golem

Trustless Voting Protocol

Trustless Voting Protocol - Whitepaper

1.0
Date: 13.07.2020
Warsaw

Table of content

Date	Version	Who	Description
06.03.2020		Łukasz Gleń	draft
06.04.2020		Marcin Benke Jakub Konka	review
16.04.2020		Michał Kowalczyk	review
22.04.2020		Borys Popławski Rafał Wojdyła	review
13.07.2020	1.0	Łukasz Gleń	publish

Introduction

In a small group members can vote on a meeting. So everyone is a witness and can confirm fairness. But when there is a massive voting, or online voting or geographically distributed voting, we need a system that is secure and sometimes confidential, depending on the voting type. Moreover, regarding any system, we need to ask 'who I have to trust', 'what are weak points'.
Recently there appeared a new area of voting. More and more blockchain projects opt for decentralized governance. It means that a community votes on important project decisions. So the demand for trustless systems is even bigger and more current.
We propose a Trusted Execution Environment (TEE) based voting solution that eliminates trust to a voting organizer but preserves security and confidentiality.
We consider two sources of threats. The first is voters, the second is the voting organizer. Usually the voting organizer is responsible for verification whether a person is eligible for voting and votes only once. So the whole trust is moved to the central point.

We propose to replace a voting organizer with a voting enclave and analyse security aspects of the proposed solution.
But it does not solve all problems. An enclave is run on a certain machine. And an owner or an administrator of this machine - an enclave host - controls communication with an enclave in the way it can selectively drop communication. We indicate how this can be prevented but the detailed solution will be described in the next paper. Moreover, when we secure a voting then vulnerabilities are before and after. For instance, a voting organizer can incorrectly inform on a voting or publish results. We discuss it here but some of these are out of the scope of this paper.
We provide security in the following meaning: a system prevents unwanted behaviours or there is proof that one of the parties cheated.
The protocol is: trustless - it removes the need for trust in the honesty of the voting organizer; confidential - no party, including a voting organizer, knows who and how voted; secure - prevents double voting and forging extra voters; verifiable - every party is able to verify the outcome.
There is a question whether TEE can be really trusted and there are considerations on capabilities of TEE vendors to control, monitor, provide backdoors, etc. This affects whether our protocol can be actually considered trustless. That is a vast and important discussion on TEE. As we would like not to go into that here, we assume that our TEE is actually trusted and upon this assumption we consider the protocol as trustless.
Similar effect can be achieved with Ethereum blockchain based voting. But there are some differences. First is the cost. It is not a big problem, but in some scenarios should be taken into account. Everyone can read the vote of a given address. So it is not fully confidential. Everyone also knows results during voting. This is not a flaw, rather this is a feature. The same effect can be achieved here easily - a voting enclave could report partial results frequently. On the other hand, blockchain based voting is nonvolatile and a voting organizer cannot hide results.

The scope

Every voting has three phases: announcement, collecting votes and publishing results. We mainly focus on the collecting votes phase.

Unreliable announcement

The voting organizer announces the voting and the voting can start. It is crucial that every voter receives the announcement, directly or indirectly, factually or potentially, before the start date.
The malicious organizer can manipulate the voting either by delivering the announcement to selected voters only or by announcing just before the start date. It is very hard to prove that the organizer failed to announce the voting properly and honestly. This threat is out of scope of this paper. We assume that all voters are informed on the voting on time. One possible solution might involve announcing the voting on blockchain.

Denial of Access attack

Our goal is to provide a voting system that cannot be cheated or yields objective proof that one party cheated.
Denial of Access attack on the system is a situation where the voting organizer hosts the voting enclave and precludes voting by some users by simply dropping connection. It can block some IPs. Or it can give access to allied users and block random other users. This way it influences the voting results.
Preventing this kind of attacks is out of the scope of this paper. There is a solution that multiplies voting enclaves and hopefully will be presented in the next paper.

Unpublished results

We show that the results cannot be falsified. But the voting organizer can simply not publish it. In this situation only the organizer knows the results. The voters can count again the votes by resending them to another enclave. That enclave is not the same software as the voting enclave. We do not describe the details here but it is quite straightforward to design.

TEE

We assume that TEE enclaves:

- Can securely generate random numbers.
- Cannot rely on the clock.
- Can securely persist data.
- Communicate with external voters with a host owner intermediation but transferred data is encrypted so a host machine owner cannot read it.
- Are vulnerable to the replay attack.
- There is a proof that this is actually an enclave with a given code, a proof verifiable by voters. (For instance SGX attested quotes.)

The replay attack refers to stateful enclaves that persist their state. An example scenario is as follows.

- The host machine owner runs the enclave, then stops it and copies its persisted state.
- The host machine owner runs the enclave again and after a while stops it.
- The host machine owner restores previously saved state and runs the enclave the third time with the restored state.

The effect is that all changes within the second run are lost.

Community argument

Here trustless mainly means that voters do not have to trust in the voting organizer's honesty. But still some actions depend on the voting organizer. If a community, voters, recognizes that it makes tricks or abuse its position, then the community revokes it from its role. We call it the community argument. It can also be understood as the voting organizer's concern of the community reaction. It best applies to things that are visible but they are inconvenient to include in the protocol. For instance.

- Vague Voting Description.
- Too short voting duration.

- The voting organizer does not respect the voting schedule.
- The voting announcement is just before the voting starts.
- The voting announcement is sent to only selected voters.
- The voting organizer does not show the voting results or delays showing.
- The voting organizer prevents some voters from voting.

Some of those, like the voting duration, can be part of the protocol and can be verified by an enclave. We do not discuss it here, but it seems to be straightforward and would make the protocol unnecessarily complex. Some of those, like the voting announcement, can be enhanced by blockchain. It is described below. Some of those, like denial of access, can be solved by using multiple voting enclaves. It is described below. Although using multiple voting enclaves is beyond the scope of this paper and it is expected to be the next version of the protocol.
Of course we prefer provable solutions over community arguments, as the latter are not strict. So we want to push the protocol boundaries so the community arguments are unnecessary.

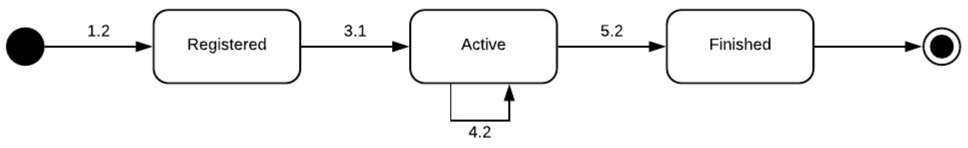
Scenario

Our reference scenario, that you should think of, is blockchain based voting. This is typical for this industry and refers to governance. Governance in this case means that a community is able to govern or manage projects, tokens, smartcontracts, global parameters, etc, through voting.
Every voter has an Ethereum or other blockchain account. The address of the account is a voter's identity. The voter authenticates by signing transactions or some other data with its private key. This is the advantage of blockchain, open authentication as an intrinsic part of the system. The votes can be weighted with the number of tokens held or a similar measure. One possible approach is to take into account holdings of a given token at a specific block number. Another one requires voters to pay time locked stakes.
Usually such voting takes place on blockchain, on-chain, as the only trusted but distributed component. We propose to replace it with TEE. Thus such voting is off-chain.
The blockchain solves two problems by design: verification whether a given user is eligible to vote and authentication. Our solution can be adopted to other scenarios as long as those two points are addressed.

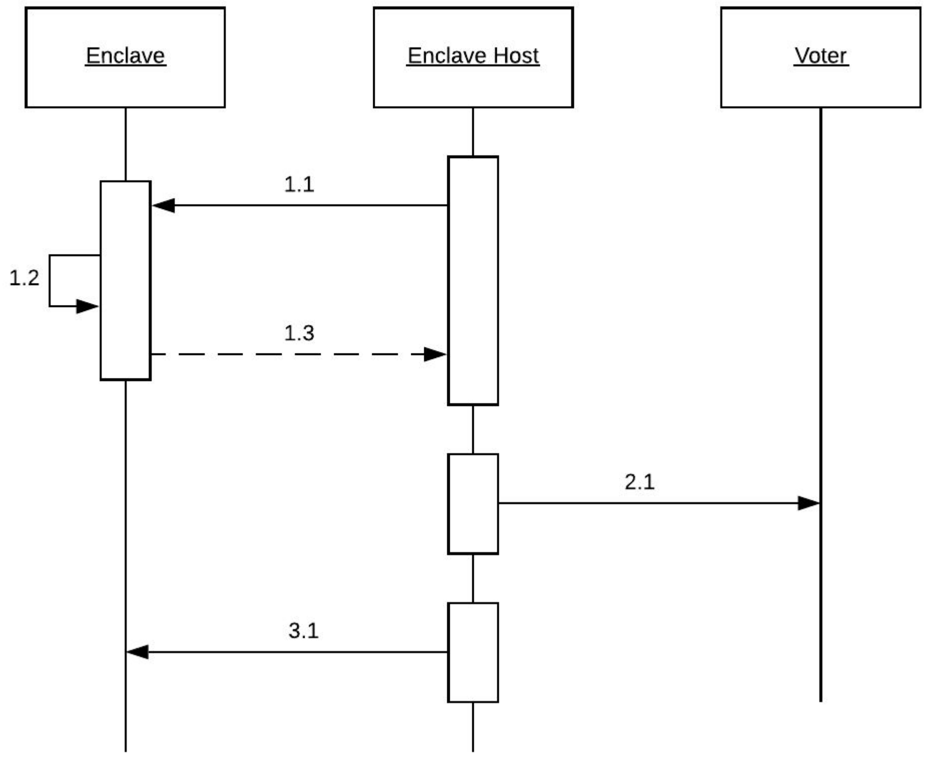
The protocol

There are three actors.

- The Voting Enclave is a TEE based enclave, responsible for counting votes.
- The Enclave Host runs the Voting Enclave and is responsible for organizing the voting. It represents both the machine and its owner.
- The Voter. A person or a machine. It votes and verifies the voting results.

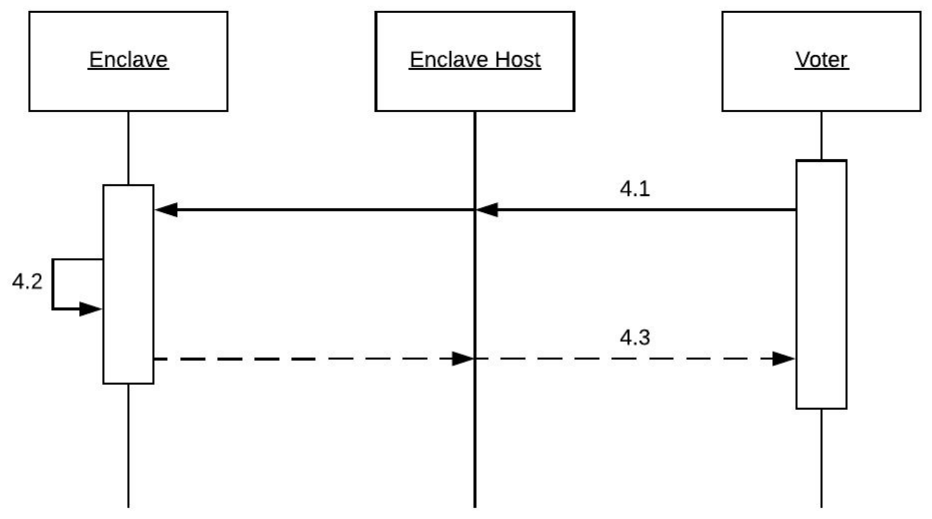


The states of a voting in the Voting Enclave.
All state transitions are triggered by the Enclave Host.
In the states Registered and Finished the voting is disabled, it is only enabled in the Active state. Finished is the final state.
Registered means that voting is registered in the Voting Enclave - the Voting Description is sent to the Voting Enclave.
Active means that the Voting Enclave is collecting votes.
Finished means that the Voting Enclave stopped collecting votes and provides the Voting Result.
The transitions Registered->Active and Active->Finished should be triggered by the Enclave Host at the time included in the Voting Description.

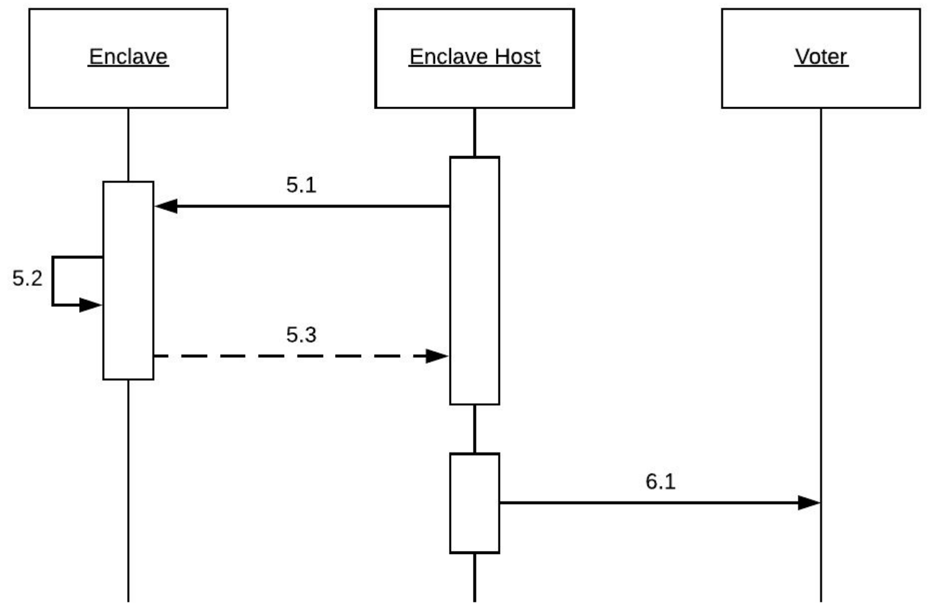


The Voting Initialization.
1.1 The Enclave Host sends a VD (Voting Description) message to the Voting Enclave.
The VD structure:
{a description, the number of options, start date of the voting, end date of the voting, a list of eligible voters with their vote weights}.
A description is a text that describes the voting and presents options. It is not interpreted by the Voting Enclave. Options are numbered, sequentially. The number of options instructs the Voting Enclave what are possible options. A list of eligible voters with their vote weights is a list of pairs, a voter and its vote weight. For instance, it is an Ethereum address and the balance of GNT at the given block number; or it is an Ethereum address and 1 as a weight.
Remark. Of course the Voting Description message could have different structure. Just the Enclave Host must understand it. The approach presented here is very simple. The description is any human readable text containing: what is this voting about, what are conditions, what are options. The options are numbered. So the vote is just the number of the option. The Voting Enclave needs to know whether the vote is valid. Actually the Voting Enclave does not need to interpret the vote, the number is enough. For that reason VD contains the number of options. This instructs the Voting Enclave how many options are

eligible. Having all options numbered sequentially, the Voting Enclave knows all options.
1.2 The Voting Enclave registers the voting according to the VD.
The Voting Enclave needs to save the list of eligible voters with their vote weights, and hash(VD, nonce1).
Nonce1 is a random number drawn by the Voting Enclave.
1.3 The Voting Enclave responds with a VDVE (Voting Description signed by Voting Enclave) message to the Enclave Host.
The VDVE message structure:
{VD, nonce1, sig VE}.
Sig VE is the signature of the hash(VD, nonce1) with the Voting Enclave private key.
2.1 The Enclave Host sends a VDEH (Voting Description signed by Enclave Host) message to the voters. The voters can also request this message at any time. The Enclave Host must deliver this message to the voters before the start date included in the VD message. But the voters can request it at any time.
The VDEH message structure:
{VDVE, sig EH, the quote, the public key of the Voting Enclave}.
Sig EH is the signature of the hash(VDVE) with the Enclave Host's private key. Note that it is enough to sign hash(VDVE) only. The quote is a proof that the Voting Enclave actually is a TEE enclave - for instance the quote could be an attested SGX quote.
By signing the Voting Description, the Enclave Host takes responsibility for its content. In particular it is responsible for the correct list of voters and is blamed in case of a Replay Attack detection.
3.1 At the start date included in the VD message the Enclave Host activates the voting in the Voting Enclave by sending a start message. The voters can send votes now.



4.1 The Voter sends a VV (Voter's Vote) message to the Voting Enclave.
The VV message structure:
{the voter's address, hash(VD, nonce1), the option, sig V, the voter's public key}.
The voter's address is the Voter's Ethereum address. Note that the Ethereum address is a derivative of the public key so the latter is also required. The option is the Voter's choice in the voting. Sig V is the signature of the hash(the voter's address, hash(VD, nonce1), the option) with the Voter's private key.
The Voter communicates with the Voting Enclave through a secured channel, so the Enclave Host cannot read the messages. But still the Enclave Host can drop connection.
4.2 The Voting Enclave registers the vote.
The Voting Enclave creates the message RV (Registered Voted). At the moment it is only for the registration purpose.
The RV message structure:
{the voter's address, hash(VD, nonce1), the option, nonce2}.
The nonce2 is a random number drawn by the Voting Enclave for each vote.
The Voting Enclave verifies if the Voter is eligible to vote and registers its choice according to its vote weight. It persists the whole RV message in order to remember who already voted and nonce2.
4.3 The Voting Enclave responds with the VVR (Voter's Vote Response) message to the Voter.
The VVR message structure:
{RV, sig VE(hash(RV), hash(VD, nonce1))}.
It is important that the Voting Enclave signs both hash(RV) and hash(VD,nonce1).
This assigns a hash of vote to the voting.
The Voter should keep VVR.



5.1 The Enclave Host stops the voting at the end date included in VD by sending an end message to the Voting Enclave.
5.2 The Voting Enclave stops the voting.
5.3 The Voting Enclave responds with a VRVE (Voting Results signed by Voting Enclave) message to the Enclave Host.
The VRVE message structure:
{hash(VD, nonce1), a voting result, a list of hash(RV), sig VE}.
A voting result is a list of options and collected weighted votes. A list of hash(RV) is a list of hash(RV) of all votes.
Sig VE is the signature of the hash(hash(VD, nonce1), a voting result, a list of hash(RV)) with the Voting Enclave's private key.
6.1 The Enclave Host publishes a VREH (Voting Results signed by Enclave Host) message. It should be sent to every voter and a voter can request it at any time.
The VREH message structure:
{VRVE, sig EH}.
Sig EH is the signature of the hash(VRVE) with the Enclave Host's private key.

The keys

- The Voting Enclave keys (priv/pub) are generated by the enclave once and then are sealed. The hash of the public key is included in the quote. And along with the quote is sent to voters in VDEH message.
- The Enclave Host public key refers to the Enclave Host's Ethereum address. It is understood as identity and signatures authenticate the Enclave Host.

- The Voting Enclave public key refers to the Voting Enclave's Ethereum address. It is understood as identity and signatures authenticates the Voting Enclave.

Analysis

The Voter

Verification of voters.

The voter is identified with its Ethereum address. A user that holds a private key is eligible for voting.
The message VV includes the signature of the voter, sig V. So the sender has to be the voter. And no one intermediating in communication cannot change the vote without invalidating the signature. Even if it would be sent plain text. So when the vote reaches the Voting Enclave, it is sure that the vote was submitted by the voter only.
The Voting Enclave verifies if the voter's address is in the voters' list included in VD. So only eligible voters can vote.
The Voting Enclave internally saves the voter's address and its option. So the voter cannot vote twice under any condition.

Verification of results.

How can the voter verify if its vote was counted in the final results?
Having the RV message and the VREH message, the voter can verify whether its vote was counted in the final results. Simply by checking if its hash(RV) is in the list of hash(RV) in the VRVE message, which is a part of the VREH message. Since it is signed by the Voting Enclave, the voter is sure that it comes from the enclave. If the Enclave Host is honest, then the voter's hash(RV) must be in the list. It is because the Voting Enclave saves the vote upon receiving the VV message but before responding with the VVR message. There is no chance to lose the vote by the Voting Enclave even if it was restarted during processing the vote. So if the Enclave Host is honest, then the voter's hash(RV) must be in the list. And if it is not, then this is an objective proof that the Enclave Host cheated.
Note, the Enclave Host can manipulate the results by Reply Attack, described below.

False claim.

A malicious voter can attempt to break the voting with a false claim that its vote was not counted in the final results.
The voter needs to deliver proof that consists of: the VREH message, the hash(RV), sig VE(hash(RV), hash(VD, nonce1)). The sig VE(hash(RV), hash(VD, nonce1)) is a part of the VVR message. It is important that the Voting Enclave signs both hash(RV) and hash(VD, nonce1). So the hash(RV) has to be signed by the Voting Enclave. But the voter cannot deliver data from another voting since it is signed along with the hash(VD, nonce1).

Verification of votes.

The voter can simply verify if its vote was counted - by resending the same vote again. It gets back the same VVR message in a response as it was at the first time.

When the Voting Enclave receives the vote at the first time, it saves the voter's address, the chosen option and nonce2. It is enough for the Voting Enclave to recreate the RV/VVR message and send it back upon the second request. Note, that a voter cannot change its choice - it gets its first choice in a response when it votes again.
In particular, if the vote was counted but the Voter did not get the VVR message due to communication problems, it can resend the same vote in order to get the same VVR. If the problem repeats, because the Enclave Host somehow managed to drop communication for instance, then this is a case of Denial of Access attack described elsewhere.

The Enclave Host

Parallel votings.
The value of hash(VD, nonce1) is actually the voting identifier.
When the Enclave Host runs twice the voting, then they actually get different hash(VD, nonce1) and those are different votings.
Voters' list.
If the Enclave Host modifies the voters' list included in the VD, then hash(VD, nonce1) is also changed. And this is a different voting. So the Enclave Host cannot send one voters' list to the Voting Enclave and another to the voters.
The Voting Enclave is not capable of verifying whether the voters' list is proper and weights are correct. It just cannot connect to the blockchain directly. The list is handed to the Voting Enclave by the Enclave Host within VD. But the message VDEH includes the signature. It confirms that the Enclave Host takes responsibility on what was handed to the Voting Enclave. The voters can verify if the voters' list is correct because the blockchain is public. If it is not, then this is objective proof that the Enclave Host cheated.
Corrupted enclave.
The voters have never direct access to the Voting Enclave. One may think of using an enclave emulation or corrupted enclave software. So the Enclave Host - the voting organizer - can change the voting results without being detected. So, it is required for the voters to be able to verify that a real TEE is used and authorized software that is used to count votes.
In the case of SGX it is given by design. The attested quote proves that it comes from a real SGX enclave. The hash of the initial state of the enclave is included in the quote. It can be considered as identification of the software being used. So the Enclave Host cannot replace it with its own software.
Moreover, the Voting Enclave puts the public key into the quote and does not reveal the private key. When the voter encrypts a message with the public key, only the Voting Enclave can decrypt it. So the Enclave Host cannot replace the Voting Enclave with its own software and redirect communication to this software.
Replay attack.
The Enclave Host can publish results delivered by the Voting Enclave only, since the VRVE message contains the signature of the Voting Enclave. It also cannot modify the voters' list without being detected as described above.

The Enclave Host can try to execute a replay attack, then some votes can be not counted. In order to prevent this, voters verify whether their votes were counted as follows.
The VREH message includes the signature of the Enclave Host so it takes responsibility for the results. The VREH message includes the VRVE message signed by the Voting Enclave. And the VRVE message includes the list of hash(RV). A voter verifies whether its hash(RV) is included in the list. If it is not, then it is objective proof that the Enclave Host executed a replay attack. An omitted voter can verify only itself so in order to ensure that the voting was honest, every voter has to do this verification.
<u>Unpublished voting results.</u>
The Enclave Host is obligated to present the voting results on demand. There is no technical way to prove that it failed to deliver it. But not publishing is clearly visible for the community. Moreover, in case the Enclave Host constantly refuses to deliver the voting results, the votes can be recounted by resending votes to a special enclave. No new votes can be added in this case.
<u>Voting out of schedule.</u>
As was said, the Voting Enclave cannot rely on system time. The actual time of starting and ending voting is controlled by the Enclave Host. So the enclave itself cannot generate reliable proof whether actual start and end dates complies with VD. The obvious attack is that the Enclave Host starts voting earlier, its allied voters vote and it ends the voting before the schedule. So only its allied voters are able to vote.
We take the voter's point of view. So the voter tries to vote according to the schedule but the Voting Enclave responds with 'voting inactive'. The enclave can append the timestamp to the message. If it is valid then this is objective proof that the Enclave Host does not act on schedule. But as it was said, the Enclave Host can set a false time on the machine and the enclave cannot detect it. So we cannot rely on timestamps.
Moreover. If we even manage to force the Enclave Host to start and end the voting on the schedule, then the Enclave Host can still do the Denial of Access attack as described below. Putting it straight, having prevention against Denial of Access attack, we also solve Voting out of the schedule problem.
Nevertheless, we consider some countermeasures.

- There is the community argument of course. If there are enough complaints, the community can decide to invalidate the voting. But this is not objective.
- Having multiple Voting Enclaves, the voting starts or ends when all or majority of enclaves agree. This is similar to the approach mentioned in the Denial of Access section. And it is also above the scope of this paper.
- As it was said, the countermeasures against the Denial of Access attack applies here.
- The open question is: how an enclave or a ring of enclaves can determine a reliable current time. Having this, we could use reliable timestamps in the Voting Enclave's messages.

- There is a very promising solution. Upon the voting starts, the Enclave Host needs to provide data signed by it. The idea is that the data would need to be something impossible to know in advance, like a hash of the most recently mined ETH block. The same when the voting ends. These data is included in the voting results. This is proof that the Enclave Host did not start and end the voting before the schedule. This bounds the start date and end date from below. In order to bound these dates from above, the Voting Enclave submits signed data upon starting and ending the voting. The Enclave Host is responsible for putting these data on blockchain immediately. So it is clearly visible when it happens.

Denial of Access attack

The Enclave Host cannot modify or add votes, nor cannot change results. It cannot even know who and how votes. But it can drop connection, so some voters would be unable to vote. For instance it can block given IP.
There is no way to prove that a given voter could not vote. At this level we accept this threat and recall community trust's argument.
But there is a way to improve accessibility. There can be more than one eligible Voting Enclave and when the voting is finished enclaves can exchange the results in order to deliver one combined version. We do not expand this idea here, that will be described in another paper.

Confidentiality

The only messages that contain explicit votes are VV, RV, VVR. The vote is transferred between the Voter and the Voting Enclave through a secured channel. So only those parties know it.
The message VRVE contains hash(RV). But the hash does not reveal the voter's address nor the chosen option. Thanks to nonce2.
In order to prove that the voting results are falsified, it is enough to show VREH, hash(RV) and sigVE(hash(RV)). None of these reveals the voter's address or the chosen option.
The system is confidential: only voters know their votes, the Enclave Host knows nothing. It cannot even conclude, which voters did vote and which did not.

Comments

- The enclave cannot rely on time provided by the machine. So it cannot rely on start and end dates of voting. It means that the enclave cannot confirm that the voting has started and ended at given dates.
- Sometimes it is possible to conclude the votes if weights have unique patterns. For instance, the weights - holds - could be 100001, 100010, 100100, 101000, 110000. This is not possible if all weights are equals 1.
- The Enclave Host can capture the Voting Enclave state after each vote submission along with the IP address of the voter. Then, the Enclave Host can roll back the Voting Enclave to a given state/snapshot, put it to 'Finished' and fetch results. Following increments in results, it can assign a

vote to a voter's IP address. This breaks confidentiality. It seems it is impossible to prevent this with a single enclave. It can be done with more than one eligible Voting Enclave as it was mentioned in the Denial of Access section. With such a solution, a single Enclave Host handles only a part of votes. Even better approach is joint start and end of the voting by all/majority of Voting Enclaves. This way the voting can be finished only once. But the details will be described in another paper.

- The threat described above is even more serious. It can be combined with Denial of Access attack. The scenario goes as follows. The Voter establishes a connection and sends a vote. Then the Enclave Host freezes the connection. Of course it cannot read the vote directly. But it can perform action as described in the previous point and that way it knows the vote. Then it decides whether to let the vote go or to reject the vote. In order to reject, it rolls back the enclave state and drops the frozen connection so the Voter does not get the response. So this is the advanced version of Denial of Access attack. In particular it is dangerous if the votes are balanced and a small number of votes can change the results. As previously, multiple Voting Enclaves can be a prevention.

Special thanks to Michał Kowalczyk for his important suggestions.