Rune: A Comprehensive Solution for Transactions of Blockchain-based Funds

Chris Odom and Justus Ranvier

with Michal Pospieszalski

and Todd Douglas

chris@runewallet.com

November 22, 2019

Abstract

A comprehensive system, based on open protocols, providing:

- User-controlled identity and payments across all blockchains, based on BIP-47 payment codes.
- A solution for the problems of receiving addresses in order to have a single, re-usable address across all blockchains, enabling return addresses, privacy for on-blockchain transactions, and proof that any given deposit, payment or withdrawal really was sent to the identified person.
- User credentials for handling KYC-AML regulatory requirements for regulated parties such as banks, while maintaining users' control over their own identity.
- User-controlled issuance of tokens on-blockchain, for whatever purpose, based on SLP, ERO-20, and other popular token protocols.
- Server deposits for off-blockchain transactions of coins and tokens, based on Open-Transactions. [6]
- Currency agnostic integration of multiple blockchains and their derivatives.
- On-blockchain, off-blockchain, and cross-blockchain interactions.
- A solution for theft and fraud wherever funds are deposited with 3rd-party custodians, based on pools of servers that employ multisig storage of funds.

1. Introduction

Rune's Open Transactions and BIP-47 Identifier technologies provide a cryptographic solution for the issuance, storage, and transaction processing of blockchain-based assets. The technology solves the exchange theft and fraud problem by replacing transactions based on trust with cryptographic proof, and it allows issuers, users, and exchanges to meet KYC/AML requirements while instantaneously issuing, storing, and trading blockchain agnostic assets.

The Bitcoin blockchain and its derivatives implemented a much needed 'foundational layer' of hard money, where transactions are irreversible, censorship-resistant, and consist purely of peer-to-peer payments [6] secured by proof-of-work. This ecosystem is also an ideal medium for the issuance and circulation of third-party coins and tokens. Today, several issuance protocols exist, including Simple Ledger Protocol (SLP) [2] for Bitcoin-based blockchains and ERC-20 for Ethereum. However, significant problems of user identity, exchange theft and fraud, and transaction latency persist.

Blockchain users and custodians need solutions to a variety of identity problems, such as a reusable, return address that can be used for public identification and return receipts. Solving the problem of receiving addresses as we know them today is critical for users' experience and blockchain utility. Rune proposes to solve this problem using the BIP-47 address format in conjunction with its identity credential support built into the Open-Transactions (OT) financial library.

Blockchain users, seeking functionality and liquidity, often deposit funds with third-party custodians such as exchanges. These custodians are historically where most of the theft and fraud have occurs, and the losses are in the billions. Under the current system, 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. Rune solves the problem of trust with its OT server, which uses cryptographic proof and allows willing parties to contract with each other and enjoy the benefits of a server without needing to trust it.

In this paradigm, transaction servers are demoted to mere notaries, only able to countersign contracts that have already been signed by their clients. The client is the only one with access to his/her private key, making receipts unforgeable by the notaries, who only "timestamp" transactions as they come through.

Exchange theft and fraud can be eliminated by storing blockchain coins and tokens in multi-signature (multi-sig) voting pools (X-of-Y) that consist of notaries who audit each other. Whenever a withdrawal needs approval, the auditors must perform a multi-sig vote on-blockchain to release only the funds authorized by a user's signed receipt. In this scenario trading latency will disappear as transactions occur instantaneously.

2. Identity

Ascertaining and using *identity* is critical for many blockchain-based businesses. What is *"identity,"* from a software perspective? Your social security card or driver's license is not the sum total of your identity, not even in court. Like a marriage license, or a dog license, or even a witness statement in an affidavit; each is just one piece of evidence about you. Or more accurately, each is a claim (or set of claims) about you that has been certified by someone you know, or by some official authority that others trust. These pieces of evidence are like a cloud formed around you. But even the cloud is not your identity itself. Identity is an ethereal thing. The cloud merely consists of various pieces of *evidence* about your identity, which is the closest we can ever achieve in the real world to tracking a person's identity. This will always be true. Put another way, having the name of "Bill Gates" is not the same thing as being *the* Bill Gates, the founder of Microsoft. It's not the name alone, nor some singular taxpayer ID number that imbues identity, but rather, those names and numbers are just examples of verified (or verifiable) facts about that person. *A person's true identity can never be tracked as anything more than a set of claims – and claims about those claims – regarding specific facts, relationships and accomplishments in that person's life, that distinguish him from anyone else subjectively, in the eyes of those he is interacting with.* The only question that remains is whether the user will create and control his/her credentials, rather than an authority.[1] However, even given a system where users have full control over theirf cryptographic credentials, we can still *have* authorities, and authorities can set permissions or revoke access in their own systems. This decentralized model of identity provides users with self-determination, while ensuring that various platforms can still regulate their internal systems.

[1] As seen with X.509 certificate authorities.

3. Identifying users for AML-KYC or other purposes

One requirement for many blockchain-based businesses is to be able to identify a user, and from there to be able to link that user to the signatures they sign, the payments they make, the messages they send, the claims that distinguish their identity, and authentications about those claims made by various authorities. But how can we associate a person's identity to a single ID, given the ethereal nature of identity itself? How can we prove which metadata is *authoritative,* given one person's signature or another's?

To answer this question, we will examine what's possible using BIP-47 payment addresses, identity credentials, and blind claims.

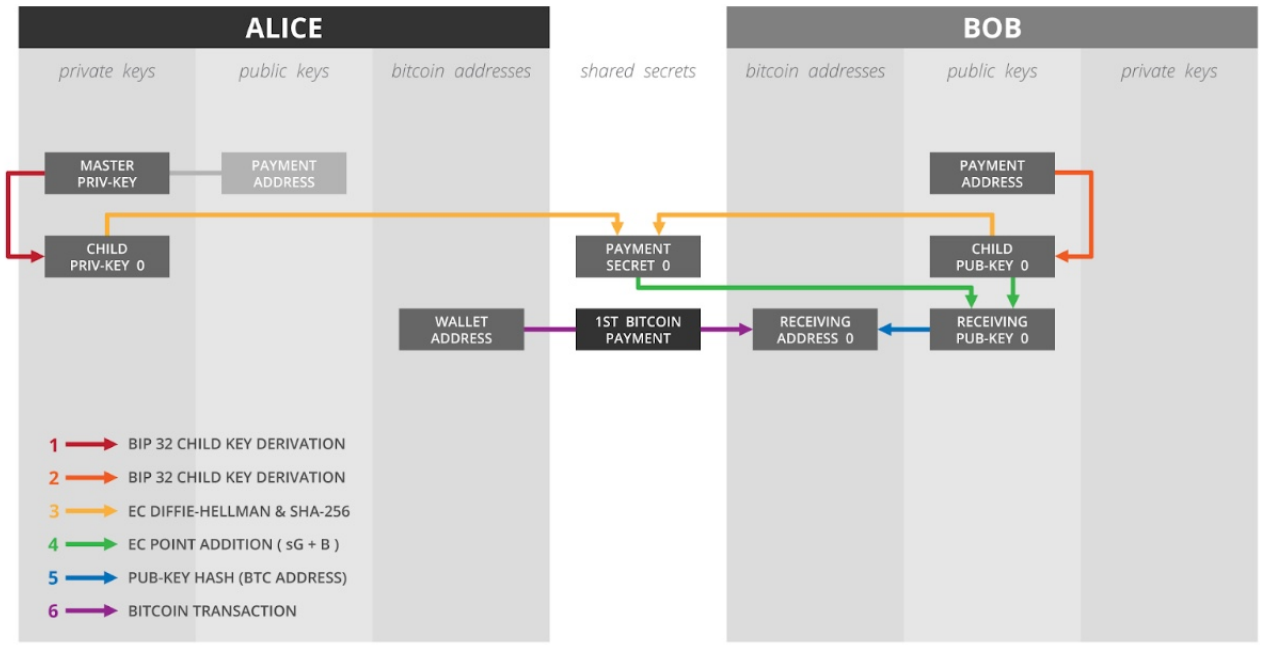
Rune employs Open-Transactions identity credentials, which allow users to make claims about themselves regarding their facts, relationships, and accomplishments. Moreover, anyone else, including trusted authorities, as well as colleagues, family members, etc. can sign verifications of those claims. And it is in those authenticated claims where we can find the proofs needed for identifying each person according to the needs of each scenario, such as when an exchange performs a withdrawal for one of its users.

Authenticating claims requires linking a set of credentials to a specific re-usable ID and its signatures, and that is the purpose of BIP-47.

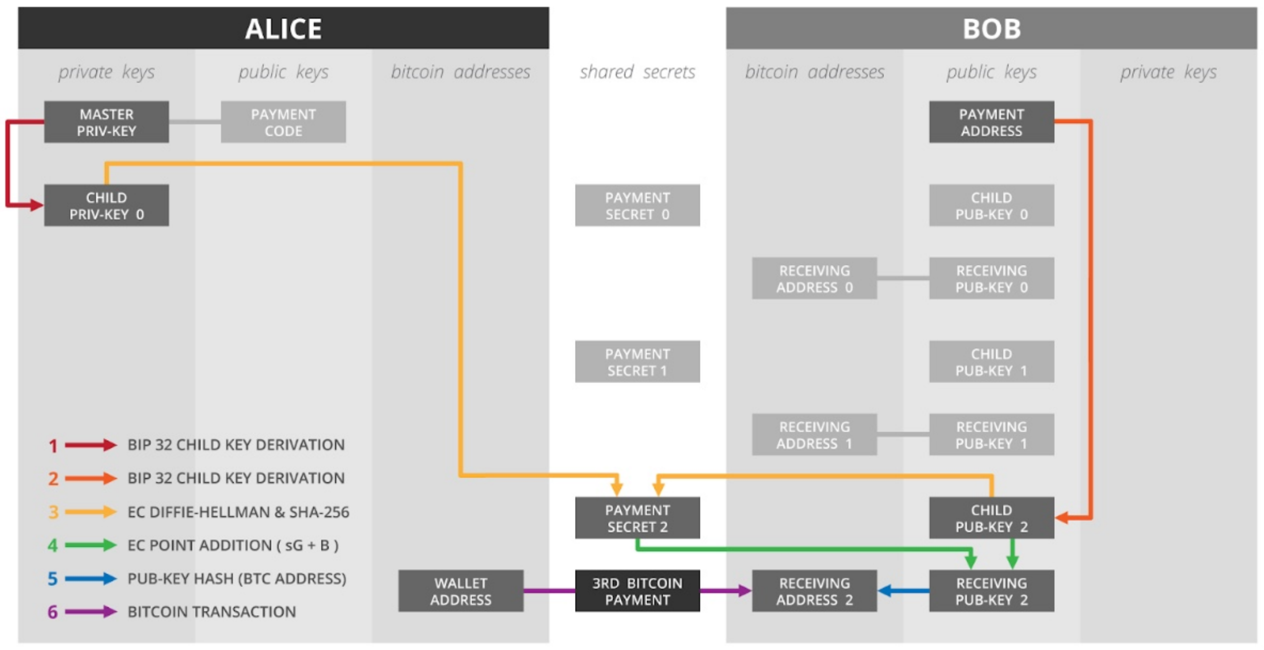
4. BIP-47

The *BIP-47 Payment Code* is a way to format blockchain addresses so that each user can have a single, re-usable address across all blockchains, without the address itself appearing in any on-blockchain transactions, and thus cannot be observed publicly on-blockchain by any third parties, unless they have the party's private key. Instead, the receiving addresses are calculated deterministically via *"spooky action at a distance"* using a Diffie-Hellman shared secret. [12] This is a process whereby each party, using his/her own private key and the other party's public key, is able to calculate a shared secret key, which no one else can calculate without one of the private keys. The parties also increment an index after each transfer, so there is a new blockchain receiving address calculated for each transaction between them *(see diagrams).*

Alice Pays Bob - 1st Time



Alice Pays Bob - 3rd Time



The calculation, transfer, storage, and maintenance of new Bitcoin receiving addresses has up until this point been an enormous headache for Bitcoin merchants, exchanges, and users, who usually need to generate and track a new receiving address for every transaction. Receiving addresses really are a low-level detail that should not require user involvement, and BIP-47 makes that experience a reality for the user, by restoring some privacy and fungibility to on-blockchain transactions, while simultaneously allowing users to know *for certain* who they have paid and who is paying them, without the need to constantly exchange new receiving addresses.

Justus Ranvier, Rune's Chief Architect and the inventor of BIP-47, said:

> *Payment codes are a technique for creating permanent Bitcoin addresses that can be reused and publicly associated with a real-life identity without creating a loss of financial privacy. They are similar to stealth addresses but involve a different set of trade-offs and features that may make them more practical. [8]*

It's important to note that payment codes do not require any changes in the Bitcoin protocol itself, and so adoption has already begun. Bitcoin wallets including Billion, Samourai and Rune Wallet have already adopted the technology.

The BIP-47 payment code provides users with a single ID that can be used across all blockchains, without having to generate a different ID for each blockchain. The same ID will work on Litecoin, Bitcoin, Bitcoin Cash, etc. — any blockchain where the user holds a private key.

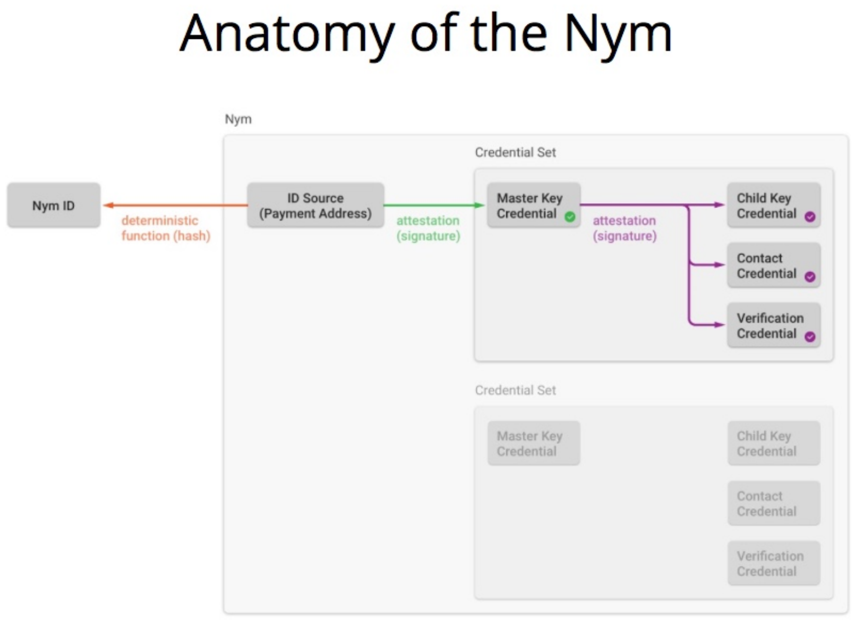
I. Useful Properties of BIP-47:

- A user's payment code can be freely published everywhere and serve as his 'public ID' without fear of losing any security or anonymity.
- Users only need to exchange payment codes *once.* After that, messages and payments, agreements, and smart contracts can safely flow between them, without having to exchange a new address each time.
- The same payment code works across *all Bitcoin-derived blockchains*, such as Litecoin, BSV, and Bitcoin Cash, as well as on-and-off blockchain. Rune's roadmap also includes support for other popular chains like Ethereum and EOS.
- The same BIP-47 code that is used for payments can also be used for messaging, and for signing messages that are sent, as well as for verifying signatures on messages received.
- A third party public observer will *not* see your payment code in any of your actual on-blockchain transactions. On-blockchain transactions will use receiving addresses that are generated deterministically by both parties without having to transmit anything, using a Diffie-Hellman shared secret *("spooky action at a distance.")* [12].
- The parties to a transaction can see their entire history between each other. This is possible because each party has one of the private keys needed to generate the deterministic receiving addresses used by the other party.
- When a message or payment is sent, the sender knows for certain who the recipient is, and there is no confusion that the recipient has provided some third party's receiving address instead of their own.
- When a user receives a payment, the recipient knows for certain who the sender is. The recipient will see the sender's *"display name"* on the incoming payment, and can hit 'reply' to send a payment or message back to the sender. Bitcoin ATM operators always ask about this functionality.

5. Identity Credentials

It's not enough to prove that a signature was signed by the expected BIP47 ID. We must also be able to attach **metadata** to that ID in the form of *credentials*, consisting of *claims* and *claim verifications* by various authorities.

In crypto software, a *pseudonym* (or simply *"Nym")* is a potential identity that has a public address (his BIP-47 ID in our case), a corresponding private key, and a set of credentials associated with them. However, that doesn't mean that any real-world information has been authenticated regarding the *actual person* behind that Nym.



$$PaymentAddress + Credentials = Nym(pseudonym) \tag{1}$$

Bitcoin wallets often disappointingly show *your own receiving address* (not the sender's address) when displaying a record of received funds. But with BIP-47, when you receive an on-blockchain payment, there *is* a display name for the sender. This occurs naturally because the Opentxs API has already downloaded the sender's *identity credentials*[2], which are cryptographically verified against the sender's BIP-47 address. The *display name* field is one good example of the sort of metadata that users can attach to their credentials via signed claims.

A user's credentials will contain other claims made about a person, regarding various facts, relationships, and accomplishments. For example, *"My SSN is XXX-XX-XXXX, signed Bob."* His claims may also be *verified* (signed) by other users, and those *claim verifications* may also appear on his credentials. [13] For example, a verified claim might look like this, if we imagined it in human-readable form: *"The hash fingerprint of an encrypted JPG of my scanned driver's license is 1Jasd876y2jhg34h998, signed Bob"* followed by: *"We have officially authenticated this information in person and can reproduce it upon demand,* ***signed,*** *Some Trusted Authority."* Note that digital signatures are unforgeable.

[2]See ContactManager::NewContact(), https://github.com/Open-Transactions/opentxs/blob/develop/include/opentxs/api/ContactManager.hpp

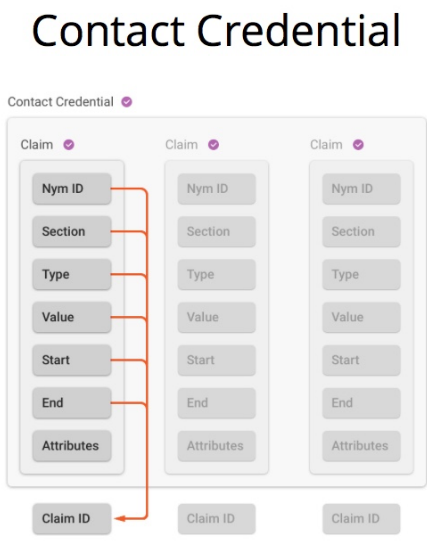


Figure 1: *Contact Credential*

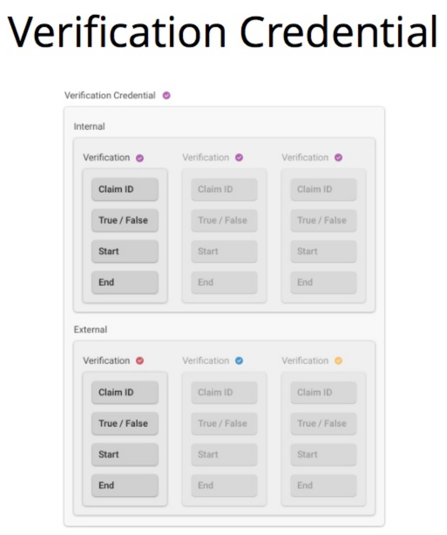


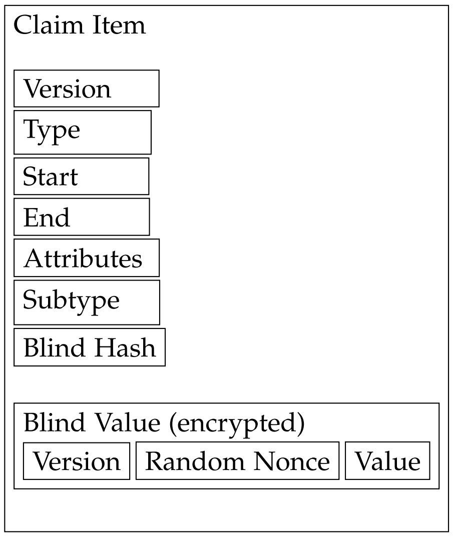
Figure 2: *Verification Credential*

For example, when you first meet someone and add them as a new contact by scanning their QR code, Rune Wallet creates *reciprocal verified claims* between both parties, proving to themselves later that they have definitely met each other in real life. This way, a *"web of trust"* can form organically, without any need for *"key signing parties"* as envisioned in the earlier days of crypto. Various trusted authorities will be responsible for identifying users and verifying claims, as we shall explore below. That way, users will be able to prove their identity to one another, without having to ask the authority, since his signature already appears on the claim verification.

6. Blind Claims

User credentials will also take advantage of *blind claims*, meaning those claims are proven in a 'blind' way. To do this, the *value* field of the claim is encrypted, and the Trusted Authority signs a hash of this *blind value* instead of signing the plaintext value as might normally otherwise occur. However, they have still seen and verified the plaintext value before signing the claim verification. This blinding is often desirable. For example, there may be legal requirements in certain situations to prove a person's age, and so it would be useful for users to be able to prove their age and photo, *without* having to post their birthdate or photo publicly for the whole world to see. Using a blind claim makes this possible.

In Open-Transactions the structure of a blind claim is the same as for a normal claim, except there is no *value* field. Instead, the claim includes a ***blind value*** field that's encrypted to a (deterministic) key; structured like this in Open-Transactions:



Note that the blind value itself is encrypted to a deterministic key, which is recoverable from the user's seed.[3] Publicly only the blind hash is visible on the user's credentials, as well as any signed verifications of that claim.

For example:

- The State of California could sign the hash of your encrypted birthdate allowing you to prove your age on demand, when required.
- Alternately, someday you may be able to properly identify yourself to a notary public, and then he'll take your picture on the spot and sign the hash of an encrypted JPG of your face. Now when you are signing documents, not only is your signature provable from your BIP-47 ID, but you can also reveal your photo to any challenger while simultaneously proving that an *authority* has certified that they created the photo, and authenticated it for *that same BIP-47 ID.*
- A notary public – or onboarding department – could additionally scan your driver's license and then sign the hash of an encrypted JPG of your driver's license.
- The possibilities are endless. *Any* data can be authenticated and thus made available for users to prove to others in the future, without having to reveal that data publicly, and without having to re-verify their identity over and over again to an on-boarding service over a webcam.
- We must consider that any valuable data, ***will* end up being authenticated** by someone or another. Organizations will spring up to authenticate the data and sign the appropriate credentials.
  - Sherpas will certify that claimants have actually climbed Mount Everest.
  - The Trusted Authority will certify claimants with *Approved* or *U.S. Person* status.
  - Colleagues will certify they have worked with each other in the past.

Other forms of proofs, including zero-knowledge proofs, will eventually be added to user credentials. For example, zk-SNARKS and zk-STARKS are promising innovations that will likely

[3]Otherwise we might be forced to use a *deterministic nonce* instead of a *random nonce*.

be used for various blind or transparent proofs.

7. Open-Transactions

The Open-Transactions (OT) project is a collaborative effort to develop a robust, commercial-grade, fully-featured, free-software toolkit that implements **off-blockchain** transactions purely as **cryptographic proofs.**

I. Financial Cryptography

Open-Transactions uses strong cryptographic techniques to create secure financial instruments, such as digital signing to create non-repudiable contracts, and homomorphic encryption to create untraceable digital cash. In OT, transactions are unforgeable, receipts are destructible, and balances cannot be falsified or changed without user consent. OT can prove all balances, and determine which instruments are valid and which transactions have closed, without storing a receipt history, as long as the user has the last signed receipt.

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 contracts in OT use the same basic structure: all parties involved sign an agreement that is notarized by an independent third party witness. This technique is known as Triple-Signed Receipts. Importantly, transactions are formed and signed on the client side first, before being notarized by any server. An OT client is thus ensured that an OT notary server cannot falsify his receipts against his will since the server can't forge the client's signature. This basic structure can be built upon to create many types of financial instruments.

II. Financial Instruments

We define an off-blockchain transaction as a group of operations on contracts capable of objectively proving balances (and changes of balance) between adversarial parties.

All transactions use the same basic structure: the parties involved sign agreements which are then countersigned by an independent notary server. 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:

- **Transfers.** An atomic movement of funds from one account to a different account, similar to a bank account-to-account transfer.
- **Cash.** Untraceable cryptographic tokens which can be securely redeemed by the recipient without revealing the sender.[4]
- **Cheque.** A payment that is not deducted from the sender's account until the recipient claims it.
- **Voucher.** A payment that 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.

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

- **Recurring Payments.** An agreement between two parties that includes an optional initial payment, followed by a set number of additional payments over a specified period of time.
- **Smart Contract.** A customizable agreement between multiple parties, containing user-defined scripted clauses, hooks, and variables.

III. Destructible Receipts

In the solution we propose, each notary-signed receipt contains a copy of the original user-signed request.[5] Since that request also includes a statement of the balance, we can always prove the current balance using the most recent receipt.[6]

We can also prove which instruments are still valid by including a list of open transaction numbers on each signed request [7]. The request also includes a list of any incoming transactions; each transaction must use an open transaction number.

At all times throughout the process, all parties using Open-Transactions can 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.

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 that require a full history are $O(n)$, whereas Open-Transactions proofs are $O(1)$. $O(1)$ balance proofs are preferable to $O(n)$ proof, 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.

IV. 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 be sent to Alice or Bob.

Fortunately, counterfeiting is easily prevented by auditing the receipts. But while it is technically easy to construct an audit server, we still need a party that is incentivized to operate that audit server.

V. Voting Pools

The most ideal solution is for users to deposit their on-blockchain funds (coins or tokens) into a multi-signature voting pool composed of several notaries who all audit each other, voting on-blockchain when necessary to approve withdrawals. This solution is only possible if the off-blockchain system is based on cryptographic proofs, as Open-Transactions is.

This Voting Pool multi-signature arrangement solves two problems at once:

1. Counterfeiting. Multiple parties are auditing each notary, which prevents it from counterfeiting on its internal ledger.

[5]Based on work by Ian Grigg [4].
[6]Based on work by Bill St. Clair [16].

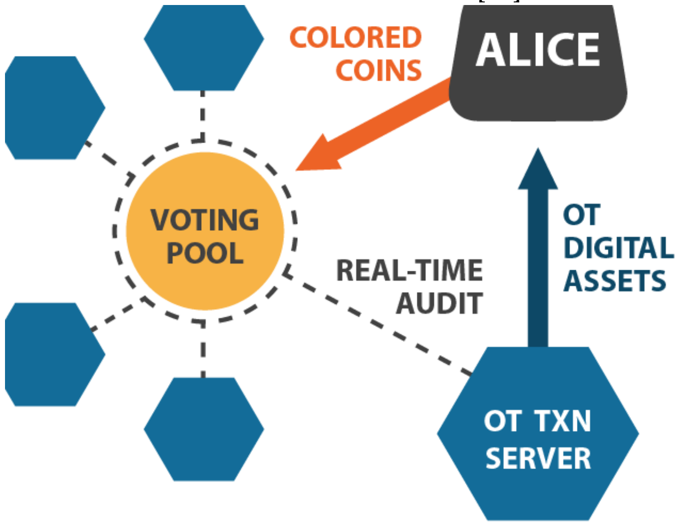
2. Theft. An individual notary is also incapable of stealing coins out of the pool because a multi-signature vote is necessary to retrieve the funds. (And those votes are controlled by the auditors.)

8. Voting Pools

Currency exchanges and other trading platforms prefer to perform order matching more rapidly than what is possible on the blockchain itself. These services currently accept custody of user funds, perform transactions in a separate off-chain system, and use a database to track customer balances. Typically these services are not cryptographically secured or independently auditable. Customers give full control of their deposited funds to the custodial service, which exposes them to the risk of theft or loss.

Unlike legacy currencies, cryptocurrencies can be irrevocably lost or stolen, and it's typically not possible to distinguish between insider or external theft. Historically, this ambiguity appears to have been routinely exploited.

We propose *voting pools* as an open standard intended to be a universal replacement for bespoke systems that handle customer cryptocurrency deposits. A voting pool is an arrangement between notaries to store and account for customer cryptocurrency deposits and to honor valid withdrawal requests even in the event the customer's notary of choice has disappeared. Voting pools are designed to ensure that no single person or organization can ever perform unilateral actions on deposited funds, and to reduce the risk of loss or theft, and custodial liability [11]. 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 [15] and colored coins [14].



When a customer deposits cryptocurrency into a voting pool, he receives corresponding units in his account on his notary of choice [9]. 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 [10], 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 the notaries in the pool, and simultaneously enforces it. The auditors and the wallets hold the keys to creating blockchain transactions at the request of the user, and the auditors must all act by consensus with the cooperation of the wallet to create multi-signature blockchain transactions.

I. Security Goals

To achieve the desired security and robustness goals for voting pools, the following criteria will be enforced:

- Customers should be strongly discouraged from reusing deposit addresses. The voting pool itself must never intentionally reuse a bitcoin address.
- All Bitcoin addresses used by the pool must be deterministic for auditing purposes. Each member of the pool should be able to calculate all members' series of deposit and change addresses.
- Withdrawal transaction input selection must be deterministic to minimize the cost of coordinating transaction signing.
- It must be possible to keep a majority of the private keys offline for security reasons and bring them online as needed to process withdrawals.
- It must be possible to alter the voting pool by adding, removing, or replacing members in a coordinated and secure fashion. It is necessary to use *Smart Properties* to accomplish this. These actions are described in the section below on colored coins.

II. Security Model

The goal of the voting pool security model is that users of deposit-accepting services should never experience a loss of deposited funds.

There are three broad reasons users lose deposited funds:

- Type 1 Event (Theft/Loss). A user permanently loses their funds because a third party has gained control of them without the user's consent, or because the private keys needed to spend them have been irrevocably lost.
- Type 2 Event (Denial of Service). A user temporarily loses some or all of their ability to use their funds, but no third party has gained control over them.
- Type 0 Event. Type 0 Events will be used to describe all other abnormal conditions from which the pool must recover, which does not directly involve a loss of customer deposits.

III. Security Theorem

If the probability of $m + 1$ (Type 1 Event) or $n - m + 1$ (Type 2 Event) services simultaneously and identically behaving in a malicious or incompetent manner is lower than the probability of any individual server acting in a malicious or incompetent manner, user deposits on that service are at less risk of loss if the service is a member of an $m - of - n$ voting pool than they would be at risk if the service is not a member of a voting pool.

Voting pools can guarantee the integrity of user deposits if, in any given situation, at least m pool members are well-behaving for Type 1 events and at least $n - m$ pool members are well-behaving for Type 2 events.

IV. Auditor

The Auditor listens to the Audit Streams broadcast by all the Notaries and independently verifies them for correctness. The same stream which carries regular OT transaction information also contains the OT receipts for Bitcoin withdrawal requests from the pool. The auditor initiates or authorizes a blockchain withdrawal transaction via the wallet if and only if the audit for that service is clean (verified correct).

The auditor is responsible for maintaining an independent copy of the same deposit database as the transaction server. It also tracks withdrawals from the time at which it receives an OT receipt, containing a withdrawal request, until the corresponding Bitcoin transaction is fully confirmed on the blockchain.

All messages which must travel between the transaction server and the blockchain wallet pass through the auditor.

In order to create withdrawal transactions, wallets must be able to select inputs and change outputs, and calculate the minimum required transaction fee deterministically. In order to achieve determinism, the sequence of withdrawals must be globally consistent. Before sending any withdrawal request to the wallet, the auditors are responsible for achieving consensus on a serialization order for withdrawals.

V. Notary

The Notary must keep track of all blockchain-denominated balances via OT receipts. In addition to the separate account(s) for each customer, the server must track a service account to hold the blockchain-denominated balance owned by the service itself.

If necessary, the server should also maintain an application account to hold the balance of any funds which are being manipulated by an external system.

For example, in the case of a high-frequency exchange, the application account would belong to the order matching engine. When a customer enters a trade, the exchange front-end will call the applicable OT functions to transfer the appropriate balance from the selling customer's OT account to the application OT account, and also from the application account back to the the appropriate purchasing customer's OT account. Any trade fees that the exchange earns would be sent to the service account as part of the transaction. This separation of application account, service account, and customer account, prevents the mingling of funds.

The Notary is also responsible for passing PaymentRequests from the Auditor to the customer, crediting customer balances after the successful receipt of a blockchain deposit, and passing withdrawal requests to the rest of the voting pool via the audit servers.

The Notary must maintain a permanent deposit database containing each PaymentRequest generated for that server and its associated status (number of blockchain confirmations and the OT receipt crediting the appropriate Nym with the deposit).

VI. Audit Stream

The Notary must broadcast five types of messages in an indexed and hash-chained sequence, which form an Audit Stream.

The five types of messages are:

- Update to an inbox
- Update to an outbox
- Update to an account balance file
- Update to a Nym box file

- All Notary replies to transaction requests.

VII. OT Client

For users to deposit cryptocurrency into the voting pool, the OT Client must be capable of parsing, verifying, and, if necessary, forwarding to another blockchain wallet client, the BIP70 payment requests. This step is required to ensure a malicious server can not send a fake deposit request that results in a customer sending funds to an address not controlled by the pool.

All users of services that are part of a voting pool must have an OT client running on their device to use the service. The service front end can communicate transparently with the OT Client via a local websocket interface.

The OT client will be capable of operating in the background with the bare minimum of necessary interaction in order to exert the lowest possible disturbance of the front end service's user experience (UX).

VIII. Voting Pool Key Management

Each server in the pool requires an offline, air-gapped machine for key generation called a key server. The server is equipped with either a dedicated, non-networked printer or a CDR drive. No media of any kind is ever allowed to cross the air gap in the online-to-offline direction.

The key server generates random BIP32 seeds in batches (default: 52, or enough for one year). When a batch is created, it prints the xpubs (extended public keys) for all 52 seeds on paper as QR codes (alternately on a virgin CDR, which is discarded after a single use). This paper is then manually walked to the auditing server and scanned. The auditing server adds each xpub to the keyset definition.

At the same time, the key server also prints two redundant copies of the QR codes containing the xprivs (extended private keys), one per page (one per CD), which the service should hold in some physically secure fashion and back up securely. It is not necessary for all individual services to take extraordinary measures to protect the private keys from physical destruction since the pool can tolerate a loss of keys that involves less than (n-m) members. One copy held in an offsite location with another copy held on-site is sufficient.

Xprivs are loaded into the auditing server in series number order to create the hot series. Each participant in the pool should have a method of being notified when the hot series is close to being exhausted so that an employee can be instructed to load the next xpriv into the auditing server.

New key batches should be generated early before the old batch is consumed (default: 75 percent used). If for any reason one of the participants is late and does not generate a new batch on time, the last defined series number is used for accepting deposits until the administrators of the other pool members can correct the situation.

The key server must also be equipped with a scanner. Before putting any keys into service, they must be validated.

Validation procedure: The key server will create the first one million public and private keypairs from each seed in the batch, sign a nonce with each private key and verify the signature with the corresponding public key.

Then the user will scan in the printed public and private keys, and the key server will verify the scanned versions match the original versions and repeat the million keypair test.

Both versions of the test must match identically before placing any of the keys into service.

IX. Standard Pool Sizes

Every pool represents a compromise between performance and cost. For security and reliability purposes, higher reliability levels are better, however they must be balanced against the cost factor. Standard pool sizes are the lowest cost pools that produce a given reliability level.

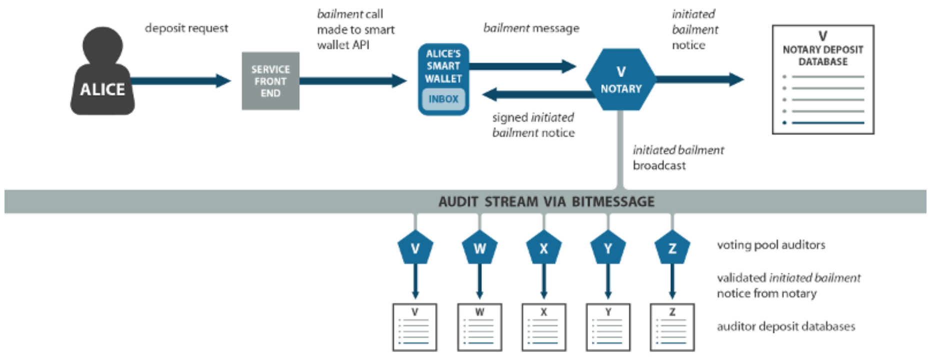
Reliability Level	Theft Resistance Factor	Redundancy Factor	Cost Factor	Pool Size
1	2	1	5	2-of-3
2	3	2	8	3-of-5
3	4	3	11	4-of-7
4	5	4	14	5-of-9
5	6	5	17	6-of-11
6	7	6	20	7-of-13
7	8	7	23	8-of-15

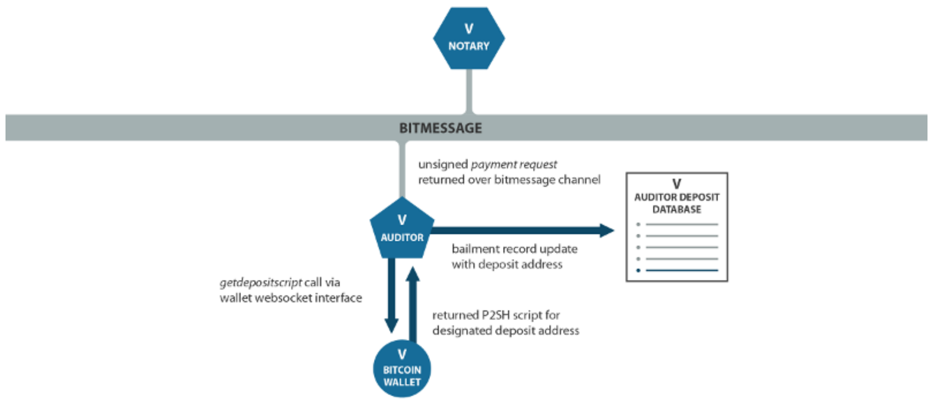
9. Voting Pool Deposit Process

I. Initiation

Customers will typically request a deposit address by interacting with the service front end web site or some other software application. When the service receives such a request, it notifies the OT client via the OT client API function: *requestBailment*.

When the OT client receives notice of a user desire to deposit funds to a voting pool, via any method, it sends a *bailment* transaction request to the notary to initiate the deposit process. The notary validates this request and replies with a signed receipt. A copy of this receipt is broadcast to the *audit stream*, and another copy is stored inside an *initiatedBailment* notice that's placed in the user's inbox. The notary adds this association to a *bailment database* for future reference.


ALICE
deposit request
SERVICE FRONT END
bailment call made to smart wallet API
ALICE'S SMART WALLET
INBOX
bailment message
signed initiated bailment notice
V NOTARY
initiated bailment notice
V NOTARY DEPOSIT DATABASE
initiated bailment broadcast
AUDIT STREAM VIA BITMESSAGE
V
W
X
Y
Z
voting pool auditors
validated initiated bailment notice from notary
auditor deposit databases


V
NOTARY
BITMESSAGE
unsigned *payment request* returned over bitmessage channel
V
AUDITOR DEPOSIT DATABASE
V
AUDITOR
bailment record update with deposit address
getdepositscript call via wallet websocket interface
returned P2SH script for designated deposit address
V
BITCOIN WALLET

II. Payment Script Generation

When an audit server validates the notary's reply to the bailment message from the notary to which it is assigned, it adds the receipt identifier to its *bailment database* and calls *getDepositScript* via the websocket interface to the blockchain wallet, using the address identifier for the next unused deposit address.

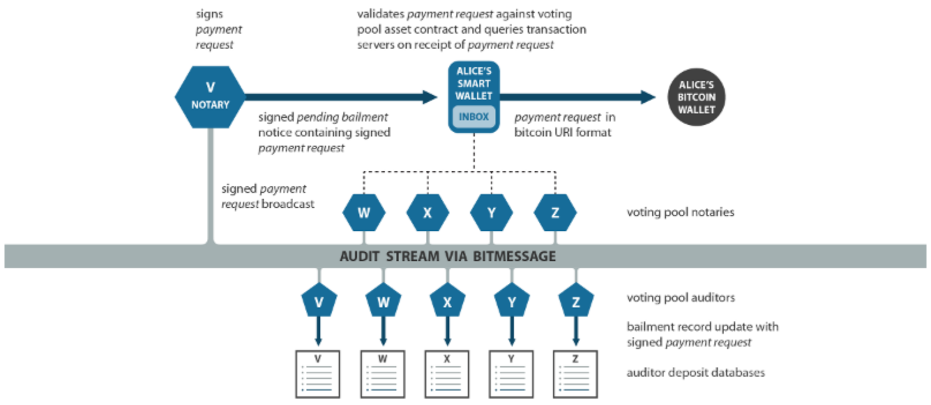
The wallet calculates and returns the designated deposit address as P2SH output script. The audit server uses this information to update the bailment database, and to assemble and sign a *PaymentRequest*.

III. PaymentRequest delivery

The audit server broadcasts the *PaymentRequest* to all notaries and auditors in the pool. The notary replaces the user's *initiatedBailment* notice in the inbox with a *pendingBailment* notice containing a copy of the *PaymentRequest*.

When the other audit servers in the pool receive the *PaymentRequest* broadcast, they add the deposit to their respective bailment databases. The other notaries in the pool cache the *PaymentRequest* to answer verification requests from the OT client.

The OT client should validate the PaymentRequest against the voting pool asset contract. If it is valid it should query a random selection of other notaries, at least $m - 1$, using the *PaymentRequest identifier* to verify whether they have seen it. If this check is successful, it then initiates the blockchain transaction by passing the *PaymentRequest* to the user's local wallet application, which is configured to handle [[*bitcoin* :]] *URIs*.

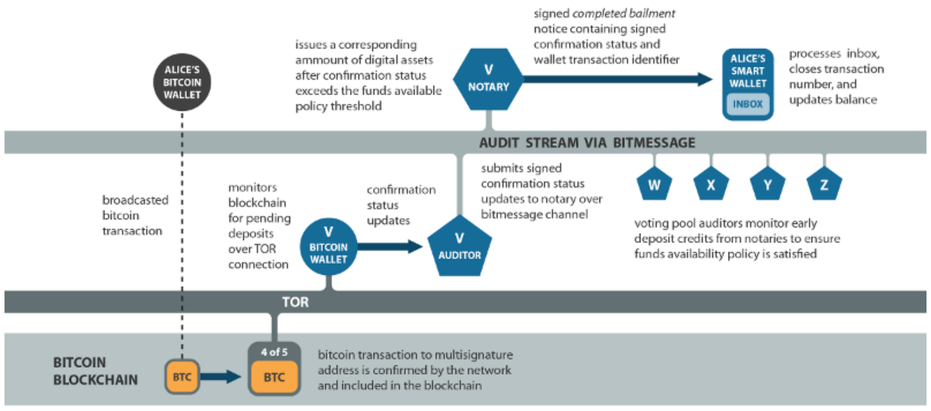

signs *payment request*
validates *payment request* against voting pool asset contract and queries transaction servers on receipt of *payment request*
V NOTARY
ALICE'S SMART WALLET
INBOX
ALICE'S BITCOIN WALLET
signed *pending bailment* notice containing signed *payment request*
payment request in bitcoin URI format
signed *payment request* broadcast
W
X
Y
Z
voting pool notaries
AUDIT STREAM VIA BITMESSAGE
V
W
X
Y
Z
voting pool auditors
bailment record update with signed *payment request*
auditor deposit databases

IV. Deposit and Crediting

The audit server broadcasts the *PaymentRequest* to all notaries and auditors in the pool. The notary replaces the user's *initiatedBailment* notice in the inbox with a *pendingBailment* notice containing a copy of the *PaymentRequest*.

When the other audit servers in the pool receive the *PaymentRequest* broadcast they add the deposit to their respective bailment databases. The other notaries in the pool cache the *PaymentRequest* to answer verification requests from the OT client.

The OT client should validate the *PaymentRequest* against the voting pool asset contract. If it is valid then it should query a random selection of other notaries, at least $m - 1$, using the *PaymentRequest identifier* to verify whether they have seen it. If this check is successful, it then initiates the blockchain transaction by passing the *PaymentRequest* to the user's local wallet application, which is configured to handle $[[bitcoin:]]$ $URIs$.


ALICE'S BITCOIN WALLET
issues a corresponding ammount of digital assets after confirmation status exceeds the funds available policy threshold
V NOTARY
signed *completed bailment* notice containing signed confirmation status and wallet transaction identifier
ALICE'S SMART WALLET
INBOX
processes inbox, closes transaction number, and updates balance
AUDIT STREAM VIA BITMESSAGE
broadcasted bitcoin transaction
monitors blockchain for pending deposits over TOR connection
V BITCOIN WALLET
confirmation status updates
V AUDITOR
submits signed confirmation status updates to notary over bitmessage channel
W
X
Y
Z
voting pool auditors monitor early deposit credits from notaries to ensure funds availability policy is satisfied
TOR
BITCOIN BLOCKCHAIN
BTC
4 of 5
BTC
bitcoin transaction to multisignature address is confirmed by the network and included in the blockchain

Type 1 Event: Fraudulent Deposit Address. A malicious or hacked operator may give the customer an invalid PaymentRequest in an attempt to steal deposits.

Each pool member's Bitcoin wallet must notify its audit server of any deposits received to an address the pool controls. When an incoming transaction is received to an address the audit servers are expecting due to previously broadcast *PaymentRequest*, the servers will use the *getinfomultisigwalletaddress* calls to identify the (inbound) transaction, and *gettransactionstatus* to monitor its confirmation status.

Type 0 Event: Deposit Never Received. It's possible the customer may never actually transfer bitcoins after requesting a PaymentRequest.

Type 1 Event: Unknown Deposit. The deposit is received to an address that has never been used, and for which a PaymentRequest was never created, so no member of the pool knows to which nym it should be credited.

Type 1 Event: Duplicate Deposit. The deposit is received from an address that has been previously used, so the audit servers know to that Nym the address is assigned.

Type 0 Event: Dust Handling. The size of the deposit may be below the network dust threshold (small enough that it would require more in transaction fees to spend than it is worth).

The audit server relays the number of confirmations the incoming transaction has received to the notary. Once the number is sufficient according to the Funds Available Policy, the notary will issue an OT asset for the amount of the deposit to the nym of the user.

The notary will replace the *pendingBailment* notice (in the inbox) with a *completedBailment* notice, that includes a signed copy of the original bailment request, and a copy of the audit server's signed notice of confirmations, and transaction identifier provided by the blockchain wallet.

The OT client processes the user's OT account inbox, removing the *completedBailment* notice, which simultaneously closes the transaction number and updates his OT balance.

The audit servers in the pool must monitor all deposits to ensure the Funds Available Policy is satisfied to avoid the risk of a double-spend or chain fork causing insolvency. Any server which offers more early deposit credits than what it can cover with its service account must have its audit status set to degraded.

Type 2 Event: Non-credited Deposit. The notary fails to place a *completedBailment* notice in the user's inbox after a successful Bitcoin transfer.

If an initially-seen deposit fails due to a chain fork, and if the user has not yet been credited with an OT receipt for the deposit, the status of the deposit remains pending. The audit server should notify the notary by setting the number of confirmations back to zero. In the typical case of blockchain reorg event, the deposit transaction should re-enter the mempool automatically, and the wallet can assist with this by rebroadcasting it.

If an initially-seen deposit makes it onto the blockchain, and becomes invalid due to conflicting transactions, the audit server should mark the deposit as failed by setting the number of confirmations to -1. The audit server should notify the notary of the failure, who then replaces the user's *pendingBailment* notice with a failedBailment notice. The notary should update the status to failed in the bailment database and the address should not be intentionally reused. The OT client can then process the user's inbox, removing the *failedBailment* notice and closing the transaction number. In this case there is no change to the OT account balance, unlike with a *completeBailment* note.

Type 1 Event: Reversed Deposit. A deposit could disappear from the blockchain after the user has been issued an OT receipt

10. Voting Pool Withdrawal Process

I. Initiation

Customers will normally request a withdrawal of bitcoins from the pool by interacting with the service front end web site or some other software application. When the service receives such a request, it notifies the OT client via the OT client API function: outBailment.

When the OT client receives notice of a user desire to withdraw funds from a voting pool, via any method, it sends an outBailment transaction request to the notary containing the destination Bitcoin address where the withdrawal should be sent, the amount of the withdrawal, and an extraFee value. The extraFee is added to the transaction fee required by the Bitcoin network and is paid directly from the user's balance and can be zero.

Note: Some customers may wish to have additional restrictions placed on withdrawals. These restrictions can include the prevention of withdrawals to arbitrary Bitcoin addresses, or two-factor confirmation of withdrawals, and time delays to allow for notification and manual review of withdrawal requests. All this functionality and more can be provided by users electing to store their deposits in an OT smart contract instead of a standard receipt. Discussion of the capabilities of OT smart contracts is outside the scope of this document.

After the notary receives the outBailment transaction request, it removes the total amount from the user's balance and places it in the outbox as a pendingBailout receipt.

Type 2 Event: Withdrawal blocking. The notary handling a customer account may fail to respond to a valid withdrawal request.

When the auditors see the pendingBailout receipt, they create an entry in their withdrawal database and add the pendingBailout to their queue for the next consensus.

II. Consensus Round Transaction Formation

Each time a new consensus decision is finalized, the auditors begin processing the specified pendingBailouts (if any) by passing the address identifier of the first input to be used, the address identifier of the first change address to be used, and the withdrawal identifier of each output to the Bitcoin wallet via the *startwithdrawal* API call.

The *startwithdrawal* API call accepts a list of outBailments to process, and the set of parameters which are needed to ensure the transaction process is deterministic.

When the wallet receives this call, it processes the list and parameters per the transaction construction algorithm and returns a list of signatures and status information for each *outBailment*.

If the wallet requires the private keys from additional series to fulfill the outBailments, that information will be returned with the status information.

III. Consensus Round Signature Exchange

The auditor takes the signature list from the wallet and broadcasts it to the rest of the auditors. It also collects signature lists from the other auditors and queues them for delivery to the wallet.

IV. Consensus Round Transaction Fee Accounting

Before the auditor can provide the missing signatures to the wallet, it must ensure the transaction fee has been properly accounted for. While the auditor is broadcasting and gathering signatures, it also sends a *txFeeNoticemessage* to each notary from which a withdrawal has been processed, indicating their share of the blockchain transaction fees included in this consensus round.

Blockchain transaction fees are allocated to a specific server by the fraction of total withdrawals in the round originating from that server. The method of calculating the individual shares should ensure the individual totals add up exactly to the blockchain transaction fees consumed by the transaction.

When the notary receives at least m identical txFeeNotice messages, it then performs a balance adjustment by subtracting the amount from its service account for the pool, and adding that amount to the issuer account for the pool. The auditor cannot release the rest of the signatures to the blockchain wallet until it validates the appropriate balanceAdjustment notice in the audit stream.

Type 0 Event: Transaction Fees Accounting. The originating notary may fail to deduct blockchain transaction fees from its service account and broadcast this receipt in the audit stream.

When the auditor validates all needed balanceAdjustment notices, it delivers the signature lists to the wallet via the updatewithdrawal API call. The blockchain wallet then adds signatures to the transaction(s) until it has m, then it broadcasts the transaction(s) to the network. It is not necessary for each wallet in the pool to include the same list of signatures in the transactions they broadcast. As long as all the transactions are valid, the network should accept one version and include it into a block.

Type 2 Event: Transaction Signature Error. All or a portion of the signatures a wallet receives from the other pool members may be invalid for the given transaction.

The startwithdrawal API call returns a list of one or more normalized transaction identifiers (ntxid), where each ntxid is linked to a list of the withdrawal identifiers corresponding to the outputs in the transaction. The auditor updates its withdrawal database with the ntxids and begins tracking the confirmation status of the withdrawal. The auditor reports all this data and each new transaction confirmation to the notary.

V. Completion

Like deposits, withdrawal transactions will not be final until *maturationTime* confirmations have occurred. Once the auditor reports a number of confirmations greater than or equal to *maturationTime* for the blockchain transaction associated with a pendingBailout receipt, the notary replaces that receipt with a *completedBailout* receipt. When the auditors see the *completedBailout* receipt in the audit stream they can remove the associated entry from their withdrawal database and stop sending confirmation updates.

Type 2 Event: Failed Withdrawal Transaction. The Bitcoin network may fail to confirm any version of the withdrawal transaction.

11. Token Issuance

Various techniques make it possible to issue new units onto an existing blockchain, e.g., ERC-20 on the Ethereum blockchain, and Simple Ledger Protocol on the Bitcoin Cash blockchain. These tokens can be stored and transacted using wallet software, as well as deposited onto servers for off-blockchain transactions such as exchange. The following is a brief introduction to the concepts and techniques involved.

I. Colored Coins

The term ”colored coins” can mean two different things:

- A technique for carefully constructing blockchain transactions in a way that preserves information apart from the base monetary value of the underlying units.
- The extra information preserved in the blockchain by employing the colored coin technique.

For the sake of clarity we will differentiate the technique from the information by using the term *virtual tokens* to refer to the extra information that is preserved using the colored coins technique.

II. Virtual Tokens

Virtual tokens possess all the capabilities of currency, plus one additional capability, smart property, which is helpful for non-currency usage.

Users of virtual tokens can:

- Transfer them between individuals
- Combine multiple tokens into a single token of greater value
- Divide the value of a single token into multiple tokens
- Use them in blockchain-scripted contracts
- Store them on the blockchain with multisig scripts
- Unambiguously prove that any particular virtual token is a valid member of a set created by the issuer, without requiring the issuer to create and manage a token registry.

III. Enforcement

Virtual tokens represent ownership information, but they can't enforce real-world obligations. For example, a particular issuance of virtual tokens might represent tickets to a concert. The virtual token can prove the bearer should be allowed to enter the concert, but it can't force the bouncer to step aside and let him pass. Colored coin techniques can't prevent the user from manipulating the underlying bitcoins in a way that destroys the extra information, because operations of virtual tokens are governed by Bitcoin transaction rules, and colored coin requirements are more strict but not enforced by the network. Using virtual tokens in a transaction that does not obey the colored coin rules destroys their extra meaning, leaving behind only their base monetary value. This is equivalent to taking paper concert ticket and setting it on fire.

IV. Metadata

The quantity and ownership of virtual tokens can be stored in the blockchain, but the semantic information that indicates what a token means is not (and cannot be) similarly stored. For example, the blockchain will track how many concert tickets have been issued and which address owns them, but not the fact that they represent authorized entry into a particular concert at a specific time and place. The storage of and operations on metadata require a specific kind of external system, such as Open-Transactions.

V. Blockchain Limitations

The speed of colored coin transactions and the capabilities of scripted contracts that use virtual tokens are the same as those of the underlying blockchain.

12. Coloring Techniques

Two techniques may be used to create virtual tokens: transaction-based coloring and address-based coloring.

I. Transaction-based coloring

Transaction-based coloring was pioneered by ChromaWallet and works by identifying a specific Bitcoin transaction at a particular time as the ”genesis transaction” and tracking all units which descend from the genesis transaction. Transaction-based coloring can produce the full range of virtual token types and has the security property that even a loss of the original private keys to the genesis address cannot result in the issuing of counterfeit virtual tokens. This security property means the number of virtual tokens matching a color definition is fixed at the time of creation and cannot be altered in the future—which can be an advantage or a disadvantage, depending on the application.

II. Address-based coloring

Address-based coloring was created by Coinprism and tracks bitcoins, which are descended from any transaction that passes through a defined address. This means the issuer can easily create new units in the future, but so could a thief who manages to steal the private key for that address.

13. Virtual Tokens

The different use cases for virtual tokens can be divided into three general categories, which form the different types of virtual tokens.

I. Tickets

Tickets are transferable bearer tokens designed to be redeemed for some real world value.

Examples of tickets include:

- Event entry passes
- Store coupons and special offers
- Frequent flyer miles and other redeemable rewards

Both address-based and transaction-based coloring can be used to create tickets.

II. Certificates

Certificates are transferable and redeemable in the same manner as tickets, and they additionally entitle the bearer to revenue paid through the blockchain. Certificates can be used for bearer securities, such as securitized loans, mortgages, bonds, and dividend-paying stocks. Both address-based and transaction-based coloring can be used to create certificates.

III. Smart Property

Smart properties are transferable like tickets and certificates, and, every particular smart property is both unique and atomic. Only one smart property of a given identifier can be created, it may not be subdivided.

Smart properties can be used to indicate ownership of a unique real-world asset and also for objective naming of content-addressed mutable data. This naming function is related to, and an extension of, hash-based naming.

A common operation in software engineering is to use cryptographic hash functions to create short identifiers for large pieces of data. These functions are useful because hash values are easy to communicate since they are short, and also are easy to check since they are deterministic. As a result, if you know the name of some piece of data, you can independently verify that you have the correct copy of it. But the limitation of hash-based naming is that the named data can never change.

Smart property overcomes this limitation. A Bitcoin feature, called OP RETURN, allows arbitrary data to be attached to transactions. Every time smart property is moved it can be associated with a new hash. If data is identified by a smart property identifier instead of using the hash, the identifier of the smart property can objectively and unambiguously identify the most current version of the data.

14. Conclusion

Rune's technology leverages the Open Transactions financial library and Bip-47 identifier, to simplify and cryptographically secure token issuance, storage, and transaction processing of blockchain-based assets. The implementation of this technology allows for the creation of new business use cases across a myriad of global industries.

Use case examples:

- Retail Banking - Users can deposit cryptocurrencies in the operating pools of financial institutions, or federated collectives of financial institutions. These "off-chain" deposits are protected by the multi-sig pools, but don't require mining and will process with speeds comparable to regular financial transactions. The technology will reduce transaction costs, and simplify a user's ability to quickly pay for items with cryptocurrencies or a bank-issued stable coin fiat equivalent.
- Exchanges, Investment, and Trading. Voting pools are the currently the most secure model for operating an exchanges that can trade any asset. Furthermore, the identity credentials afforded by BIP-47 allow money transmitters to KYC/AML their users and provide cryptographic proof that they are sending and receiving funds from verified identities. Identity credentialing solves a significant compliance problem for centrally regulated businesses in a decentralized global ecosystem. Rune Wallet's technology meets the 2019 FINCEN guidelines for both hosted and un-hosted wallets.
- Exchanges - Rune Wallet's technology meets the 2019 FINCEN guidelines for both hosted and un-hosted wallets. Voting pools are the most secure model for operating exchanges. The identity credentials afforded by the BIP-47 protocol allows money transmitters to KYC/AML their users and provide cryptographic proof that they are sending and receiving funds from verified identities. Identity credentialing solves a significant compliance problem for centrally regulated businesses in a decentralized global ecosystem.
- Token Offerings - Rune's Colored Coin implementation simplifies the issuance and liquidity of any token, whether it's a security, asset, or "coin." Token issuers with the Rune system can make the cross-chain movement of tokens possible, e.g., issuers can start with Ethereum ERC-20 tokens and move them to the Bitcoin Cash blockchain at a user's request. Tokens on disparate blockchains can also be deposited into a single OT notary and traded against each other.

- Gaming/Social Media - Presently, these ecosystems usually employ their own in-house financial transaction systems that aren't globally liquid. If they use cryptocurrencies for payment, which is commonplace, the present solutions pose money laundering challenges as game operators find themselves unwilling middlemen for illicit transactions. Rune's identity protocol makes it easy for game companies to digitally meet the same financial requirements as traditional Fintech companies while maintaining their business model.

Rune solves the fundamental problem of bridging the gap between the modern value creation systems and its consumers. Global economies will continue to rely on centralized ”value creators” offering services consumers need and demand. Rune’s goal is to create a worldwide standard for off-and-on-blockchain financial functionality that allows centralized value creators to do business in a globally decentralized, but cryptographically secure system.

References

[1] 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.

[2] Jonald Fyookball, James Cramer, Unwriter, Mark B. Lundeberg, Calin Culianu, and Ryan X. Charles. *SLP Token Type 1 Protocol Specification*, 2018. `https://github.com/simpleledger/slp-specifications/blob/master/slp-token-type-1.md`.

[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] Chris Odom. *Open-Transactions: Secure Contracts between Untrusted Parties*, 2010. `http://www.opentransactions.org/open-transactions.pdf`.

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

[8] Justus Ranvier. *Payment Codes*. `https://www.reddit.com/r/Bitcoin/comments/3alzga/bip47_reusable_payment_codes/`.

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

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

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

[12] Justus Ranvier. *BIP-47 Specification*, 2015. `https://github.com/bitcoin/bips/pull/159`.

[13] Justus Ranvier. *Universal Identity*, 2015.

[14] 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`.

[15] 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`.

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