Blade: a protocol for tokenizing device security and delegation of trust

Hrishikesh Huilgolkar
DRAFT revision 1
July 11th 2018

Abstract

Current decentralized blockchain networks such as Ethereum, Bitcoin and others have scalability limits. Several layer 2 scalability solutions such as plasma, plasma cash, payment channels and state channels have been proposed and are being developed. State channels allow large number of transactions between various parties with low cost and low latency. Such transactions are 'guaranteed' by the smart contracts deployed on the main blockchain.

Due to various reasons, primarily due to mistrust between various participants such transactions cannot be settled on the root blockchain instantly and can take between several days to several weeks to settle. This paper presents a novel protocol **Blade**, which eliminates requirement of settlement periods mainly by (i) using traditional hardware wallets to emulate 'Trusted Execution Environments' or by adding incentivization mechanism for secure, correct execution and honest behaviour of devices with TEEs (ii) by **delegating trust** to wallet issuers for correct execution and honest behaviour guarantees.

1. Introduction

A popular and promising scalability solution for blockchains includes payment channels, or the generalised version of it, called state channels. State channels allow large number of off-chain transactions between participants with only two on-chain transactions. One for opening the channel and one for settling the channel.

Usually in state channels, there is mistrust between different participants due to the possibility of fraud. If the participants detect fraud, they provide fraud proofs to an on-chain smart contract to settle dispute. However sufficient period must be provided to the human participants so that they can detect fraud, prepare a fraud proof and submit it to an on-chain smart contract. Due to this reason, such games have 'settlement periods' which can range from few days to a few weeks. This friction deters players from participating in such games with high value.

In other 'side chain'-like scaling solutions such as side chains, plasma, plasma cash, etc. settlement of transactions is done in batches on the main chain. Plasma chains may be nested multiple layers deep and it can take multiple weeks to be fully settled on main chain. The delay is added to reduce number of on-chain transactions and provide time for participants to exit chain in case of fraud.

This paper presents Blade, which eliminates requirement of settlement periods as under normal circumstances, it is not possible for players to cheat. This is guaranteed by the wallet issuers with a stake.

Currently there is currently no in-protocol incentive for hardware wallet issuers to not introduce backdoors which steal users' private keys. Blade adds a strong monetary disincentive for hardware wallet issuers to introduce backdoors, security vulnerabilities, etc.

Blade uses combination of smart contracts and in-device software firmwares and cryptography to guarantee correct execution. Due to this, settlement periods are unnecessary and games such as state channels can be settled on-chain instantly without trusting any party.

Blade supports any blockchain with support for turing complete smart contracts. It is a general purpose framework and can support payment channels, state channels and other similar games. For the rest of the paper we will explore using Blade for creating payment channels using ethereum. In later part of the paper, we will explore other potential applications.

2. Background and motivations

Previous works to solve this problem include TEECHAIN for Bitcoin and other cryptocurrencies. It uses Trusted Execution Environments such as intel's SGX (Software Guard Extensions) to guarantee correct execution with Memory and storage isolation and remote attestation. While this method is novel and effective, it requires trust in hardware manufacturers and their proprietary hardware designs. Hardware issuers have reportedly included backdoors in their devices deliberately for various reasons. The hardware manufacturers do not have a strong disincentive to include such backdoors. Also the remote attestation service requires a trusted centralized authority in case of Intel. Intel can stop this service or it can behave in a byzantine manner and disrupt services relying on it.

In addition to this, devices which include TEE can be expensive and out of reach of average consumers. We have not seen many smartphone devices with TEEs and they face similar issues. Blade protocol is independent of hardware and can be run in any device, including ones with TEE, and provides same game theoretical guarantees and security.

The protocol is decentralized and permissionless and allows anyone to issue wallets which use this protocol. We will call them 'wallet issuers'. The wallet can be a hardware wallet manufactured by wallet issuers, or it can be manufactured by a third party and initialized by wallet issuers. Wallet issuers may also choose to distribute software to remotely initialize users' devices having Trusted Execution Environments.

Wallet issuers are free to use open source or proprietary hardware wallets with a firmware which implements Blade protocol to participate in the network. It is responsibility of wallet issuers to ensure that the hardware implements the protocol securely to guarantee security and correct execution. The wallet issuer will be punished by the smart contract if it detects any form of fraud or misbehaviour by the wallet.

Wallet issuers are incentivised as they can use different models to earn profit. For example, they can charge transactions for off-chain transactions, charge a recurring insurance/subscription fee, etc.

3. BLADE overview

3.1 Scenario

We will consider a scenario where two mutually distrusting parties want to exchange funds using a payment channel. The parties are connected to each other using an insecure network and have access to the ethereum blockchain.

3.2 Approach

The fundamental idea behind Blade is to use hardware wallets as pseudo 'Trusted Execution Environments' regardless of the true nature of hardware. Wallet issuers generate

an ‘Issuer private key’ in each wallet which is unique to the wallet and may not be known by the wallet issuers. Wallet issuers only take the public part of the issuer private key and provide it to their staking smart contract. Staking contract holds a large amount of funds staked by the wallet issuer to guarantee the security and correct execution. This key is used to identify the wallet issuer and her staking contract for remote attestation and for exchanging encrypted messages.

Blade then allows the participants to execute a protocol to construct bidirectional payment channels and to exchange payments in a peer-to-peer manner via those channels. To ensure the correct operation of these payment channels, the participant’s remotely attest each other’s wallets.

3.3 Threat model and assumptions

Our threat model assumes that two parties want to exchange funds but mutually distrust each other. They do trust wallet issuer as long as no fraud has been committed by her and it has not been observed on chain and is providing sufficient stake to compensate the parties in case of fraud. The parties may behave in a byzantine manner and may broadcast invalid and malicious messages to each other, their devices and to the blockchain. They may drop, tamper or replay messages arbitrarily. They may try griefing attacks to make other party wait, or make them lose funds at a cost to themselves. Fig. 1 shows the trust assumptions made by two parties in a Blade payment channel.

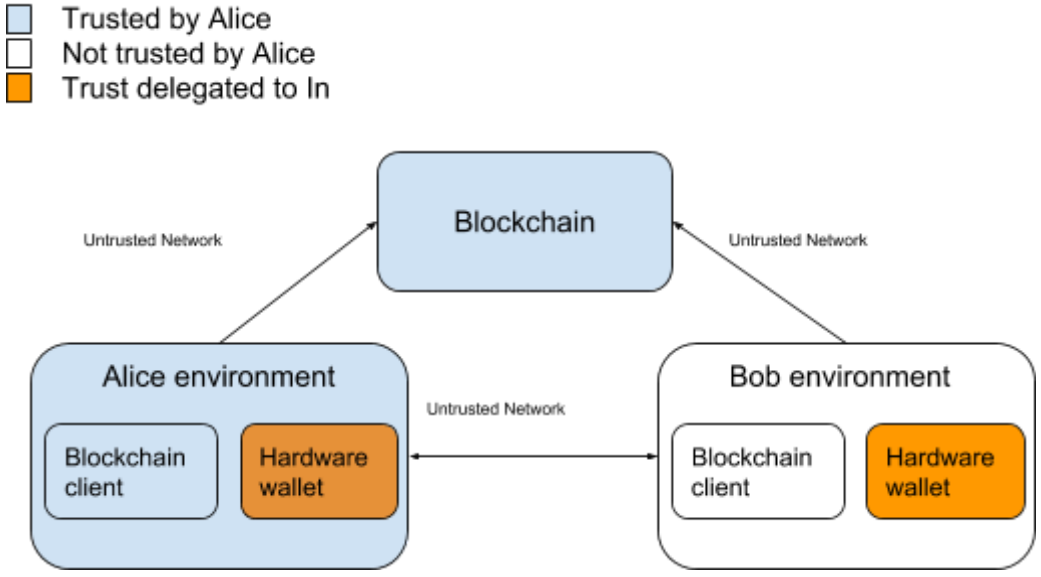

Figure 1: Trust model between Alice and Bob. Alice trusts Bob’s wallet as long as no fraud is reported according to on chain smart contract.

Each party trusts the blockchain, its own environment, the local and remote wallets by the virtue of Blade protocol. The rest of the system, including the network channels and the other parties’ software stacks (outside the wallet) are untrusted. During protocol execution, any party may therefore access or modify any data in its host system memory or stored on disk, view or modify its application code, and control any aspect of its operating system and other privileged software and hardware.

Ingrid (not shown in the figure) uses a general purpose computing environment to access the blockchain and to perform computation. She does not need a hardware wallet or a trusted execution environment.

3.4 Protocol overview

In Blade protocol, each participant operates their own wallet inside a hardware wallet supporting Blade protocol. The wallet issuer, however does not need a device supporting blade protocol for herself.

Payment channel using Blade works as follows:

1. First, wallet issuer initializes the devices and securely generates a issuer private key inside the device. She then provides the public key associated with this private key to an on-chain smart contract, where she has staked her funds.
2. Pairs of parties who want to open channel perform remote attestation and open bidirectional payment channels. Before a party may send funds over such a channel, they must deposit the funds in the payment channel smart contract. All deposits will be collateralized by funds of Wallet issuer to guarantee compensation in case of security compromise.
3. While the channel is open, any number of transactions can be performed by the parties. The wallets correctly remember the latest off-chain transaction performed by the user in each channel.
4. A channel participant is free to close the payment channel any time by requesting the wallet to sign the settlement transaction. The wallet will securely sign only the latest valid off-chain transaction stored in the wallet with the issuer private key. Since this removes the possibility of cheating, the state channel can be closed instantly.

3.5 Restrictions on wallet

To participate in the protocol, certain restrictions are placed on the wallet and its users:

1. User is not able to recover the private key
2. User is not able to recover the issuer private key
3. User cannot deploy arbitrary smart contracts
4. User cannot sign arbitrary messages with her private key or with her issuer private key
5. Restrictions may be placed on users ability to sign on-chain transactions depending on local state of the wallet.

4. Blade protocol

In this section we will describe the Blade protocol. First we will describe Wallet issuer side initialization, then user side initialization, establishing the payment channel, exchange of funds in the channel, termination of payment channel using issuer private key and finally termination of payment channel without issuer private key. We will also explore various attacks and how they can be overcome with the protocol.

4.1 Staking contract

The protocol is decentralized and permissionless. Anyone can be a wallet issuer. To be a wallet issuer, the participant, say Ingrid, has to deploy a contract where she will stake a certain amount of funds to guarantee correct execution of her wallets:

1. Ingrid sends a request to create her contract to Blade wallet issuer registry. The registry deploys a new smart contract for her and adds it into the registry.

2. She then deposits any amount of funds, above a minimum amount of required funds, in the smart contract as a stake
3. All the payment channels created using her wallets will interact with this contract

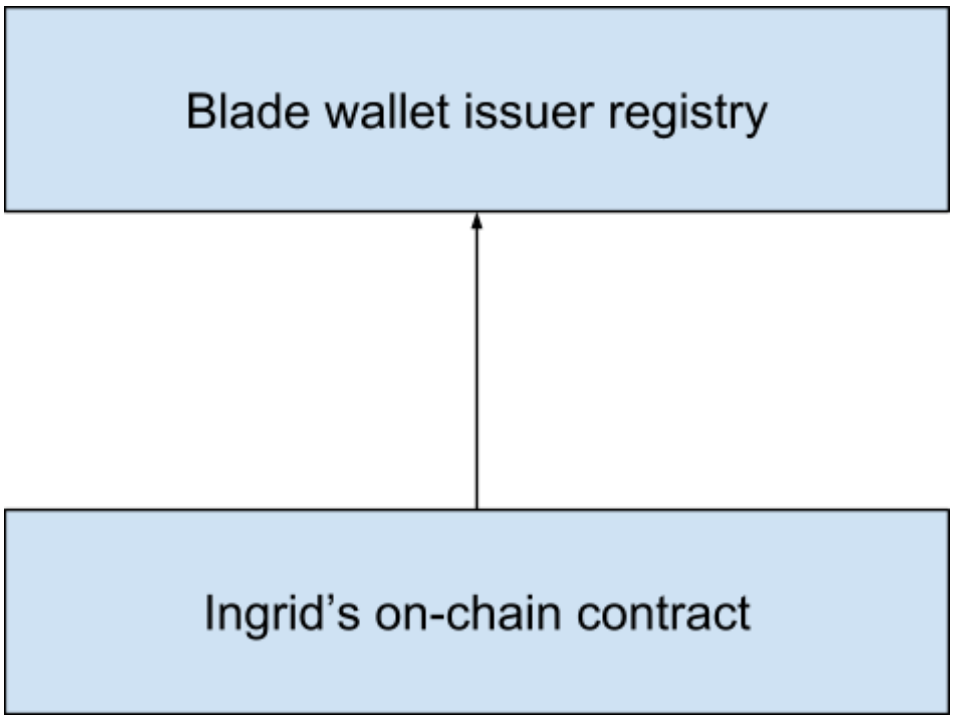

Figure 2: Each wallet issuer contract is registered with blade wallet issuer registry onchain

4.2 Wallet issuer side initialization

Ingrid has to take following steps to initialize the wallets from her side:

1. Ingrid can either:
    a. Manufacture her own hardware wallet
    b. Use off the shelf secure devices (such as java smart cards)
    c. Create apps which work with Desktop and Smartphone TEEs

We will assume case b for the rest of the paper as we find it to be most economical, secure and accessible.

2. She then uploads a firmware to the device. The firmware can be official Blade open source firmware or any custom firmware conforming to the protocol. It should be signed by the developer of the firmware so users can evaluate authenticity and version of the firmware.
3. She initializes the wallet. During the initialization, a issuer private key is generated inside the device, and only public key is provided to her by the wallet. This issuer private key can never be accessed by anyone once initialized and is not recoverable.

Note that it is possible that Ingrid gets access to this issuer private key, but it does not compromise the security of the protocol.

4. She then regularly publishes the merkle root of the tree containing all the public keys she initialized in devices. Ingrid will also periodically publish this merkle tree on a public channel.
5. She then sells the wallets to users Alice and Bob

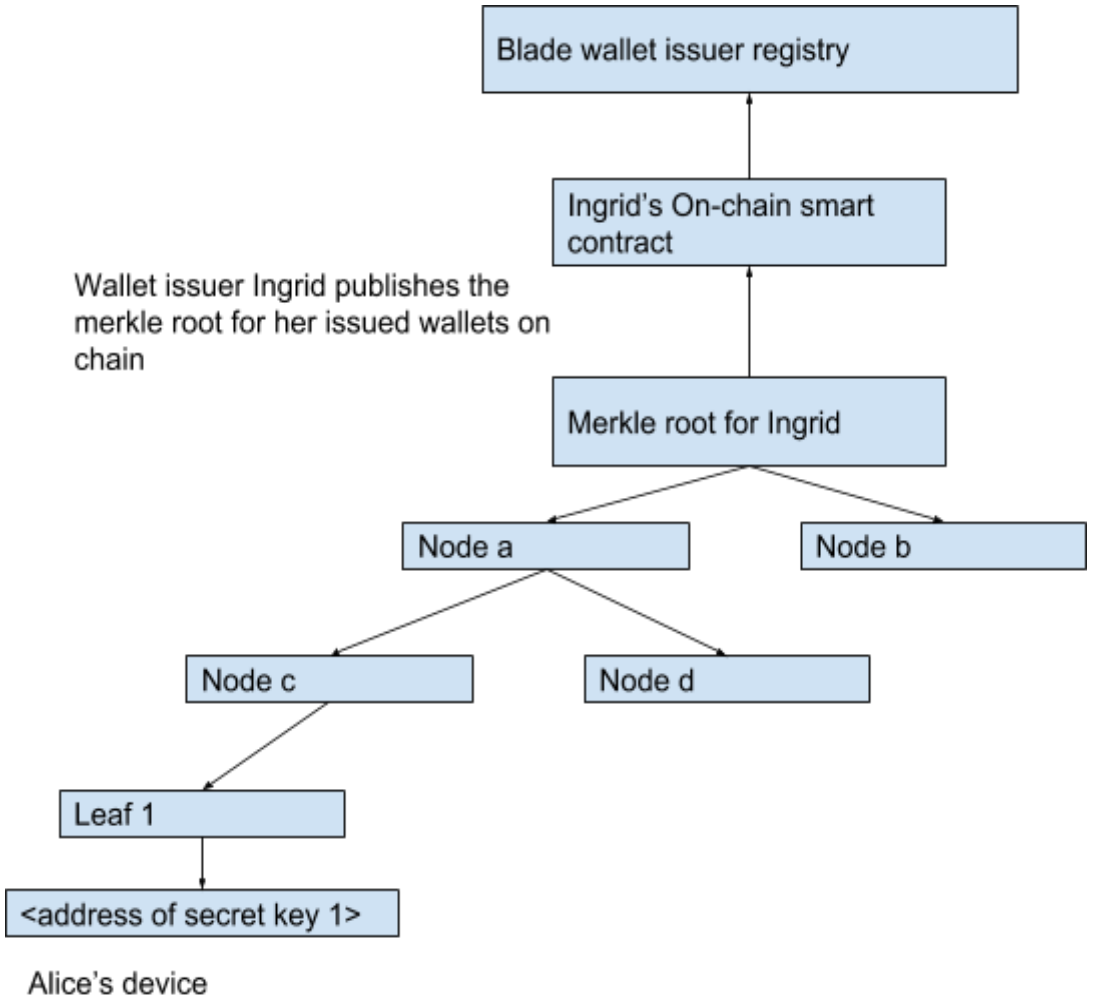

Figure 3: Wallet issuer Ingrid creates a merkle tree of addresses of issuer private keys present in wallets issued by her. She publishes the root regularly as well as the tree is published off-chain so that anyone can verify validity of the tree.

4.3 User side initialization

Alice, upon receiving the device takes the following steps:

1. She gives a randomly generated challenge to the wallet, wallet then makes sure the message is in a specific format and signs the challenge with the issuer private key and gives it to Alice.
2. Alice looks up the on-chain registry to check whether the public key is valid and is included in the merkle tree whose root is published by Ingrid on chain.
3. Alice generates her personal private key inside the wallet and notes her public key/address.

Note: wallet does not provide the private key or a seed phrase to generate it to anyone including Alice. Hence Alice must take care and store only small amount of ether in this address and store majority of funds in on-chain smart contract which has recovery feature.

4. Alice then creates a channel with Ingrid by deploying a smart contract, making a deposit and signing a message using the issuer private key. She also provides merkle proof proving that the public part of issuer private key is part of the markle

tree's root. This proves that she has a wallet provided by Ingrid. Any other wallet cannot be part of the channel.

We will explore how payment channels can be established across customers of different wallet issuers later. Also we will explore later how it is possible to establish payment channels in Blade without on-chain transactions.

Bob takes the same steps as Alice to initialize his wallet and open payment channel.

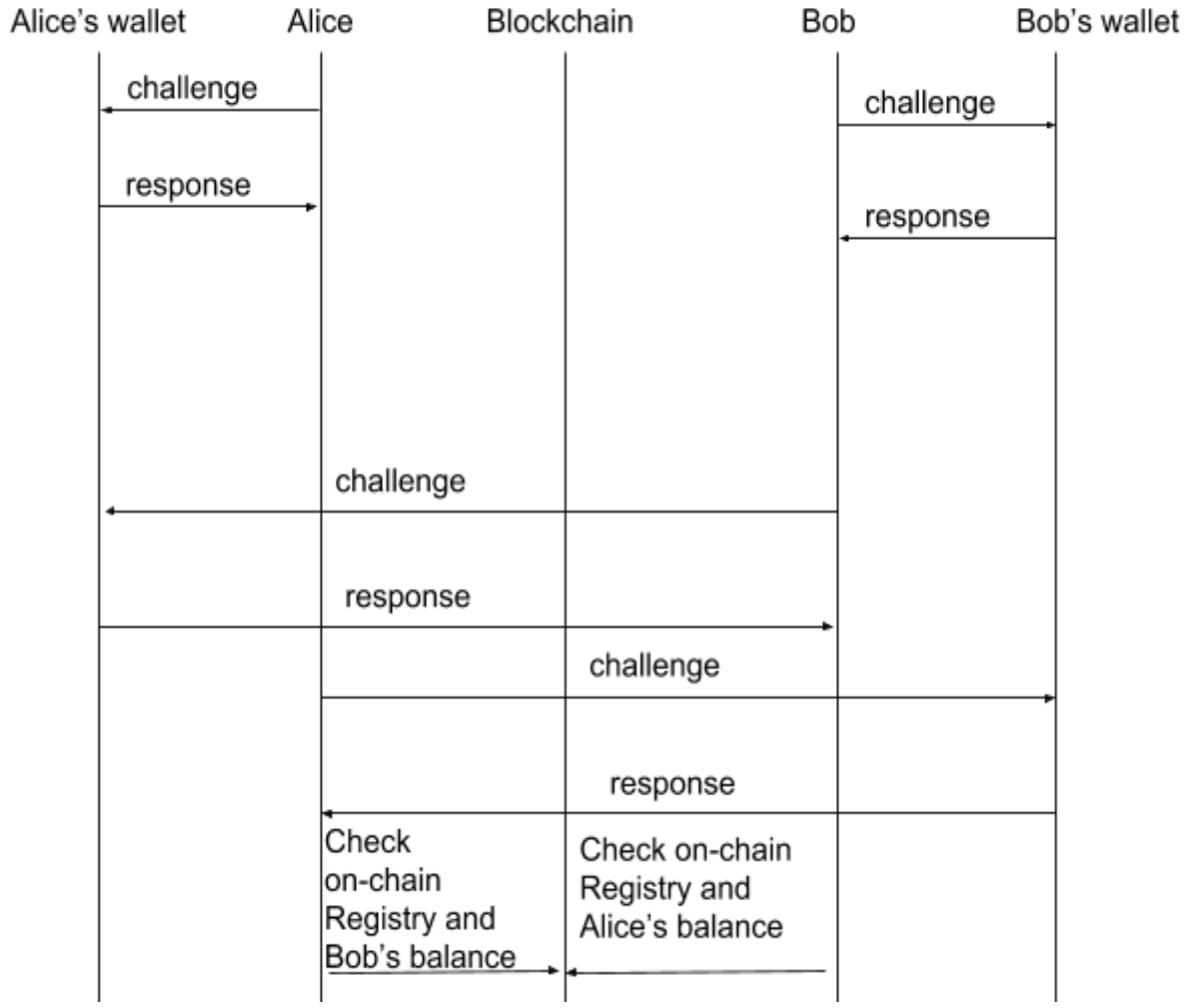

Figure 4: self attestation and remote attestation

Once Alice and Bob initialize their wallets and open channels with their respective wallet issuers, they can start exchanging funds off chain.

1. To send funds to Bob, Alice signs a message of following format:

   send,<nonce>,<Bob's ethereum address>,<amount>,<token contract address else 0 for eth>,<payment channel address>,<total locked amount>

   Here, total locked amount is total amount of locked tokens of Alice of the same type of token in all channels created by this wallet. This is necessary so that Bob can check Alice's on-chain balance and make sure Alice has sufficient funds and is not double spending.

The message can be encoded in hexadecimal format for security and performance reasons.

2. The message will be validated by Alice's wallet and saved in its secure persistent memory. Any message of lower nonce will be removed from its memory.
3. Alice can send this signed message to Bob in an insecure network.
4. Bob can verify this message with information available on-chain and can check her signature against her address.
5. Bob provides the message to his wallet. The wallet verifies the validity of the message and saves it in its secure storage.

   If Bob doesn't perform this step, he may lose some funds but he cannot make Alice or wallet issuer lose funds.
6. Bob can make a transfer to Alice in the same way. His message will be replace the older message signed by Alice since it will have a higher nonce.

4.4 Payment through the Channels

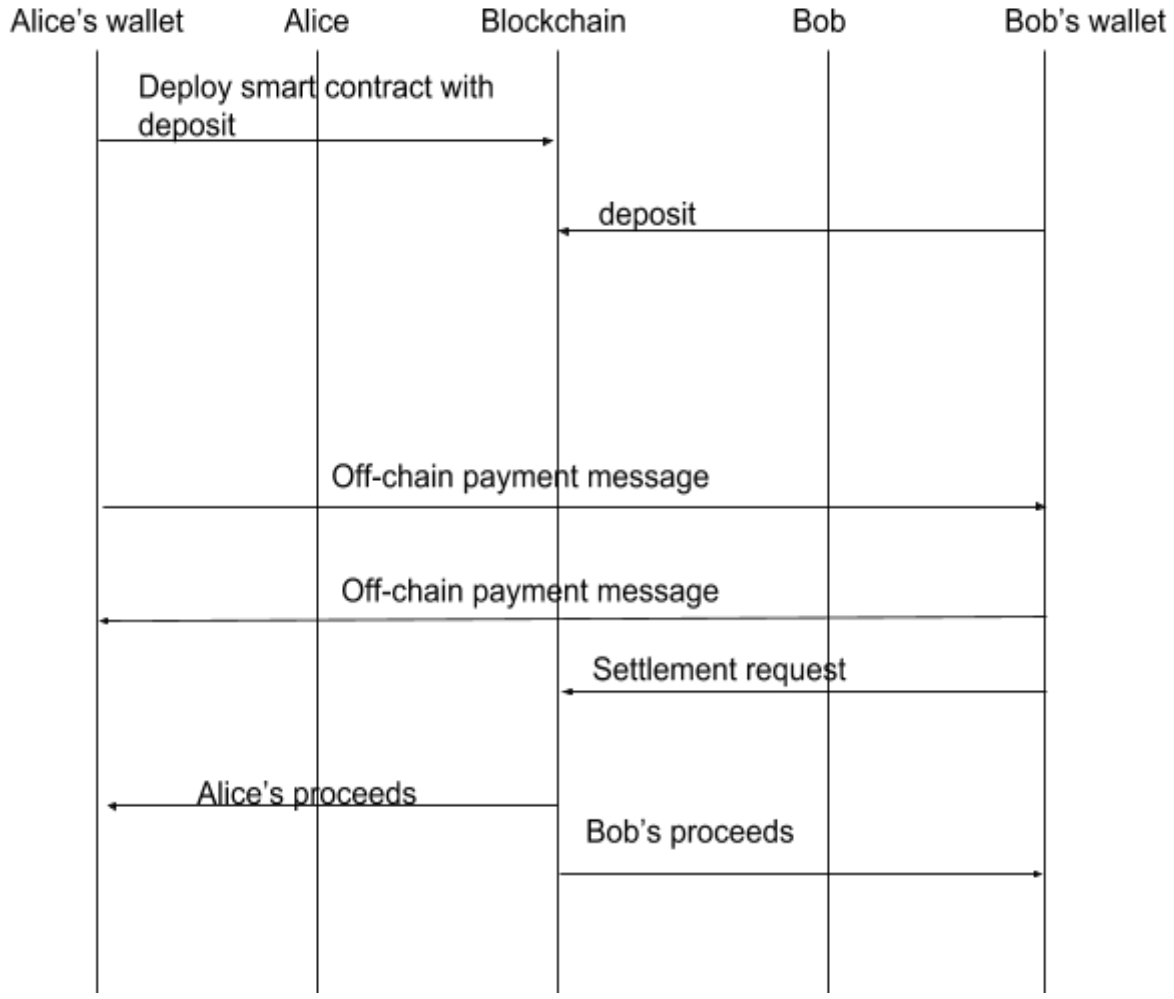

Figure 5: Opening channel, off-chain transactions and settlement.

Note that there is no delay or settlement period after settlement request is sent by any party. Settlement request and whole settlement happens in the same transactions. It is separated for clarity

4.5 Graceful termination of payment channels with issuer private key

Alice and Bob can instantly close and settle the payment channel on chain at any time. Let's say Alice wants to settle the channel:

1. Alice asks the wallet that she wants to **settle.**
2. The wallet asks Alice if she received any off-chain messages later than the one currently stored on the device.
3. Alice provides any such message she received to the wallet. If she does not perform this step, she may lose some of the funds she recently received.
4. Her wallet verifies and signs this message with issuer private key. If no message is provided, it signs the latest message stored in the device with issuer private key.

   Do note that the issuer private key is restricted to sign only certain messages as instructed by the firmware. Alice cannot use the issuer private key to sign any arbitrary message.
5. Alice then submits this message signed by her and the issuer private key to the smart contract. She also submits the merkle proof proving that the public part of the issuer private key is contained in the merkle tree of the merkle root published in the smart contract.
6. Smart contract verifies the message and the proofs provided and closes the channel. It then sends the proceeds to the parties instantly.

4.6 Termination of payment channel without issuer private key

It is possible that Alice or Bob lose their wallet. They may initialize a new wallet supporting blade protocol or not use any specialized hardware and recover their key on any wallet. But it is not possible for them to recover the issuer private key since they never had access to it. Alice can take following steps to close the channel and withdraw her funds from the channel.

1. Alice recovers her private key on her computer using backup phrase.
2. Alice asks the smart contract to close the payment channel. She sends the latest signed message to claim balances.
3. The smart contract starts a long waiting period (Lets say 30 days) since Alice did not provide a message signed by issuer private key.
4. Bob can submit a settle message signed by his issuer private key to instantly close the channel.
5. If Bob also lost his issuer private key, he can submit a message signed by Alice which has a higher nonce, if he has a higher nonce. Since claims have been made by both the parties with evidence, the channel can now close.
6. If Bob does not respond, the channel will be closed after the waiting period.

4.7 Storing of state inside the wallet

To make sure the owner of the wallet does not cheat, the wallet needs to remember messages signed by user for various games. However hardware wallets, especially smart

cards have extremely limited resources, which limits number of games user can simultaneously participate in.

To overcome this limitation, we will use merkle trees to track the local state of the wallet. User will store the merkle tree outside the wallet on his device. Only the merkle root will be stored in the wallet.

This section is optional and can be skipped if user is planning to participate in only one or few games simultaneously. It does not affect the security or operations of the protocol.

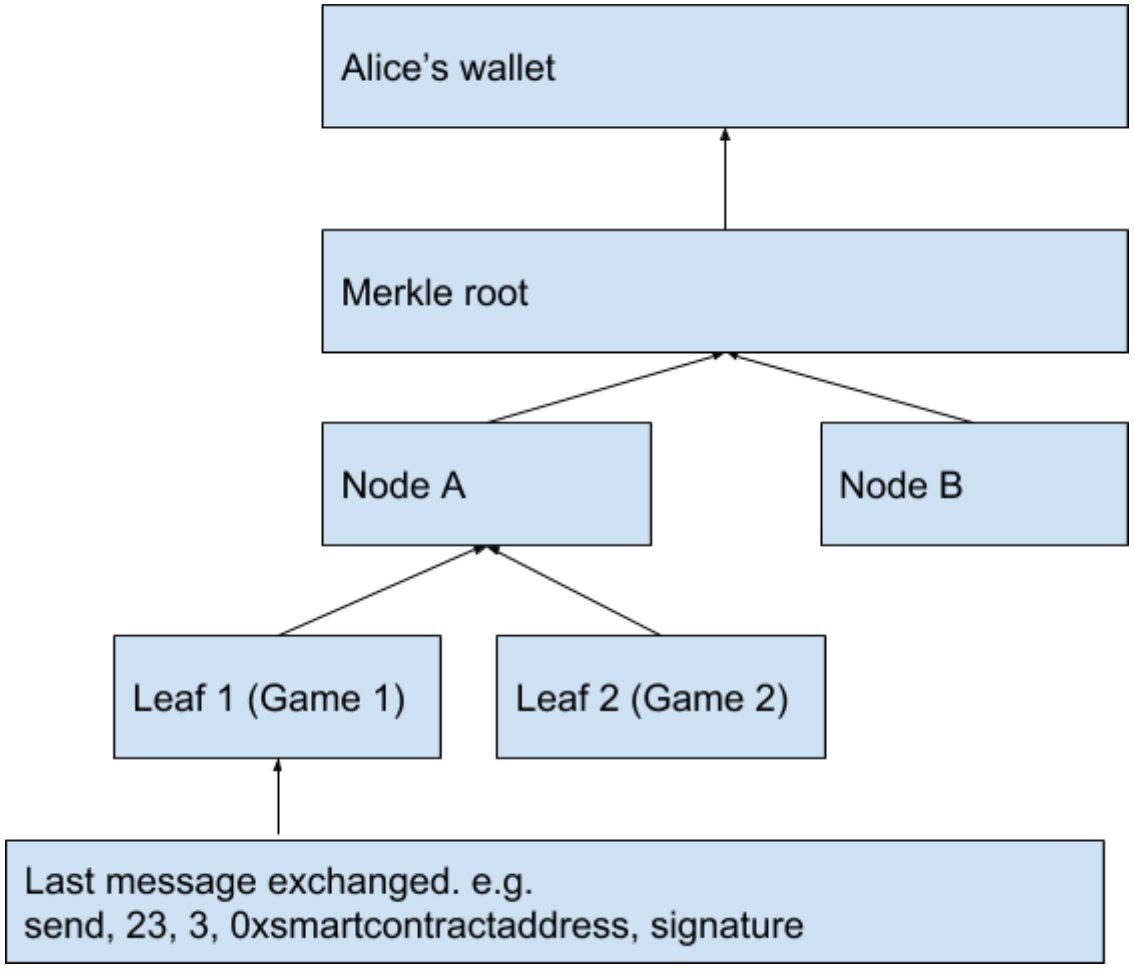

Figure 6: merkle tree used to store state of games in Alice's wallet

Suppose Alice receives a new message from Bob which increases her balance in game 1 and decreases Bob's balance. To update the state in Alice's wallet she will have to provide her wallet following data:

1. Merkle consistency proof of old Leaf 1
2. Merkle audit proof for old Leaf 1
3. Merkle consistency proof for new Leaf 1

Using this information along with merkle root stored in the wallet, it can infer following things:

1. The old leaf is present in merkle tree
2. The old leaf and new leaf belong to same game
3. The new leaf has a higher nonce than old leaf
4. It can calculate the new merkle root and provide it to the user

Alice can take the same steps outside the device to have a copy of new merkle tree. She can verify if the wallet's calculations were correct.
When Alice wants to settle the game/channel, she will have to provide merkle consistency and audit proofs of the concerned leaf.

4.8 Fraud proof about wallet issuer

Since the security of the protocol relies heavily on the correct behaviour of the wallet issuer, her incentives and disincentives are carefully designed to avoid fraud.
The wallet issuer can perform following misbehaviours compromising the secure operation of the protocol and users wallets. Anyone can provide the fraud proof which will close all channels of the issuer with long settlement periods and the issuer will be monetarily penalized. If any unavoidable loss of funds happens, the user will be compensated from this account.

1. Wallet issuer can keep a copy of all issuer private keys. This is possible and undetectable. However it will not give any benefit to the wallet issuer and instead can create additional risks for her.
2. Wallet issuer can create backdoors in her wallet. If it is used to extract issuer private key, it has the same implications as 1. If backdoors are created to extract user's private key, it is not secured by the Blade protocol and their funds can be stolen. Hence users should make sure not to keep substantial amount of funds in their private keys. Keeping funds in payment channel is more secure and convenient for them anyways.
3. If wallet issuer does not carefully secure the issuer private key in the wallets and if one of them is compromised by a malicious actor, the wallet issuer will be punished by the contract.
4. We will explore more security implications in 5.

4.9 Exchange between users with different wallet issuers

1. In case the wallet issuers of two parties are different, users will have to monitor behaviour of the counterparty's issuer and counterparty's messages for misbehaviour and fraud.
2. In case of fraud, user has to send fraud proof to counterparty's smart contract to stop its operations and get compensation/reward.

4.10 Using counterfactual channels to create and close channels with no on-chain transactions

We will describe how it is possible to create counterfactual payment channels with just 2 on-chain transaction in worst case. In best case where initial and final balances of the channel are same, no on-chain transactions are necessary.
We will describe unidirectional payment channel, however bidirectional payment channel can also be deployed.
A central contract will be used to see if the channel is open or settled. It will map sha256(user1,user2,channelnumber) -> status (bool)

Also in this example, we will assume initial funds are stored in private key controlled by the wallet and not in on-chain smart contracts.

Case I: Alice wants to close the channel

1. Alice and Bob do initialization as described in 4.3
2. Alice signs following ethereum tx which will deploy a new contract with X tokens which will allow Bob to redeem onchain on presenting valid state channel proof

An smart contract containing code to: Send x tokens to anyone who sends a valid offchain payment message, if channel is not settled already.

Alice will sign this same transaction Y times with nonces in range n to n+Y, where n is Alice's current blockchain nonce.

3. Alice sends these transactions to Bob without deploying on-chain.
4. Bob verifies these transactions with information available on blockchain.
5. Alice can send signed messages which will allow Bob to redeem eth.
6. Alice's wallet will lock X tokens from her balance and also will not let her sign transaction of nonce n+Y unless the channel is closed.
7. Alice wants to close channel. Her wallet determines the correct amount and deploys the smart contract on chain which instantly settles the balance in same tx by sending Bob's proceeds to him. It also updates the central channel registry and closes the channel.
8. Alice's wallet unlocks her funds and removes nonce restrictions.
9. The settlement is instant and no action from Bob is required.

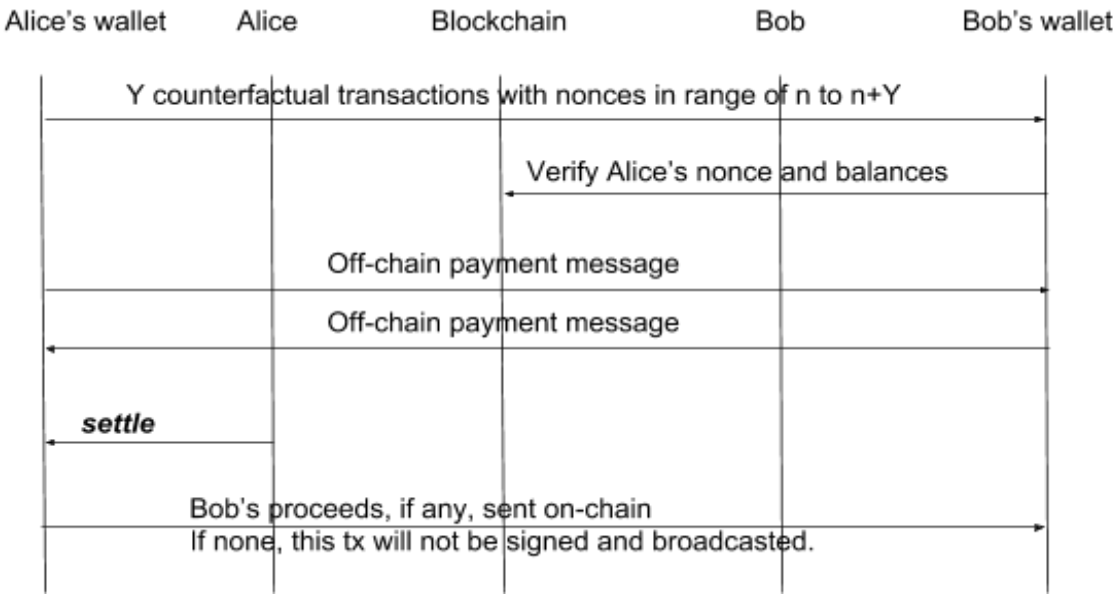

Figure 7: Case I Alice wants to close channel

Case II: Bob wants to close the channel

In case Bob wants to close the channel.

1. The channel opening and offchain transactions are same as Case I.
2. Bob has to broadcast the counterfactual contract signed by Alice with correct nonce m+1 where m is Alice's current on-chain nonce.
3. Bob has to send valid message sent by Alice to the smart contract signed by his issuer private key. By the virtue of blade protocol, the wallet will only sign the latest

message of the channel. If the channel is already closed according to central registry, this deployment will fail.

4. If the message is valid, the smart contract updates the on-chain registry of state channels and sets status of the channel to closed. Also sends proceeds of Alice and Bob according to their latest balances.
5. Alice has to send the message signed by Bob is step 2 to her wallet to unlock locked balance and nonce restrictions.

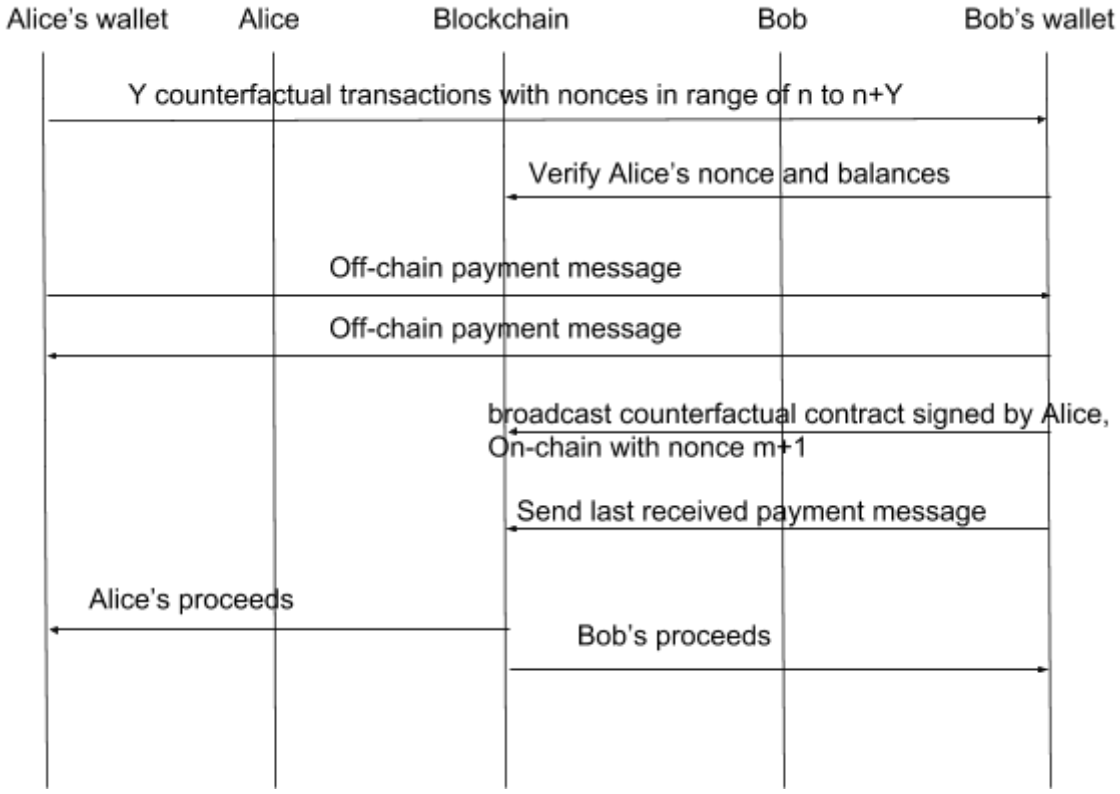

Figure 8: Case II Bob wants to close channel.

5. Security

We will first discuss various attacks and mitigation strategies and then analyse the channel protocol. If fraud proof is provided by someone to the smart contract, a reward will be given to the party proving fraud and the affected party will be compensated from wallet issuer's stake. To prevent wallet issuer from losing large amount of stake in case of an attack, the insured amount has to be limited to S/N where

Let **X** be price of wallet
Let **Y** be cost of research of reverse engineering the hardware device
Let **S** be the stake in the contract
Let **N** be the ratio limit for the stake accessible to each wallet
An attacker is then incentivized to either reverse engineer memory contents or get the issuer private key out of the device if ***S > N*(X + Y)***
Important question here being: how much does the wallet provider value their hardware security? (value of **Y**)
And then set **N** accordingly

As soon as a fraud proof is submitted to the blockchain, all the state channels will be disabled and only frauds committed before this event will be compensated.

5.1 Attacks and mitigation strategies

We will proceed to discuss how blade secures assets in the network. We will discuss various attacks and other unintended scenarios and how they can be mitigated.

5.1.1 Remote attestation

In Blade protocol, a participant must remotely attest the counterparty's hardware, balances, their wallet issuer's state to get the guarantees provided by the protocol.

1. Alice sends a random challenge to Bob
2. Bob sends the challenge to his wallet
3. Bob's wallet signs the challenge by issuer private key and private key. Bob provides this along with the address of smart contract deployed by his wallet issuer. He also provides the merkle proof proving his issuer private key is issued by his wallet issuer.
4. Alice verifies the signatures and merkle proof on chain. Also verifies the smart contract is compliant with the protocol and is registered in the blade registry
5. Alice verifies the smart contract state to make sure no frauds have been detected in the contract
6. This information is sufficient for Alice to ascertain the issuer of Bob's wallet, amount of funds staked by issuer and to know that she can recover her funds in case of any attacks.

5.1.2 Attacks and fraud proofs

1. Intentional backdoor in the wallet

It is possible that Ingrid creates a backdoor in the wallet. It can be a software backdoor or a hardware one. Using this Ingrid can potentially clone Alice's wallet with same user and issuer private key.

Mitigation:

Ingrid can then try to transfer Alice's funds to herself. However, to get the security of the Blade protocol, users are encouraged to keep their funds on the smart contracts and not on the private key accounts. If Alice follows this guidance, she will have majority of her funds in the payment channel.
Hence to steal Alice's funds, Ingrid opens a payment channel using this card with herself and try to transfer all the funds to her wallet.
We will define state of wallet as follows
State = keccak256(<userAddress>, <userTxNonce>, <userSignNonce>, <paymentChannelAddress>, <issuerKeySignNonce>, <latestEthereumBlockHash>, <IssuerKeySignature>)

Here,
userAddress = address of the user generated account on the wallet
userTxNonce = the ethereum transaction nonce as remembered by wallet

userSignNonce = the nonce of the signed message, (payment channel nonce)
paymentChannelAddress = the contract where user has deposited funds
issuerKeySignNonce = the SignNonce of the issuerKey
latestEthereumBlockHash = latest Ethereum block hash
IssuerKeySignature = the signature of the message as signed by issuerKey

If Ingrid uses the cloned card to take any action such as opening channel, sending funds, closing channels, the state of the wallet will change.
If Alice detects any such activities, She can call **proveState** method on her wallet and call corresponding method in the smart contract as well. This will nullify the actions taken by Ingrid, as her state variables will be different. This is because the wallet should not have been able to sign a state that is different than claimed state at present time.
This proves fraud and implies the wallet issuer and her cards are potentially compromised. All channels under this issuer will be frozen and must be settled on-chain. In case of loss of funds, users will be compensated by the smart contract if they are able to provide such proof.

2. Wallet issuer uses deterministic key generation

If wallet issuer uses deterministic key generation for issuer private key, or user key, she can use it to clone users wallet. Mitigation same as 1.

3. Wallet issuer denies issuance of wallet

It is wallet issuers duty to periodically upload merkle root of merkle tree containing public keys of all the wallets she has issued. When selling the wallet to users, she should include the public key and corresponding merkle root and merkle proof of that key. Users should verify onchain whether the proofs are valid before using the wallet.

4. Wallet pretends to have a particular issuer private key

A counterfeit or tampered wallet can pretend to have issuer private key issued by any wallet issuer. Users can use a challenge response game to verify if the key exists in the wallet.
walletVerify <randomNumber>
Response: signature(0x19,<randomNumberIssuedByUser>), signatureUsingTheIssuerKey

5. Wallet issuer does not stake enough

Users should check on-chain before opening channel to make sure enough collateral is provided by wallet issuer to cover ALL channels opened under her. If wallet issuer wants to withdraw stake, it will be subject to a long waiting period of several months.

6. Wallet issuer reduces stake

Once staked, wallet issuer’s stake will be locked for a minimum duration (say 2 months). Wallet issuer can reduce her stake to n if it doesn't reduce the collateral below the amount required to cover existing channels. New channels cannot be opened if it increases the collateral requirement above n.
After the 2 month duration, wallet issuer will be able to reduce her stake and withdraw remaining funds.

7. Attacker clones wallet

If the attacker clones the wallet, and is able to execute an attack the user will lose funds. The user will be able to prove fraud and recover funds in certain cases, in which case, the wallet issuer will lose some of her deposit. Hence wallet issuer needs make sure the wallets are secure and do not have any vulnerabilities.

8. User keeps her money in key and it is stolen

Users are discouraged to keep money in key. If they do and there key is stolen, however if they can prove they still have the wallet and the wallet does not have the state saved where that money is sent, they may be able to recover funds.

9. Users wallet is lost/stolen

Wallets conforming to the Blade protocol do not have functionality of backing up and restoring private keys. Hence, it is users responsibility to provide a backup address, which can be used to close channels and withdraw funds. In this case since the issuer private key attestation is not possible, users will be subject to a long waiting period (~30 days) unless the second party to the individual channels provide attestation, in which case instant settlement is possible.

10. Double spending

Double spending is not possible if counterparties verify the total locked balance of the user and their on-chain balance before opening channel with the user.

12. Mass double spend/fraud

Wallet issuer can do multiple fraudulent transaction/ double spend attacks before they are detected. This can cause a big loss of funds to multiple users. Hence it is necessary that each and every users deposit is collateralized by wallet issuer and disable all channels as soon as fraud is observed.

13. Same issuer private key in all wallets

If wallet issuer is careless and puts same issuer private key in all wallets, it can be easily detected from onchain contract. It increased the possibility of attack on all the wallets hence wallet issuer is strongly disincentivized to do this.

14. Attacker impersonates a wallet issuer

Same as 4.

15. Self destructing issuer private key

Wallet issuer may program the wallet to destroy issuer private key forever from the wallet. Or an attacker can find a vulnerability which deletes the issuer private key permanently. User can still recover her funds as the funds are stored in smart contract. Users can send recover message from a different predefined traditional wallet and withdraw funds after a long waiting period.

16. Self destructing private key.

Same as 15

17. Attacker either takes control of memory or gets the issuer private key out by reverse engineering

Let **X** be price of wallet
Let **Y** be cost of research of reverse engineering the hardware device
Let **S** be the stake in the contract
Let **N** be the ratio limit for the stake accessible to each wallet
An attacker is then incentivized to either reverse engineer memory contents or get the issuer private key out of the device if ***S/N > X + Y***

Wallet issuers need to set N according to their risk taking ability and their confidence about security of their devices.

6. Applications

In future work regarding this protocol, the following ideas may be explored:

1. Insurance for wallets and their balances and for issuers
2. Cross blockchain Atomic swap
3. Decentralized poker and other online games
4. Decentralized Exchanges: e.g. ability to cancel orders without onchain tx
5. Use of these wallets as a medium of exchange
6. Explore applications by using encrypted communication between wallets which cannot be read by wallet owners or their counterparties.