Draft: A System and Process for Secure Account and Identity Management using a Blockchain Based Device Network

Yash Patil

October 2019

1 Executive Summary

This invention implements a threshold user authentication protocol that hides the password from the parties involved in authentication. When a user attempts to authenticate oneself, they send servers a blinded version of their password. A threshold number (as specified by the network) of servers must respond with a share of the user's encrypted envelope, which contains the user's decryption keys. These servers also respond with the blinded password raised to an exponent. The user aggregates these shares of the envelope using polynomial interpolation to create the full envelope. The user also aggregates their blinded password raised to an exponent from all the servers. The user decrypts the envelope using the aggregated blinded password (the user is able to do so because the aggregated blinded password is used to encrypt the envelope when the user registers). To establish consensus among the servers regarding user state, an underlying blockchain is used. This blockchain helps the network to agree on the state, the parameters of the cryptographic primitives, and the devices used in authentication. We maintain that this invention is not two separate protocols, but is an intertwined system.

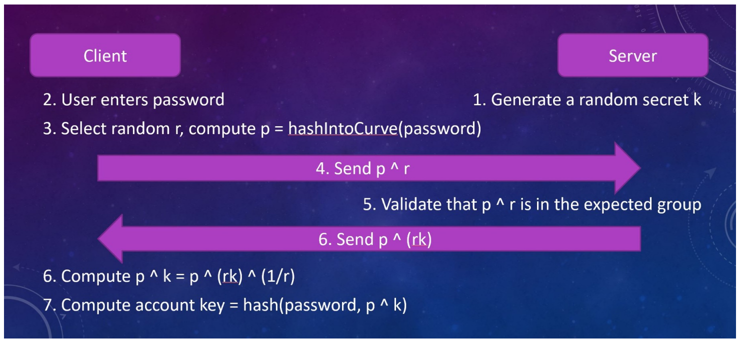

Client
Server
2. User enters password
1. Generate a random secret k
3. Select random r, compute p = hashIntoCurve(password)
4. Send p ^ r
5. Validate that p ^ r is in the expected group
6. Send p ^ (rk)
6. Compute p ^ k = p ^ (rk) ^ (1/r)
7. Compute account key = hash(password, p ^ k)

Figure 1: Device Communication with Network

Figure 1 above describes our baseline protocol in a single server setting without the use of an envelope (not in a distributed setting as in our paper). The client enters their password, hashes it into a curve and blinds it by raising it to some random coefficient r. The client the sends this value, p^r, to the server. The server validates that the value exists in an expected group and then raises it to a random secret k. The client then de-blinds this value (p^{rk}) by raising it to the power of $\frac{1}{r}$. The user can now use this value to create their decryption key for the envelope. In a distributed setting, multiple servers will each be sending a $p^(rk)$, of which the aggregates and then de-blinds to produce a single value (step 6 in above image). From this value, the client can produce the decryption key. Multiple servers will also be sending a share of the envelope, from which the client uses polynomial interpolation to retrieve the full envelope. The envelope contains the client's encryption and decryption keys (there three keys: the key to decrypt the envelope and the two keys that exist in the envelope itself). In this invention, the value p is replaced with α and the value the server sends is replaced with B_i.

2 High-Level Summary

2.a User Authentication Protocol

This protocol uses the cryptographic primitives of bilinear mappings, threshold cryptography, and threshold partially oblivious pseudo random functions (tpOPRF) to hide the password from the server and have multiple devices involved the protocol to obviate a single point of failure and provide enhanced security. A central part in this entire scheme is the envelope. Parts of this envelope are sent to devices on the network and contain the private and public key of the user. When logging in with a correct password, the user will regenerate the envelope, decrypt it, and obtain its private key.

The user first sends $\alpha = H_2(password)^r$, where H_2 is a hash function that maps onto group G_2 and r is a random prime number that blinds the hash (A hash function is a deterministic one way function that maps a value of arbitrary size onto a value of fixed size. This function is impossible to reverse). The hash function combined with the exponent r effectively blinds the password. The user then sends this value, α to a threshold number of devices in the network. The devices respond with

1. $B_i = e(a, H_1(username))^{k_i}$, where k_i is the secret key for the device, e is an admissible bilinear pairing, and H_1 is a hash function that maps onto group G_1. The device is not storing anything related to the password. Rather, the device is computing an output for the password that user will later use to decrypt the envelope in b. A threshold number of devices need to respond to the user in order for the user to create this decryption key.
2. A share of the user's envelope, which contains the user's public and private keys. This envelope is stored in the device as a map from the username to the envelope itself. This secret sharing scheme is based off of Shamir's Secret Sharing. Shamir's secret sharing allows for a piece of information to be shared among multiple parties, with a threshold number of parties needed to recover those shares. For example in a (t, n) threshold sharing scheme. Any t devices out of a total of n devices need to come together to reproduce the secret. If up to $t - 1$ devices are compromised, a hacker cannot recreate the secret.

The user will aggregate all of the outputs from a threshold number of devices to create a single β. The user will then de-blind this output by raising it to the power of $1/r$. The user then creates a randomized password, where $rwd = H(password + username, B^{\frac{1}{r}})$. H is a hash function with range $(0, 1)^{2\tau}$, where τ is a security parameter. This value rwd is able to decrypt the envelope because when the user first registers, it encrypts the envelope with rwd and then sends shares of the encrypted envelope to servers in the network. The user will also aggregate all the envelope shares sent to produce one complete envelope. This is done using polynomial interpolation. The user will then decrypt the envelope using rwd, which sends an alert to the network that the user has logged in. The user now has access to his private and public keys. This can be used to decrypt data that is stored locally on the user's device or to decrypt data that is sent to the user's device (this data is sent in similar fashion to the aforementioned envelopes – more details in full technical details implementation). This protocol protects against low entropy passwords (ie "1234") because the password is not stored on any device involved in authentication and the decryption key (rwd) is a function of a random output and a hash, which provides added randomness and bits of security. This protocol also protects against a single device compromise because the password is not stored on the device (the devices output a function of the password irrespective of the password itself) and because obtaining a share of the envelope or device data will give the attacker no information (must compromise a threshold number of devices).

2.b Blockchain Based Consensus Account Management

The devices that are in the network are not actively managed by a single entity. Rather, the devices must use a pre-defined protocol to manage which devices are involved in authenticating a user (we only need to use a threshold number of devices), devices coming in and out of the network, the parameters of threshold cryptography (if devices are coming in and out of the network, the threshold must change to accommodate security), handling devices that are faulty, and manage the state of the network.

To manage consensus in the network, this protocol uses Fast Byzantine Fault Tolerance. This algorithm ensures that nodes will end up agreeing on the state of the system.

When a device wants to join the network, they must first submit a provisioned identity (3b). Doing so allows the network to identify which devices are part of the network and to adequately take action on a faulty device. The devices then receive secret keys (4a). When a user sends a login request, the blockchain records this request, provisions which devices are involved in sending an output of $B_i = e(a, H_1(username))^(k_i)$ and the user's corresponding envelope/data, and records which devices are communicating with the user. When the device submits the output and corresponding envelope/data, BLS signatures are appended to the blockchain. Other devices in the network verify this signature to ensure that device sends the correct value. As a security mechanism to ensure same nodes are not involved every time in authenticating a user, we use a randomness generation protocol (3b,4a). This randomness is produced through a randomness generation protocol that uses BLS signatures.

When a device attempts to leave the network, it sends a notification to the blockchain. If another device is attempting to join the network, the old device can transport its information to the new device. If not, the old device can leave the network without any information being leaked because no one server can be compromised to leak information about a user and the threshold for recovering data is adjusted every certain number of blocks as devices come in and out of the network (3b).

The act of a user logging in and logging out are recorded on the blockchain to keep track of the user's session. In addition, when a user updates its data, the blockchain records that a state update has been made, but does NOT record the data itself on chain. Rather, this data is stored on the devices in the network. When a device is dishonest, it is flagged as faulty on the network.

3 Full Implementation

3.a User Authentication Protocol

A user chooses a username, password (pwd), public key (pub_U), and private key ($priv_U$). The user hides the password by sending exponentiated hash of the password and username, and their plaintext username to all devices on the network (this hash is the same as hashing into an elliptic curve). We denote this as $\alpha = H2(pwd)^r$. The devices verify that this value exists in group G_1 (if it does not, then honest servers halt, send an alert to the blockchain indicating an invalid request, and send a reply to the user for an invalid request). Each device then blinds the input of alpha with their secret key and sends it to the user. We denote this as $b_i = H1(\alpha, username)^k$. In addition, they append a BLS signature of their reply to the blockchain, which is verified by other devices in the network (More Details in Blockchain section). The user aggregates the outputs from a threshold number of devices, into a single $b = \alpha^k$, deblinds by raising it to the power of 1/r, and sets their randomized password to $rwd = H(pwd, b^{\frac{1}{r}})$. H is a hash function with range $(0,1)^{2\tau)}$, where τ is a security parameter. Using a randomized password protects against low security passwords and obviates the need for a password check. Once the user registers, they encrypt an envelope containing their private key and public key using the randomized password. The user then generates a (t, n) threshold secret sharing scheme of the envelope, where t is the minimum number of devices that come together to produce a response and n is the number of devices in the network. Each device then stores this key share in a map that correlates userId/username to the envelope. Since all values transmitted are encrypted (or have no reason to be encrypted – public keys) and reveal no sensitive information about the password or private keys, there is no need to establish a secure communications channel (ie. SSL/TLS).

When a user logs in, they select a random number, r (does not have to be the same as in registration), and sends $\alpha = H_1(pwd + username)^r$ and their plaintext username to at least $t + 1$ devices. Each device independently verifies that alpha exists in group G_1. If it does not exist in G_1, the node sends a message to the blockchain denoting an invalid request). Each device involved in authentication then sends a $b_i = \alpha^{k_i}$ to the user as well as their share of the envelope. Each device also appends a signature to the blockchain, which are aggregated and verified in totality by the leader node in the network. In the event of an invalid aggregate signature, each signature is verified individually and another node used in lieu of the dishonest node to provide a tpOPRF output and share of the envelope/data.

The user aggregates these values into a single $b = \alpha^K$, deblinds by raising it to the power of $1/r$, and sets their randomized password to $rwd = H(pwd, b^(1/r))$. In addition, the user regenerates the envelope from the shares and decrypts it using rwd to obtain its private key. Once the user is able to decrypt the envelope, they send a notification to t devices denoting the beginning of a login session. The private key can be used for decrypting and encrypting sensitive data. When the user finished their session they store and encrypt the current state of data on their device, as well as append it to the blockchain. The user then sends a logout message to a threshold number of devices.

This protocol protects against invalid passwords by using a master admin. When a user registers, a backup of their private key is transmitted via a secure communications channel to the admin device in the network. This private key is then encrypted by the admin and stored. If the user forgets their password, they must go through a two factor authentication scheme (ie. Use fingerprint, confirm email, etc) to request a reset. Once this is done, the user will receive their old decryption key, decrypt the data that is on their device (or received from the blockchain), and go through another registration protocol with a new password.

We periodically update the user's random password in 1 round of communication by sending the user an update token. The user can then update their randomized password by raising the output of the hash of the aggregated device response and password to the update token. For every request that the user makes to the devices on the network, only a threshold number of "honest" responses need to be given for the user to register and login. In doing so, we are able to solve our goals of having a distributed network of devices instead of a centralized server whilst also not taking up the entire compute power of the network.

3.b Blockchain Protocol

When the network is initialized, the tpOPRF curves are instantiated and stored on the blockchain. The curve to use is chosen at random by the initial source of randomness and is broadcast to all devices in the network. There is no need to update this curve, as the key for the curve is periodically updated, ensuring dynamic security.

For the devices to reach consensus on the network, this protocol uses Fast Byzantine Fault Tolerance. Here the leader of each shard runs a multi-signature signing process to collect other nodes' votes into one signature and broadcasts it. The block creation process works as follows:

1. The leader constructs a new block and broadcasts the header to all validators
2. Validators check block, sign it, and send it back to the leader
3. The leader waits for $2f + 1$ valid signatures and aggregates them into a one BLS multisignature
4. The leader broadcasts the aggregated BLS multisignature and a map that indicates which validators have signed
5. The validators check that the signature has $2f + 1$ signers, sign it, and send it back to the leader
6. The leader waits for $2f + 1$ valid signatures, aggregates them into a BLS multisignature, creates a map of all the signers, and commits a block with all signatures and the lists of signers

In order to securely redistribute nodes in shards, a randomness generation protocol must be used. Our randomness generation protocol uses Dfinity's decentralized randomness beacon. Here, randomness is generated at the beginning of every epoch (some predefined time period). Using the keys from the DKG protocol, a node computes a signature of the round, the previous rounds randomness, and their secret key. The randomness for the genesis epoch can be a hash of a string. Each node broadcasts this signature, which can be verified against the verification vector. Once a node receives t valid signatures, it recovers the group signature. The randomness is then computed as the hash of the group signature. This randomness generation protocol is void of leaders and is non-interactive.

There are no economic incentives to join the network; as such, we just use identity as a staking mechanism. Each device in the network uses a Samsung provisioned id. Each device in the network will only be provisioned one vote as they cannot have multiple identities.

The entire network is divided up into epochs (a predefined period of n blocks). At the beginning of each epoch, a new random number is generated and devices are shuffled between shards. New devices will not be able to join the network until the end of an epoch. When a device attempts to join, they are added to a queue that is stored on the beacon chain. When a device attempts to leave, they exchange keys and envelopes via a secure communications channel to a new device; however, this device will not be an active participant in the network until the next epoch. In the event that a device leaves without one to take its place, no security information is leaked. In the event that many devices leave the network in one epoch, every next device that leaves must transfer their key and envelope share to a trusted server, which will synchronize with the state of the shard and be a member of the network until the next epoch.

If more devices attempt to join the network than leave the network (or more devices leave than join), then the threshold number of nodes to maintain security will have to increase or decrease respectively. In this case, new DKG protocol will be run and shares for the envelope will be adequately adjusted (see proactive, adaptive secret sharing from above). When a user logs in after more devices enter the network, they will be sent an update token.

When a device joins a shard, they must synchronize to the current state of the shard. Devices download the current state trie of the shard. The device checks the signature of the first block in the current epoch, which has a hashed pointer to the first block of the previous epoch, and so on. Eventually, the device can verify blocks up till the genesis block.

At a high level, the blocks must have the following information: hash link to previous blocks, routes to nearest blocks, bls signatures, a session root: merkle tree of user session state user communication, account root: merkle tree of encrypted account state, state root: hash of signatures, signees, current nodes, and flagged nodes, a block header (hash of the above roots), block headers of all blocks, the current tpOPRF curve, and queue of nodes attempting to come in and out of the network.

3.c Encrypted Data Storage

We now describe the protocol for encrypted device storage on the network. The network provisions account data such that it can be accessed from other devices. When a user sends the encrypted state of their account to the beacon chain, the beacon chain leader creates a (m, n) Reed Solomon erasure encoding for the data, and transmits it to all nodes in the network, with the threshold being the number of nodes in the shard. M shards must send their data to the user.

This invention uses a routing protocol, such as Kademelia based routing. Each device in the network will maintain a routing table with devices from other shards. The distance between shards is the or distance of the shard ids. This has a time complexity of O(logn), compared to the O(n) time complexity of gossip broadcasting. Messages will need to be broadcasted from a leader to validators, from a device in the beacon chain to a device in each shard chain, and for devices across shards. During an epoch, if too many nodes leave a specific shard and not enough join, then shards will have to verify transactions across shards in order to keep up with the security of the network or the beacon chain must provision backup devices to take their position.

When a device is dishonest (ie, signatures are not verified, leaves the network without notifying other devices), it is flagged. When a node is flagged twice, it is kicked out of the network, with its provisioned id being an identifying factor for appropriate consequence.

There are two chains in the network: a beacon chain and a shard chain. The shard chains will be involved in consensus among the devices in the shard chain, login attempts, verifying signatures for authentication, and maintaining the encrypted state of the account. The beacon chain is also a shard in addition to being involved in random number generation at the beginning of the epoch, keeping track of the devices in each shard, provisioning when nodes come in and out of the network, rate limiting, broadcasting blocks, and coordinating login attempts. The beacon chain also has validators, similar to the shard chain. The beacon chain strengthens consistency of the network by verifying the block header and signers of a block in a shard chain. All shards will keep track of the block headers for the other shards in the event that they need to verify the validity of another shard.

4 Appendix

4.a Math

Our protocol uses BLS signatures for verifying outputs, which rely on bilinear pairings. We use an admissible bilinear pairing $e : G_1 \times G_2 \longrightarrow G_T$. Hashes H_1 and H_2 map onto G_1 and G_2 respectively. Groups G_1, G_2, and G_2 have generators g_1, g_2, and g_t respectively. In order to hide the password from the servers, we use oblivious pseudo random functions, which create random outputs that whose input is unknown to the computing party. More specifically, the curve that we use for the threshold partially oblivious pseudo random function (tpOPRF) is a pairings friendly curve. This curve has the unique property of the user only learning the output of the function and the server learning nothing about the user's or function's input. The threshold instantiation necessitates multiple entities to be involved in verification. The "partial" instantiation allows the output of the function to be verifiable by a third party via BLS signatures and the private key for the underlying OPRF to be updatable. Our curve is compatible with BLS signatures for verification and randomness generation due to it being pairings friendly. The tpOPRF function is defined as $F_k = e(H1(username), H2(password))^k$

BLS signatures allow for the output of the tpOPRF to be verified. We do so by interpolating each signature into an aggregate signature. This signature is verified against an aggregated group public key. If it fails, each signature is verified independently and the faulty node is realized.

In order to provision the private keys used for signing and the tpOPRF, we use a distributed key generation protocol. For a (t, n) threshold signatures scheme, a distributed key generation (DKG) protocol creates a group public key, individual private key share, and individual public key for each party. Our DKG protocol is an instantiation of the Joint-Feldman DKG protocol. Once the DKG protocol is finished, a verification vector is recorded to the blockchain.

The partial nature of our tpOPRF allows for keys to be updated. When two DKG's are run, each device has their old key (k_i) and new key $(k_{i'})$. We define delta as $\frac{k_f}{k_{f'}}$, where f denotes the aggregation of the secret key share. Given two secrets, a and b, we are able to generate the product $a * b$ without learning anything about the secret or product. Using this to our advantage, each device generates a product of p (randomly chosen prime number from group Z_q) and each key share: $r_i = p * ki \;\; r_{i'} = p * k_{i'}$. Each device sends the user these values and the user reconstructs $r = p * k_c \;\; r' = p * k_{c'}$. They are then able to generate $\delta = \frac{r}{r'}$. The client is then able to take the output of the old tpOPRF and set that to the power of $1/\delta$, giving a new decryption key.

To provision shares of the envelope, we take advantage of adaptive proactive secret sharing. This builds upon the established (t, n) Shamir's secret sharing. Adaptive proactive secret sharing allows for the envelope share to be periodically refreshed after every n number of block and also have the threshold itself be updated.

4.b Figures

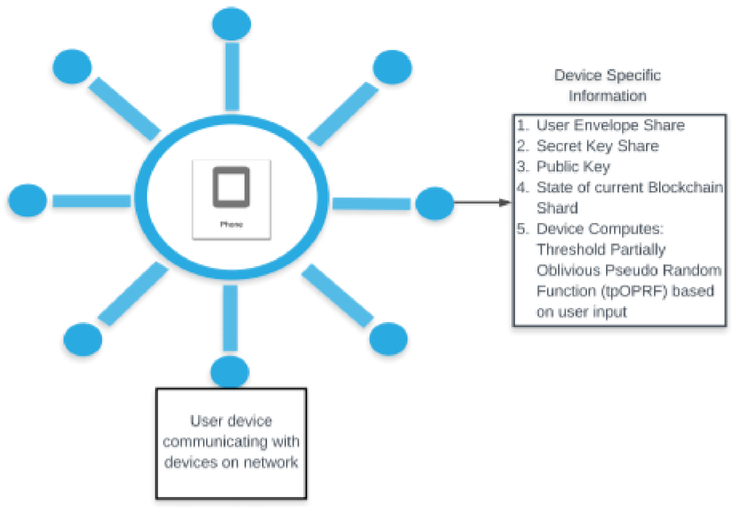
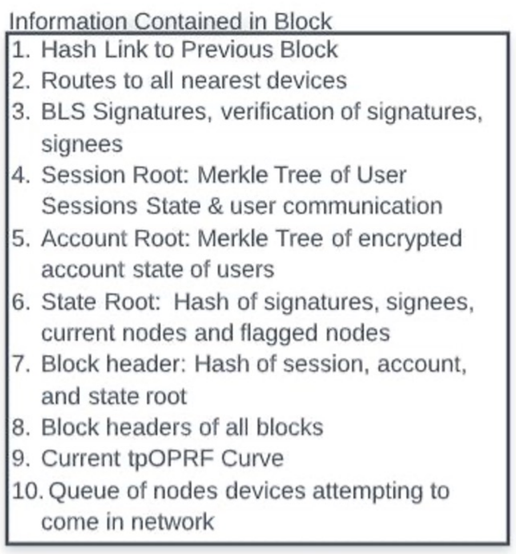

Device Specific Information
1. User Envelope Share
2. Secret Key Share
3. Public Key
4. State of current Blockchain Shard
5. Device Computes: Threshold Partially Oblivious Pseudo Random Function (tpOPRF) based on user input
Phone
User device communicating with devices on network

Figure 2: Device Communication with Network



Information Contained in Block

1. Hash Link to Previous Block
2. Routes to all nearest devices
3. BLS Signatures, verification of signatures, signees
4. Session Root: Merkle Tree of User Sessions State & user communication
5. Account Root: Merkle Tree of encrypted account state of users
6. State Root: Hash of signatures, signees, current nodes and flagged nodes
7. Block header: Hash of session, account, and state root
8. Block headers of all blocks
9. Current tpOPRF Curve
10. Queue of nodes devices attempting to come in network

Figure 3: Block information

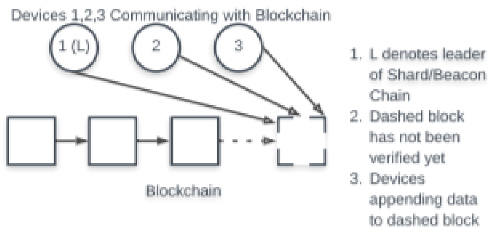

Devices 1,2,3 Communicating with Blockchain
1 (L)
2
3
1. L denotes leader of Shard/Beacon Chain
2. Dashed block has not been verified yet
3. Devices appending data to dashed block
Blockchain

Figure 4: Client Communication

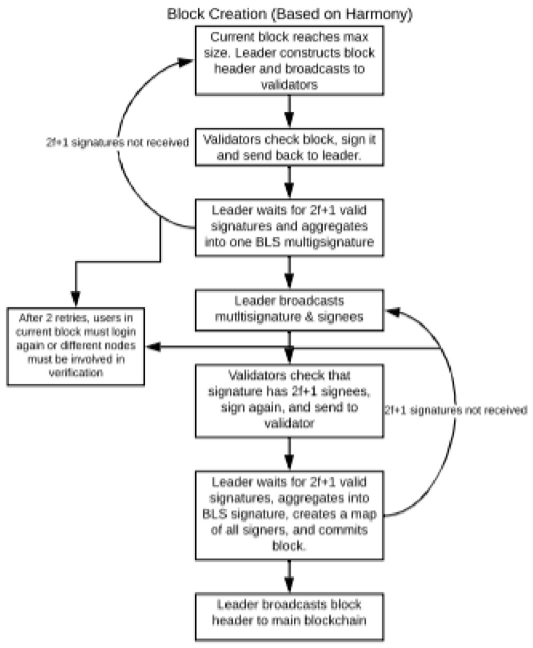

Block Creation (Based on Harmony)
Current block reaches max size. Leader constructs block header and broadcasts to validators
Validators check block, sign it and send back to leader.
2f+1 signatures not received
Leader waits for 2f+1 valid signatures and aggregates into one BLS multigsignature
Leader broadcasts multisignature & signees
After 2 retries, users in current block must login again or different nodes must be involved in verification
Validators check that signature has 2f+1 signees, sign again, and send to validator
2f+1 signatures not received
Leader waits for 2f+1 valid signatures, aggregates into BLS signature, creates a map of all signers, and commits block.
Leader broadcasts block header to main blockchain

Figure 5: Block Creation

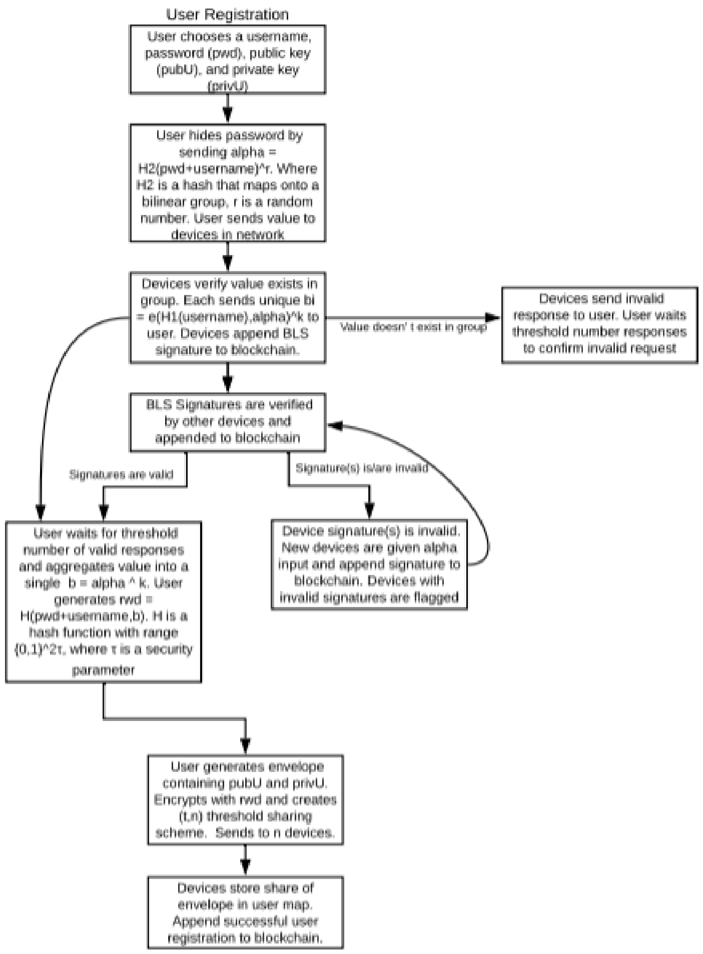

User Registration
User chooses a username, password (pwd), public key (pubU), and private key (privU)
User hides password by sending alpha = H2(pwd+username)^r. Where H2 is a hash that maps onto a bilinear group, r is a random number. User sends value to devices in network
Devices verify value exists in group. Each sends unique bi = e(H1(username),alpha)^k to user. Devices append BLS signature to blockchain.
Value doesn' t exist in group
Devices send invalid response to user. User waits threshold number responses to confirm invalid request
BLS Signatures are verified by other devices and appended to blockchain
Signatures are valid
Signature(s) is/are invalid
User waits for threshold number of valid responses and aggregates value into a single b = alpha ^ k. User generates rwd = H(pwd+username,b). H is a hash function with range {0,1}^2τ, where τ is a security parameter
Device signature(s) is invalid. New devices are given alpha input and append signature to blockchain. Devices with invalid signatures are flagged
User generates envelope containing pubU and privU. Encrypts with rwd and creates (t,n) threshold sharing scheme. Sends to n devices.
Devices store share of envelope in user map. Append successful user registration to blockchain.

Figure 6: User Registration Flow

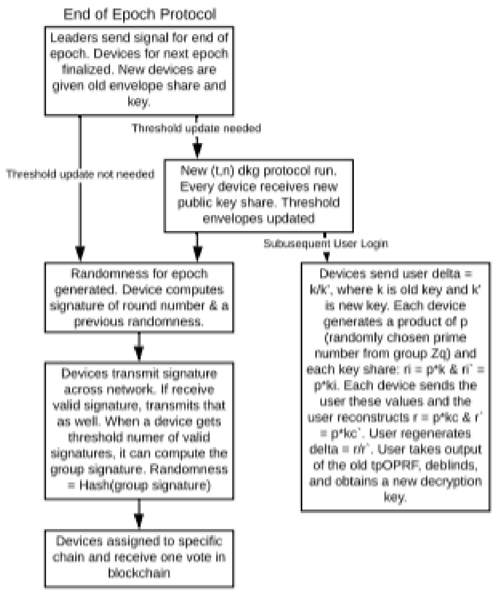

End of Epoch Protocol
Leaders send signal for end of epoch. Devices for next epoch finalized. New devices are given old envelope share and key.
Threshold update needed
Threshold update not needed
New (t,n) dkg protocol run. Every device receives new public key share. Threshold envelopes updated
Subusequent User Login
Randomness for epoch generated. Device computes signature of round number & a previous randomness.
Devices send user delta = k/k', where k is old key and k' is new key. Each device generates a product of p (randomly chosen prime number from group Zq) and each key share: ri = p*k & ri` = p*ki. Each device sends the user these values and the user reconstructs r = p*kc & r` = p*kc`. User regenerates delta = r/r`. User takes output of the old tpOPRF, deblinds, and obtains a new decryption key.
Devices transmit signature across network. If receive valid signature, transmits that as well. When a device gets threshold numer of valid signatures, it can compute the group signature. Randomness = Hash(group signature)
Devices assigned to specific chain and receive one vote in blockchain

Figure 7: Epoch Transition

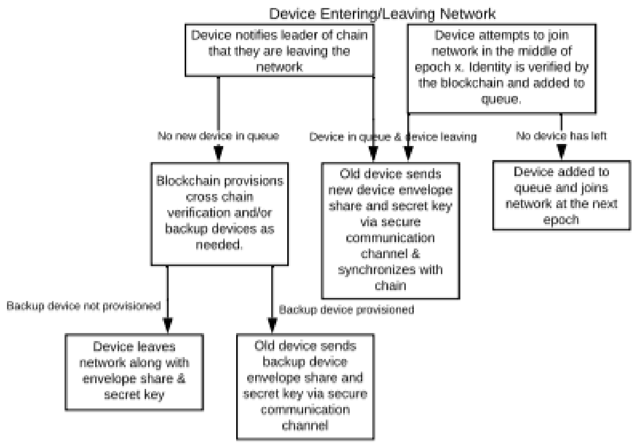

Device Entering/Leaving Network
Device notifies leader of chain that they are leaving the network
Device attempts to join network in the middle of epoch x. Identity is verified by the blockchain and added to queue.
No new device in queue
Device in queue & device leaving
No device has left
Blockchain provisions cross chain verification and/or backup devices as needed.
Old device sends new device envelope share and secret key via secure communication channel & synchronizes with chain
Device added to queue and joins network at the next epoch
Backup device not provisioned
Backup device provisioned
Device leaves network along with envelope share & secret key
Old device sends backup device envelope share and secret key via secure communication channel

Figure 8: Device Tracking