A large-scale anonymous electronic voting scheme based on zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) offers unique value and significance in ensuring the anonymity, security, and reliability

of voting. zk-SNARKs is a cryptographic technology that enables one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing any information other than the truth of the statement itself. Applying zk-SNARKs to electronic voting brings several key advantages:

1. Protection of Voter Identity Privacy

: The connection between a voter's choice and their personal identity is not disclosed; it's impossible to identify from public information whether a specific voter has participated. This also enhances the fairness of the voting system.

1. Prevention of Vote Buying and Selling

: Voters participate under pseudonyms and are unable to prove their own voting results. This means voters cannot sell their votes to third parties who may wish to purchase them.

1. Verifiable Voting Results

: Everyone can verify the voting results based on the public counting proofs.

Convex Analysis

The governance of Convex relies on the vote-locking mechanism of CVX tokens, the native token of Convex. By locking CVX, one can vote on any proposal presented on Convex (published on snapshot). This proposal regarding Curve aims to add a gauge to the crvUSD/USD+ StableSwap-ng liquidity pool on the Arbitrum chain to determine the amount of CRV tokens the pool should receive as liquidity rewards.

[

1264×1049 85 KB

](https://ethresear.ch/uploads/default/original/2X/8/839a3ffa37d596eca6b3df8cb7fac9106948ab91.png)

In this case, to secure more liquidity rewards for their pool, project operators need to garner more votes for this proposal. There are two ways to achieve this:

1. The project operators buy CVX and lock these CVX for voting;
2. The project operators seek other CVX holders to vote for their pool.

The first method involves the project operators buying CVX, while the second method involves buying the voting rights represented by CVX, meaning the project operators need to bribe CVX holders to vote for them. In reality, the first method is more costly than the second, so project operators tend to prefer purchasing voting rights of CVX.

However, the second method may allow project operators with more resources to gain an unfair advantage, making it difficult for smaller or newer projects to compete.

To address this issue, we employ a large-scale anonymous electronic voting scheme based on zk-SNARKs to assist Convex in private governance. This scheme involves five different participants and is defined by eight related functions. The entire voting process can be divided into five stages.

Participants Composition

The electronic voting scheme involves the following five participants: Proposers, CVX Holders IDs={id_1,id_2, ...id_m}

, Cryptographic Coordinator, Pseudonym Registrar, and Counters Ts={T_1,T_2, ...T_m}

. Among them, proposers and CVX holders are inherent to Convex. Proposal-related information and voting results are still published on snapshot.

- Proposers: Also known as project operators, they can initiate proposals for their liquidity pools and are also CVX holders themselves.
- CVX Holders: Also known as voters, they can lock CVX tokens to vote. They vote under pseudonyms to ensure anonymity. As shown in the figure below, the voting results published on snapshot will not display the on-chain addresses of CVX holders, but their pseudonyms instead.
- Snapshot: Publishes proposal information, voting information, cryptographic public parameters, voting results, and proofs of correct counting. The voting results include not only those cast by CVX holders but also null ballots cast by

snapshot.

- Cryptographic Coordinator: Responsible for generating cryptographic parameters and key pairs, and publishing the public parameters on snapshot.
- Pseudonym Registrar: Registers pseudonyms for CVX holders after they lock their CVX tokens and signs these pseudonyms.
- Counters: Responsible for calculating and verifying the voting results, and publishing the results and proofs of correct counting on snapshot

, as shown in the figure below. Counters use partial decryption keys from a k-out-of-n encryption scheme.

[

1255×1035 92.7 KB

](https://ethresear.ch/uploads/default/original/2X/a/ac79e901b656821b72cba5b80458cd54849c20e5.png)

Function Definitions

The voting scheme is defined by eight functions, namely: VS=(Setup,PseudonymRegister, Register, Vote, ValidVote, Append, Tally, VerifyTally).

1. Setup

$$\text{Setup}(\lambda, R) \rightarrow (PP, sk_{T}, sk_{\sigma}): \text{On input of the security parameter } \lambda \text{ and the relation } R \ \text{represented as an arithmetic circuit, generate the prover and verifier key pairs }\ (pk, vk) \leftarrow \text{KeyGen}(\lambda, R), \text{ voting key pair } (pk_{T}, sk_{T}) \leftarrow \text{KeyGenE}(\lambda), \text{ registrar key pair} \ (pk_{R}, sk_{R}) \leftarrow \text{KeyGenS}(\lambda), \text{ commitment parameters from commitment setup} \ CR \times T \leftarrow \text{Setup}C(\lambda), \text{ and public parameters } PP = (G, q, g, H, pk_T, pk_R, (pk, vk)).$$

1. PseudonymRegister

$$\text{PseudonymRegister}(\text{id}) \rightarrow \left(cr_{\text{id}}, c_{\text{id}}, t_{\text{id}} \right): \text{ On implicit input } PP \text{ and CVX holder}\ \text{identity id}, \text{randomly select a pseudonym } cr_{\text{id}} \leftarrow CR, \text{ compute } \left(t_{\text{id}}, c_{\text{id}} \right) \leftarrow \ \text{Commit}(PP, cr_{\text{id}}), \text{ and return } \left(cr_{\text{id}}, c_{\text{id}}, t_{\text{id}} \right), \text{ where } t_{\text{id}} \text{ is randomly selected from } T.$$

1. Register

$$\text{Register}(id, c_{id},\{L\}) \rightarrow (L, MR_{L}, S_{R}): \text{On input of the CVX holder identity and}\ \text{commitment} (id, c_{id}), \text{ and list } L, \text{ add } (id, c_{id}) \text{ to list } L, \text{ compute } MR_{L}, \text{ sign } (L, MR_{L})\ \text{ with the registrar} \text{private key} sk_{R} \text{ to produce the signature } S_{R}, \text{ and then return } \(L, MR_{L}, S_{R}).$$

1. Vote

$$\text{Vote}(id, sk_{id}, pk_{r}, v) \rightarrow \beta: \text{On input of CVX holder identity } id, \text{ voting public key } pk_{r}, \ \text{ and CVX holder private key } sk_{id} = (t_{id}, cr_{id}), \text{ it generates a ballot } \beta = (e_v, cr_{id},\{\pi_{id}\}),\ \text{ where } e_v = \text{enc}\{pk\{v\}\}(v;r). \text{ Additionally, compute a disjoint proof } \text{Prove}(pk, x, \omega) \rightarrow \pi, \ \text{ where } \omega = (r, c_{id},v, t_{id},)x = (e_v, cr_{id}, MR_{L}), \text{ and simulate a null ballot proof.}$$

1. ValidVote

$$\text{ValidVote}(\beta) \rightarrow 0/1: \text{On input of a ballot } \beta = \left(e_v, cr_{id},\{\pi_{id}\} \right), \text{ check if it is valid, i.e.,}\ \text{whether this proof is correct and well-formed, by completing the verification through}\ \text{ executing } \text{Verify}\left(vk, \left(e_v, cr_{id} \right), \pi_{id} \right) \rightarrow 0/1.$$

1. Append

$$\text{Append}(\text{snapshot}, \beta) \rightarrow \text{snapshot}: \text{On input of a ballot } \beta, \text{according to } D_t, \text{ append }\ \beta \text{ to snapshot. It generates and appends one or more null ballots } \left(e_0, c_{id}, \pi_{id}\right) \text{ according}\ \text{to the probability distribution } D_r \text{ and } D_t. \text{ It computes } e_0 = \text{enc}\{pk\{r\}\}(0; r) \text{ and disjoint} \ \text{proof } \pi_{id}.$$

$$\text{Prove}(pk , x, \omega) \rightarrow \pi_{id}, \text{where } \omega = (r, 0), x = \left(e_0, cr_{id}, MR_{L}\right), \text{ and simulates the other }\ \text{ side} \text{ of CVX holder proof.}$$

1. Tally

\text{Tally}(\text{snapshot}, sk_T) \rightarrow (s, \Pi): \text{ On input of the public snapshot, calculate the }\ text{voting results, return } (s, \Pi), \text{ where } s \text{ is the voting result, } \Pi \text{ is the proof of correct}\ \text{ counting, with the following steps:}

- Run ValidVote(\beta)

and return 0 in case of failure.

- For each cr_id

appearing in the ballots, calculate

,where

is the set of (e_v, cr_{id},\pi_{id})

identified by cr_{id}

.

- Remove (cri,\pi_i)

from each B_{cr_i}

, mix the ballots {{B_{cr_1}, B_{cr_2},..., B_{cr_k}}

}, where k is the number of pseudonyms cr_i

, and return the mixed ballots

and proof of valid mixing.

- For each B_i^{'}

and voting option v \in V

, apply the privacy equivalence test (PET) and provide corresponding proof.

- Calculate the result s for each voting v based on the PET result and publish the proof.
- VerifyTally

\text{VerifyTally}(\text{snapshot}, s, \Pi) \rightarrow 0/1: \text{ On input of } (s, \Pi), \text{ if all proofs are valid, return } 1;\ \text{ otherwise, return } 0.

Implementation Phase

The flowchart of this electronic voting scheme illustrates the process of Convex’s private governance as follows.

[

1251×1011 79.7 KB

](https://ethresear.ch/uploads/default/original/2X/5/54ee0524f5613bf4e96a923d3fcc3f8e0533f29a.png)

Based on this flowchart, we divide the Convex private governance process into five stages:

1. Setup Phase

Given security parameters and relation R, the Cryptographic Coordinator runs Setup,R to:

- Generate cryptographic parameters (G,q, g)

, counting party threshold tuple (k, n)

, voting key pair (pk_T,sk_T)

, registrar key pair (pk_R,sk_R)

, commitment function and its parameters H:CR * T\rightarrow C

, and the zero-knowledge proof key pair pk,vk for relation R.

- Publish public parameters PP=(G,q, g, H, pk_T, pk_R, (pk, vk))

.

1. Registration Phase

CVX holders (id) run Register(id) to:

- Choose a voting pseudonym cr_{id} \in CR

.

- Compute c_{id} = H(cr_{id}, t_{id}) \in {0,1}^{（0，\lambda）}

using t_{id} \in T

and store (cr_{id}, t_{id})

locally.

- The pseudonym registrar adds (id, c_{id})

to the pseudonym list L

- Compute the merkle tree root MR_L

based on the commitment order in list L

.

- Finally, sign L

and MR_L

and publish them on snapshot.

- CVX holders verify c_{id} \in L

and merkle tree root MR_L

.

1. Voting Phase

To cast a vote v, CVX holders run Vote(id,sk_{id},pk_T,v)

, where sk_{id}=(t_{id}, cr_{id})

, including:

- Compute e_v = \text{enc}_{pk_r}(v; r)

, where r \in \mathbb{Z}_q

is a random value for encryption. In the case of revoking a previous vote v_{\text{pre}}

and voting for v_{\text{new}}

, CVX holders set v = v_{\text{new}} - v_{\text{pre}}

.

- Calculate zero-knowledge proof \pi_{id}

using proving key pk

:

[

image

1246×88 9.17 KB

](https://ethresear.ch/uploads/default/original/2X/2/2cf5ce74c1f1ba0496f801cba254c9a49c679274.png)

- Submit \beta = (e_v, cr_{id}, \pi_{id})

to snapshot via an anonymous channel.

- Snapshot runs \text{ValidVote}(\beta)

, checks the validity of the proof on the ballot, and verifies if the ballot already exists on snapshot.

- Snapshot runs \text{Append}(\text{snapshot}, \beta)

, appending the ballot \beta

and null ballots to snapshot.

- CVX holders verify if \beta

is appended to snapshot.

- Snapshot generates null ballots as follows: Calculate e_0 = \text{enc}_{pk_r}(0; r)

, choose a cr_{id}

from \beta

on the snapshot, compute \pi_{id}

:

[

image

1246×80 9.36 KB

](https://ethresear.ch/uploads/default/original/2X/b/b4a16d2041cac84d93a17a598f78a4ba9f3bfae7.png)

1. Counting Phase

Counters run Tally(snapshot,sk_T)

, including:

- Verify ballots on snapshot and select those with valid proofs.
- Apply the homomorphic properties of the encryption scheme to calculate the final ballot for each cr_{id}

on the snapshot.

- Shuffle, mix, and publish the final ballots without crid, providing proof of correctness.
- Apply PET to the final ballots and select encrypted votes from the voting.
- Decrypt ballots and publish results and decryption proofs on snapshot.
- Verification Phase

Anyone can verify the correctness of the counting by running VerifyTally(snapshot,s,\Pi)

, which verifies the results and all proofs published during the counting process. This implementation achieves large-scale electronic voting while ensuring anonymity, security, and reliability.

Compression

Recursive zk-SNARKs allow a prover to put “more knowledge” into a proof while ensuring that these proofs can still be verified by a verifier in constant or poly-logarithmic time. Using recursive zk-SNARKs as an “aggregator” of information enables independent “aggregation” of more computations than the largest (non-recursive) circuit can handle.

More about recursive zk-SNARKs.

Future Work

Performance and Scalability:

As the number of voters increases, the system’s performance and scalability will face challenges. Future research could focus on the study of fully recursive zk-SNARKs to support larger-scale elections.

User Friendliness:

An important task in promoting ZK technology is to address the current complexity of the system for ordinary users. Simplifying user interfaces and operational processes, and lowering the barrier for user adoption, will contribute to the widespread acceptance and adoption of the system.

Compliance and Legal Challenges:

Decentralization and compliance have always been antonyms; typically, only the government can prove your identity. However, this leads to the possibility of data being tampered with at the source (but if this identity is recognized by the government, then even the fake becomes real). ZK cannot solve the problem of a trusted source, but we might need to consider: what kind of world it would be if I could prove I am who I am.

If you are interested in integrating ZK technology into your project governance to enhance privacy, scalability, and achieve governance innovation

, Salus offers end-to-end ZK services and comprehensive solutions. By collaborating with us, you can explore the extensive applications of ZK technology in project governance, providing the community with a more secure, transparent, and efficient electronic voting system.