Clair. *Truledger in Plain English*, 2008. `http://truledger.com/doc/plain-english.html`.