BitShares 2.0: Identity Management

Fabian Schuh, Daniel Larimer
Cryptonomex, Cryptonomex.com*
Blacksburg (VA), USA
{fabian, dan}@cryptonomex.com

***Abstract*—**

1 Introduction

2 Named Account

BitShares makes use of read-able account names that have to be registered together with a public key in the blockchain prior to its usage. On registration, a new account is associated with an individual and unique identifier that will be used internally to identify an account. In this case, the blockchain acts as a name–to–public-key resolver similar to the traditional domain name service (DNS). These named accounts enable users to easily remember and communicate their account information.

In BitShares, an account name can be transfered by updating the public key that is associated with it.

Furthermore, BitShares is the first smart contract platform with built-in support for *recurring payments* and subscription payments. This feature allows users to authorize third parties to make withdrawals from their accounts within certain limits. This is a convenient way to *set it and forget it* for monthly bills and subscriptions.

BitShares also designs permissions around people, rather than around cryptography, making it easy to use. Every account can be controlled by any weighted combination of other accounts and private keys. This creates a hierarchical structure that reflects how permissions are organized in real life, and makes multi-user control over funds easier than ever. Multi-user control is the single biggest contributor to security, and, when used properly, it can virtually eliminate the risk of theft due to hacking. Hence, BitShares does technically not have multi-*signature* accounts, but has multi-*account* permissions.

In the following we introduce the above mentioned features that come with named account in the BitShares network in more detail.

2.1 Transferability

Every BitShares account is assigned a globally unique name that can be selected by its creator. There are many potential uses for account names beyond simply being an alias to a set of *dynamic account permissions*. They can be used as user logins or mapped to domain names. These names are transferable, which means that they are valuable in their own right.

The BitShares blockchain defines a simple algorithm to determine the fee it charges to reserve a new account name. Names that contain a number, are longer than 8 characters, or contain no vowels are essentially free. Otherwise the name is priced according to its length. Committee members can propose a different fee for each length which gives BitShares the power to adapt to market demands.

As mentioned already, accounts can be transferred by updating the permissions used to control the account. However, the semantics of transferring an account are slightly different in the context of a web-of-trust. Users need a means to update their keys for security purposes while maintaining their standing in the web-of-trust. In other words, users must be given a way to explicitly transfer an account name to a new user while breaking any liability for how the account is used in the future.

When users transfer an account name to another user, they use a special transaction that clears all of the links in the web-of-trust. Both the buyer and the seller are protected by this fact, because simply updating the key that controls a named account does not signify a legal change in ownership.

2.2 Dynamic Permissions

The ability to require multiple digital signatures for sensitive operations on the blockchain is integral to the security of the platform. While a single secret key may be compromised, multiple keys distributed over multiple locations add redundant protections, which result in a far more secure experience.

Competing blockchain systems suffer from the following shortcomings:

- The M-of-N model cannot sufficiently reflect the management hierarchies of many real-life organizations.
- Equal weighting of M keys is not sufficient to express asymmetric ownership over an account.
- Coordination and signing must be done completely out-of-band.
- Keys cannot be changed without coordinating with all other parties.
- Signatures cannot be retracted while waiting on other parties.

*This work was supported by Cryptonomex and honorable members of the bitsharestalk.org community.

Multi-signature technology is all about permission management, and permissions should be defined in terms of people or organizations rather than keys. Consider an example company that is run by 3 individuals: Alice, Bob, and Carol. Alice and Bob each own 40% of the company and Carol owns 20%. This company requires 2 of the 3 principles to approve all expenses. You could define the company in terms of keys assigned to Alice, Bob, and Carol, but what if Alice wants to protect her own identity with a multi-signature check? Alice opts to use a service provider that performs 2-factor authentication on every action Alice makes. This protects both Alice and the company and the company does not need to change its permission structure to accommodate Alice's choice of 2-factor authentication provider.

In BitShares, we introduce a new approach to permissions based upon accounts which are assigned globally unique IDs.

Under this system, it is possible to define an account that has no keys itself, but instead depends solely upon the approval of other accounts. Those other accounts can, in turn, depend upon the approval of other accounts. This process forms a *hierarchy* of accounts that must grant permission. Each account can change its own permissions independently of any accounts above it in the hierarchy, which is what makes the permissions *dynamic*.

Each account defines its permissions as a set of keys and/or other account IDs that are each assigned weights by the account holder. If the combined weight of keys and/or accounts exceeds a threshold defined by the account, then permission is granted.

The second solution is to include the partially signed transaction in the consensus state and allow accounts to publish transactions that add or remove their approval of the transaction. This simplifies the signing coordination problem, enables people to change their mind before the threshold is reached, and applies the transaction immediately upon receipt of the final approval.

The process for executing a transaction that requires multi-signature authority is as follows:

1. Someone proposes a transaction and approves it with their account.
2. Other account holders broadcast transactions, adding their "Yes" or "No" to the proposal.
3. When the proposed transaction has the approval of all accounts, it is confirmed.

Every account is assigned *two* authorities: *owner* and *active*.

- An authority is a set of keys and/or accounts, each of which is assigned a weight.
- Each authority has a weight threshold that must be crossed before an action requiring that authority may be performed.
- The owner authority is designed for cold-storage, and its primary role is to update the active authority or to change the owner authority.
- The active authority is meant to be a hot key and can perform any action except changing the owner authority.
- The motivating use case is a 2-factor authentication provider as a co-signer on the active authority, but not on the owner authority.

With this approach, a user can remain confident that their account will always be in their control, and yet that control can be kept in cold storage where no one can hack it. This means that a company account can require the approval of its board of directors and each board member may in turn require 2 factor authentication.

Anyone can rotate keys frequently without having to disturb the permissions on the accounts of its users.

One of the challenges that has made multi-signature approaches difficult to use in the past is that the act of gathering the required signatures was entirely manual, or required specialized infrastructure. Once a transaction is signed, there is no ability to retract your signature, so the last party to sign gains a slight advantage over the other parties. With deeper hierarchies, gathering signatures becomes even more complex.

To simplify this process, a blockchain should manage the signature gathering process by tracking the state of partially approved proposed transactions. Under this process, each account can add (or remove) their permission to a transaction atomically, without having to rely upon an outside system to circulate the transaction. This becomes especially critical for hierarchies that are arbitrarily deep.

In order to keep things computationally bounded, an individual transaction will only traverse down two layers in a hierarchy. If more than two layers of hierarchy are present, then an account will have to propose (create one transaction) to approve a proposal (the other transaction). When the first proposal (transaction) is approved, permission is then added to the second proposal (transaction).

Under this approach, each individual pays a single transaction fee each time they approve an action, and every action involves at most 1 signature verification by the network. This process allows arbitrarily deep hierarchies to be formed without exposing the permission system to vulnerability of unbounded computation.

In theory, accounts can form a hierarchy that is arbitrarily deep, and evaluating that hierarchy can take an arbitrary amount of time. In practice, it is unlikely that a single transaction will have signatures more than 2 levels deep, which keeps them computation bounded. Anything that requires more than 2 levels is likely to involve many people, and would not be signed all at once. Instead, it would use the built-in proposed transaction infrastructure, which tracks partially approved transactions.

- With this approach, a board member can propose that his company approve a transaction.
- This can be extended logically to propose, and account propose, to approve a transaction.
- This process would collect transaction fees as all of the layers in the hierarchy gradually add their permissions, and at no time requires an unbounded calculation.

It is possible for two accounts to require each other to approve a transaction.

Imagine account X is created that requires A and Y to approve. Imagine account Y is created that requires B and X to approve. The graph looks like this:

$$A \rightarrow X \leftrightarrow Y \tag{1}$$

$$B \rightarrow Y \leftrightarrow X \tag{2}$$

A proposes that X spend 1 BTS and waits for approval from Y. B proposes that Y approve the proposal from A and waits for approval from X.

There is no way to resolve this problem with a single approval from any party due to the following reasons:

1. Neither account can act without the other and thus nothing can be accomplished.
2. Cycles don't have to be direct as in this case, they can involve arbitrarily long sequences and thus be non-obvious.
3. If users create an approval cycle in the active authority then the owner authority can be used to break the cycle; however, if they construct a cycle in the owner authority and the active authority then the accounts involved in the cycle would be locked out.
4. In practice client software can detect cycles and prevent them from being formed.

Dynamic hierarchical threshold multi-signature permissions provides people and organizations with a more natural way to express ownership and control policies. This approach makes the system easier to use, and ultimately more secure, than existing solutions.

The Ripple wiki has a documented, but unimplemented, proposal for a similar Multisign feature [1] that was discovered independently.

2.3 Recurring & Scheduled Payments

Recurring Payments are implemented as a set of withdrawal permissions. Each account can grant any number of withdrawal permissions to other accounts. A withdrawal permission includes following properties:

1. Start Date
2. End Date
3. Withdrawal Limit per Period
4. Period Length (i.e. 1 month)

Any asset type can be used in the withdrawal limit.

After a user grants the withdrawal permissions, the authorized account is allowed to make one transfer per period of an amount up to the limit. If there is insufficient funds then the withdrawal will fail. Withdrawal permissions are designed to be a convenience for merchants and users, as they do not represent a commitment to pay.

It is up to each merchant to initiate each withdrawal. The BitShares platform does not automatically authorize the transfer of funds unless sufficient signing authority has been reached.

For security purposes, many banks place daily withdrawal limits on user accounts. In the event that an account is compromised, a thief is limited in the amount of damage that they can do. Withdrawal permissions enable users to protect their BitShares funds in the same manner. To do so, a user creates two accounts: savings and checking.

The savings account has keys kept offline where they are unlikely to be compromised. Before placing the keys in cold storage, the savings account authorizes the checking account to make a daily withdrawal of up to $1000, for example.

The checking account can then pull money out of savings at up to this limit, per day, and then use those funds as needed. This gives the user confidence that their losses would be limited if their account is compromised.

As stated above, the withdrawal permission system does not automatically make payments. However, BitShares has another feature which enables scheduled payments: proposed transactions. At any time, a user can propose a transaction to execute at a specific date and time in the future. If the transaction has sufficient authorization (i.e. is properly signed by authorities) at the specified time, then it will automatically be executed.

A merchant can use this feature, combined with withdrawal permissions, to implement automatic payments after a one-time setup fee. In practice, it may be cheaper for merchants to maintain their own scheduler to automate billing, since the blockchain charges a fee to propose a transaction separately from the transaction's own fees.

2.4 Costumer Privacy

BitShares 2.0 makes use of confidential transactions (by means of blind signatures [2]) in combination with stealth addresses to protect customers privacy while keeping scalability and performance.

In [2], a scheme was proposed that allows generating a blind signature compatible with the existing Bitcoin protocol. There, the client requests a set of parameters from a 3rd partiy (e.g. a *signing server*) and synthesizes a public key to use in a transaction. To redeem the funds, the client transforms the hash of the transaction (*blinds*), sends to the 3rd party to sign and then transforms the signature (*unblinds*) to arrive at a valid ECDSA signature. The signed transaction is published revealing the synthetic public key and the unblinded signature. Hence, the 3rd party cannot learn about its participation neither from the public key nor from the signature.

The novelty of the scheme is that unlike the original Chaum blind signature scheme, this approach does not allow anyone to prove that the signing party signed a particular message, but instead provides a much stronger privacy: the resulting signature, public key and the message are all completely unlinkable to the signing party.

For BitShares, it allows to have *someone* else sign something without them being able to prove they were the one who signed it which is useful for multi-signature/two-factor on confidential transactions.

In combination with stealth addresses, we can now derive a *one-time public key* that can be signed by *someone* knowing that this *someone* cannot use any knowledge of the one-time public key to link back to us.

3 Conclusion

References

[1] Ripple Labs, “Multisig / Transaction Proposal,” https://wiki.ripple.com/Multisign#Transaction_Proposal.

[2] Oleg Andreev, “Blind signatures for Bitcoin transactions (second draft),” http://oleganza.com/blind-ecdsa-draft-v2.pdf.