Privacy Preserving Opinion Aggregation

Aggelos Kiayias
aggelos.kiayias@ed.ac.uk
University of Edinburgh
Edinburgh United Kingdom
IOG
Global

Vanessa Teague
vanessa.teague@anu.edu.au
Australian National University
Canberra Australia
Thinking Cybersecurity Pty. Ltd.
Wurundjeri land Australia

Orfeas Stefanos Thyfronitis Litos
orfeas.thyfronitis@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt Germany

ABSTRACT

There are numerous settings in which people's preferences are aggregated outside of formal elections, and where privacy and verification are important but the stringent authentication and coercion-resistant properties of government elections do not apply, a prime example being social media platforms. These systems are often iterative and have no trusted authority, in contrast to the centrally organised, single-shot elections on which most of the literature is focused. Moreover, they require a continuous flow of aggregation to take place and become available even as input is still collected from the participants which is in contrast to "fairness" in classical elections where partial results should never be revealed.

In this work, we explore opinion aggregation in a decentralised, iterative setting by proposing a novel protocol in which randomly-chosen participants take turns to act in an incentive-driven manner as decryption authorities. Our construction provides public verifiability, robust vote privacy and liveness guarantees, while striving to minimise the resources each participant needs to contribute.

1 INTRODUCTION

Motivated by the need for decentralised crowdsourced content curation, in this work we design and analyse a protocol that enables a group of participants to aggregate their opinions in a privacy-preserving and timely manner. Each party may vote once, choosing among up/down/abstain, represented by +1, -1 and 0 respectively. Contrary to existing e-voting protocols, our construction is not *single-shot*, i.e. it does not forbid revealing the results until after all votes have been cast, but instead reveals votes dynamically, while multiple instances of it can be executed in tandem allowing a continuous flow of opinions to be aggregated on various topics.

This makes our scheme suitable for aggregating user preferences on social media content on platforms such as Reddit [8], Steemit [58], Slido [57] and BitClout [59] in an online fashion. Such settings typically allow users to post, view and rate content continuously. User-generated ratings are used to sort content, therefore ratings have to be constantly updated.

What our system offers compared to existing content rating systems is the ability to perform opinion aggregation in a privacy-preserving fashion without relying on any centralized or even distributed authority to be responsible for processing and revealing the results. We stress that the requirements of (i) continuous flow of inputs aggregation and result generation, (ii) the lack of any authority for processing, and (iii) the requirement for decentralized operation and a large participant population, set aside our setting to traditional e-voting protocols which are not fit for purpose.

At a high level, our construction is as follows. Votes are decrypted in *batches* of (predetermined) size B. Each party can choose to vote at any time. Voting is done in a number of steps: first an n-sized subset of the parties, called *decryptors*, is chosen in a *verifiably* random fashion out of a large set of parties. Then the voter secret-shares its vote using t-out-of-n Shamir's Secret Sharing [51], creating one share per decryptor. Next, the voter encrypts each share with the public key of the corresponding decryptor, using an encryption scheme that is homomorphic for at least B additions (e.g. [47], or the exponential form of [21]) where B determines the anonymity set size. Once done, the voter publishes the encrypted shares on a Bulletin Board. An honest party also periodically checks whether it is the decryptor of a batch of votes. A batch is formed when B votes contain at least t decryptors in common. If there is more than one available batch, ties are broken in a deterministic manner. If a party concludes that it is a decryptor of a batch, it sums its B share ciphertexts, decrypts and publishes the resulting aggregate share. If at least another $t-1$ decryptors of the batch do the same, then anyone can reconstruct the aggregate of said B votes in a publicly verifiable manner. As discussed later in more detail, the parameters t, n and B can be tuned to achieve different tradeoffs.

We note that the participants will operate entirely asynchronously coordinating only via the Bulletin board which in practice can be implemented by a blockchain system. Moreover, participation can be incentivized assuming the underlying Bulletin board supports a cryptocurrency. Under this assumption, we design a mechanism that incentivizes utility maximizing participants to be online and engage with decryption as required.

Related Work. The field of content curation studies how opinions can be combined into an aggregated outcome, generally with expressive preferences, ongoing interaction, and little focus on privacy or the risk of manipulation [1, 3, 4, 17, 19, 23, 26, 32, 36, 39, 43, 45, 48, 53, 56, 58, 60, 62, 64]. Content curation consists of algorithms and protocols that ensure users of a system, commonly a social media platform, access the most relevant and useful content. The constant inflow of content in such platforms precludes manual classification and ranking, and crowdsourcing the procedure by leveraging users' judgement can vastly improve the quality of curation. Therefore our voting protocol, which is suitable for aggregating parties' opinions in an online fashion (as opposed to single-shot protocols), stands at the intersection of voting and content curation.

This work was partly supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE..

E-voting refers to techniques and methodologies of aggregating the votes of a group of voters for candidates or proposals and publishing this aggregate in a timely, privacy-preserving, incoercible, yet accountable manner (or at least with some combination of these properties). Academic schemes generally provide some form of verifiability [14]: *public verifiability* means, informally, that anyone can check that all included votes have been properly tallied, while *individual verifiability* means that each voter can check whether their vote has been properly included.

Internet voting schemes for public elections typically have a designated set of election administrators who are, at least, trusted for privacy [16, 20, 22]. These schemes scale to a very large number of voters and a modest number of authorities, often with some redundancy in case some of them fail or are corrupted. In our setting, no such distinguished set of parties exists, something that could be seen also as a strengthening of privacy guarantees since there is no single point of failure. Among voting schemes with publicly verifiable tallies, some focus on strong coercion resistance properties [30], which is out of scope for our setting. Other schemes focus on allowing the voter to verify that their vote is cast as they intended from an untrusted device [2, 13, 15, 27, 33, 38, 49]. In this work, we do not distinguish the intentions of a human voter from the intentions of their device. Therefore schemes based on code voting [29, 49] which are mainly focused on easy verification by human users, generally in return for requiring some non-collusion assumptions among the authorities are out of scope.

Boardroom voting schemes [24, 37, 40] are designed for a relatively small group of voters to act as equals: all voters participate in both voting and tallying. These schemes can be publicly verifiable (i.e. including for those who did not participate in the protocol), but suffer from problems of robustness and may fail if any party refuses to participate in decryption. They are therefore not suitable for large groups.

Zhang et al. [63] show how to use a distributed protocol to select a verifiably-random subset (weighted by *stake*) to act as a decryption committee. Other participants can either vote directly or delegate their votes to experts, a choice that is protected with privacy guarantees. Voting occurs in epochs, after which a new decryption committee is chosen. Their scheme achieves different goals from ours, as it focuses on treasury management for cryptocurrencies and it incorporates a vote weighting mechanism based on each voter's stake along with a stake delegation mechanism. Furthermore it does not provide a method that overcomes the *single-shot* limitation when revealing the aggregate votes on any particular topic and does not address the issue of participation incentives.

Our setting and contribution. In this work, we define an opinion aggregation scheme for one topic as $\mathcal{F}_{\text{vote}}^{B,n,t}$ (Fig. 2, Sec. 6). We provide protocol $\Pi_{\text{vote}}^{B,n,t}$ which provably (Theorem 9.4) realises $\mathcal{F}_{\text{vote}}^{B,n,t}$. In $\Pi_{\text{vote}}^{B,n,t}$, a subset of participants are randomly selected to act as authorities. Everyone can vote once whenever they wish, but the authorities' responsibilities are allocated to a constantly refreshed subset of participants. Thus voting can happen whenever a participant wishes to, while decryption happens whenever there is a large enough anonymity set, therefore providing *privacy* (Theorem 9.2). In this respect, we improve upon boardroom voting in that total number of participants is large while the need for separate authorities is avoided and robustness against misbehaving parties is guaranteed. Our scheme provides *public verifiability* (Subsection 9.1), in the sense that each (electronic) voter can verify the inclusion of their vote and the overall correctness of the tally. We also achieve *liveness* (Theorem 9.3), informally meaning that (under reasonable assumptions) every vote will be promptly included. We note that our protocol is not *receipt-free.* Furthermore honest participants need to interact with the system beyond their own voting phase, which however we deem an acceptable tradeoff in the era of always-on mobile applications. In order to motivate parties to stay online, we further propose a suitable incentive mechanism.

We also define *continuous* opinion aggregation as a protocol in which a stream of topics is presented and parties can vote for each. One independent execution of $\Pi_{\text{vote}}^{B,n,t}$ per topic trivially realises continuous opinion aggregation. This setup can be used in practice for enabling privacy-preserving decentralised content curation on a social media platform.

2 PRELIMINARIES

Our protocol builds upon a number of preexisting cryptographic constructions:

- An additive-homomorphic public key encryption scheme with algorithms ⟨keyGen, enc, dec, pk⟩, realisable by, e.g., Paillier's [47] or the exponential version of ElGamal's encryption schemes [21].
- An additive-homomorphic secret sharing scheme with algorithms ⟨share, reconstruct⟩, realisable by, e.g., Shamir's scheme [50, 51].
- A public key infrastructure (PKI) for storing parties' keypairs abstracted via $\mathcal{G}_{\text{PKI}}$, realisable by having each party generate its keypair locally and adding its public key to a bulletin board [35], or a suitable smart contract running in a Turing-complete, decentralised blockchain, e.g. Ethereum [61] or Cardano [12]. Either method ensures no central authority can influence the result, i.e. that $\mathcal{G}_{\text{PKI}}$ is decentralised.
- A non-interactive zero-knowledge proof system for proving vote and share validity, abstracted via $\mathcal{F}_{\text{proof}}$, that can be instantiated, e.g., with one of Groth's schemes [25].
- A *common reference string* (CRS) which is a public key of the encryption scheme mentioned above, abstracted via $\mathcal{F}_{\text{CRS}}$. An entity that knows the corresponding private key may prove arbitrary statements to $\mathcal{F}_{\text{proof}}$. It is realisable via a suitable *multiparty computation* (MPC) protocol, e.g. [18], for the function keyGen, which returns the public key to all parties and discards the secret key.
- A decentralised protocol for provably handling valid votes in a publicly-verifiable way abstracted via $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$, that can be realised by an MPC protocol, or by a smart contract, similarly to $\mathcal{G}_{\text{PKI}}$. If MPC is chosen, public verifiability that extends beyond the parties actively implicated in the MPC protocol can be achieved with verifiable MPC [5]. Like in the case of $\mathcal{G}_{\text{PKI}}$, either method ensures $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ is decentralised.

We exploit the additive-homomorphic property of the public key encryption scheme to achieve voter privacy: multiple encrypted votes are added together so, when decrypted, only the aggregate result, not individual votes, are published.

In an *additive-homomorphic t-out-of-n secret sharing scheme* the following is possible: B parties independently share B distinct secrets. Then t sums of shares are created, where each is the result of adding B shares, each coming from a distinct secret and with no reuse of shares across the t sums. The reconstruction of the t sums reveals the sum of the B secrets. For example, for $B = t = 2$ and $n = 3$, secrets s_1 and s_2 are shared into $(\tilde{s}_{1,1}, \tilde{s}_{1,2}, \tilde{s}_{1,3})$ and $(\tilde{s}_{2,1}, \tilde{s}_{2,2}, \tilde{s}_{2,3})$ respectively. Then the summed secret $s_1 + s_2$ can be reconstructed using e.g. $\tilde{s}_{1,1} + \tilde{s}_{2,3}$ and $\tilde{s}_{1,2} + \tilde{s}_{2,2}$.

We leverage such a scheme to ensure that no party can individually decrypt votes before a protocol-wide constant number of voters B has voted. Honest behaviour prescribes decrypting and publishing only the sum of B encrypted shares. The system parameters are chosen so that it is exceedingly unlikely for any single vote to be secret-shared to t or more malicious players.

The version of the *PKI* used in this work is simply a store of private and public keys for all parties. $\mathcal{G}_{\text{PKI}}$ also is responsible for generating keys and only leaks the private key to its owner, whereas it may send the public key of any player to any other player. This modelling simply aids the separation of concerns and does not impose a strong assumption, as the realization is as simple as having parties generate their keypairs locally and publishing their public keys to a bulletin board or blockchain.

This work leverages an idealised zero-knowledge proof system, $\mathcal{F}_{\text{proof}}$, to ensure that every vote has a value among $-1, 0, 1$ and that it has been secret-shared and encrypted correctly, and additionally to ensure that every aggregate share is the result of the correct decryption of a correct sum of valid ciphertexts.

We use the execution model of Universal Composition [11], exploiting its clearly defined and widely used entities (*interactive Turing instances* – ITIs) and interactions between them. Our security treatment however follows the standalone simulation paradigm [41] in the static corruption setting. At a high level, in this model the environment $\mathcal{E}$ represents voting parties. The adversary $\mathcal{A}$ can corrupt parties at the beginning of the protocol execution and controls network communication. Both $\mathcal{E}$ and $\mathcal{A}$ may execute the code of arbitrary PPT *interactive Turing Machines.* We define the *real-world protocol* $\Pi_{\text{vote}}^{B,n,t}$ and the *ideal-world functionality* $\mathcal{F}_{\text{vote}}^{B,n,t}$ so that, from the point of view of $\mathcal{E}$ and $\mathcal{A}$, these two are *computationally indistinguishable.* $\mathcal{E}$ and $\mathcal{A}$ can send messages to any ITI, including $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ and $\mathcal{G}_{\text{PKI}}$, but not to $\mathcal{F}_{\text{proof}}$ or $\mathcal{F}_{\text{CRS}}$. The latter are *local* functionalities and thus can only be accessed by protocol parties.

3 CONTINUOUS OPINION AGGREGATION

Definition 3.1. A scheme for *single-topic continuous opinion aggregation* accepts a single topic, a set of voting parties, and up to one up/down/abstain vote per party. It periodically publishes a tally of the votes. The detailed functional behaviour is specified in $\mathcal{F}_{\text{vote}}^{B,n,t}$ (Fig. 2).

In practice, social media platforms display a virtually endless stream of posts to its users. This situation can be accommodated in the decentralised setting by a *continuous opinion aggregation* scheme.

Definition 3.2. A scheme for *multi-topic continuous opinion aggregation* accepts a stream of topics, a stream of eligible parties per topic, and up to one up/down/abstain vote per eligible party per topic. It periodically publishes a tally of the votes of each topic.

This scheme can be instantiated by simply running one independent instance of $\mathcal{F}_{\text{vote}}^{B,n,t}$ per topic. Due to the independence of the executions any guarantees offered by $\mathcal{F}_{\text{vote}}^{B,n,t}$ can be extended to the continuous opinion aggregation scheme.

The property of *fairness* [22, 37], i.e. the guarantee that prior cast votes do not influence future votes, is not sought after on purpose. Indeed, in the setting of social media it is beneficial to have an up-to-date view of the public's reaction to a specific topic. This can promote user engagement, offer a current and realistic pulse of the public's opinions, help users decide which posts to pay attention to and ultimately aid them in choosing how to vote.

4 OVERVIEW OF $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$

$\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ is the functionality that facilitates parties' interactions, producing sets of decryptors for each vote and ensuring parties may only contribute honestly to the protocol. As we discussed previously, $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ can be realised in a decentralised manner.

In particular, $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ keeps track of the identities of parties implicated in the protocol. It also stores encrypted votes and decrypted batch shares. A party that wishes to vote asks $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ for a set of decryptors, which are decided by $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ through a special procedure discussed later. $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ ignores the request if the party has voted already, otherwise responds with the set of decryptors. The party then sends its vote, which $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ verifies is correctly created. If so, $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ stores the vote and takes a note to ignore future votes by the same party. A decryptor that has handled a batch may submit the result to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$. Once again, $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ verifies that the decrypted batch share was generated correctly and stores it. Lastly, parties may query both the encrypted vote shares and the decrypted batch shares from $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$.

As we will see below, $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ leverages the zero-knowledge proof functionality $\mathcal{F}_{\text{proof}}$ to verify the correctness of votes and batch shares. This way all possible malicious actions are effectively checked for and guarded against. This functionality has to be realised in a way that guarantees its randomness cannot be biased (e.g. by an MPC or a smart contract with good quality randomness) to ensure decentralisation.

5 OVERVIEW OF THE CONSTRUCTION

Consider a set of parties $\mathcal{P}$, of which up to s may be corrupted. Our analysis is conducted in the static corruption setting, therefore all corruptions are decided by $\mathcal{A}$ before the beginning of the execution and no further corruptions are allowed during execution. Both the real-world protocol and the ideal-world functionality (and therefore the execution) are parametrised by the following constants:

- B, the size of each batch.
- n, the number of decryptors each vote is secret-shared for.
- t, the number of shares needed to reconstruct a secret.

Protocol Description. We now focus on the real-world execution. Honest parties follow the $\Pi_{\text{vote}}^{B,n,t}$ protocol (Fig. 8). In the initialisation phase of the execution, each participant generates its keypair. This happens when $\mathcal{E}$ sends (INIT) to a party (Fig. 8, l. 1), which in turn asks $\mathcal{G}_{\text{PKI}}$ to generate the keypair (Fig. 3, l. 1). Note that, even though it is possible for $\mathcal{E}$ to skip to the next phase before initialisation is complete by sending a (READ), (VOTE, v) or (DRAINBATCH) message to a party, this will lead to the first activation of $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 8, ll. 6, 21 and 34 respectively), which triggers the initialisation of $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 1, ll. 1-9) in which $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ halts if there is any non-initialised party, leading to protocol failure (and a satisfaction of all security properties in vacuum). We thus focus only on the cases in which initialisation of all parties takes place before the next phase.

Subsequently execution moves to the voting phase. $\mathcal{E}$ may at any time send an instruction to any protocol party: (VOTE, v) instructs it to cast vote v, (DRAINBATCH) tells it to attempt decryption of a full batch of votes, and (READ) has it read and reconstruct decrypted vote tallies. These are described in detail below. Note that $\mathcal{E}$ decides the value of the vote of honest parties, since $\mathcal{E}$ models, among others, the human users using a practical software implementation of our protocol.

When an honest party is instructed to vote, it does so in four consecutive steps. It first asks $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ to provide it with a set of decryptors (Fig. 8, l. 21). $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ first ensures that the party has not attempted to vote (successfully or not) in the past (Fig. 1, l. 11) and then chooses the decryptor set according to the following logic: If the previous party that attempted to vote failed to do so (only possible for a malicious party, see below), $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ reuses the same set of decryptors (Fig. 1, l. 17 was executed but l. 20 was not, so Fig. 1, l. 14 is not run and thus Fig. 1, l. 18 is executed with the previously stored `decryptors`). Otherwise, if a number of votes that is a multiple of B has been successfully cast up to that moment, a uniformly random n-sized subset of $\mathcal{P}$ is chosen (Fig. 6, ll. 5-6). In case the number of successfully cast votes is not a multiple of B, the previously used set of decryptors is used again (Fig. 6, l. 7). This method of selecting decryptors ensures that valid batches will be formed in a timely manner, while a malicious party cannot sabota ge the completion of a valid batch of votes by asking for a set of decryptors and then stalling, thus negatively affecting liveness.

In the second step, the party secret shares its vote and encrypts each share under the public key of the respective decryptor (Fig. 8, ll. 22-30). Note that the randomness for the SHARE and ENC algorithms is sampled explicitly so that it can be subsequently passed to $\mathcal{F}_{\text{proof}}$.

In the third step, the party submits its plaintext vote, the ciphertexts it just built, the corresponding decryptors, the randomness it used and the CRS to $\mathcal{F}_{\text{proof}}$ (Fig. 8, l. 31), which in turn ensures that the vote is -1, 0 or 1 and that the ciphertexts have been generated correctly. $\mathcal{F}_{\text{proof}}$ then stores this fact along with the passed CRS and returns control to the party (Fig. 5, ll. 1-19). In a realistic software implementation of the protocol, the party would locally generate a zero-knowledge proof of these facts instead.

In the fourth step, the party sends the encrypted shares to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 8, l. 32). $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ then checks whether the ciphertexts were generated correctly by asking $\mathcal{F}_{\text{proof}}$ (Fig. 1, l. 19). We note that in a practical implementation, the party would instead have to pass the noninteractive zero-knowledge proof to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ for verification instead. If verification succeeds, $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ records the vote locally (Fig. 1, l. 21). The party has now voted successfully.

We note that in the UC execution model, a running ITI T cannot be interrupted by another machine; it is always allowed to complete its current chunk of computation. This is realistic, as secure real-world software is not interrupted by network messages while performing computation. If T sends a message m to another machine T', then T' starts to run the code that corresponds to m. Furthermore, honest ITI failure is not modelled separately, but as part of adversarial corruption.

Observe now that the communication flow when an honest party P votes is $P \to \mathcal{G}^{\mathcal{P}}_{\text{VoteBox}} \to P \to (\mathcal{G}_{\text{PKI}} \to P \to)^n\ \mathcal{F}_{\text{CRS}} \to P \to \mathcal{F}_{\text{proof}} \to P \to \mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$. There is no instant in which this flow is interrupted by $\mathcal{A}$ or $\mathcal{E}$ and all functionalities are always honest, therefore this flow can only be broken if the party is corrupted. This observation allows $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ to deduce that if a party has not successfully completed voting before a new vote arrives, then this party is corrupted.

When an honest party receives (DRAINBATCH), it first reads all votes from $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 8, l. 34). It then partitions the votes into batches (Fig. 8, l. 35) using the deterministic Batch algorithm (Fig. 7). This algorithm forms batches using the oldest votes first, adding the oldest vote not considered yet in each new attempt to form a batch. Lexicographic order of decryptors is used to break ties in case multiple batches are possible. Note that Batch is general enough to be used if an arbitrarily more complex method of choosing decryptors were used by $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$.

After batching the votes, the party finds the minimum batch for which it is a decryptor and has not been drained by the party yet (Fig. 8, l. 36) and, if such a batch exists, the party drains it. Draining consists of the following steps. First the party calculates the sum of all the ciphertexts in the batch encrypted that were the result of encrypting a share with the public key of the party (Fig. 8, l. 39). As discussed previously, these ciphertexts are guaranteed to be valid, as $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ only stores ciphertexts that have been verified by $\mathcal{F}_{\text{proof}}$. Then the party decrypts the sum, proves to $\mathcal{F}_{\text{proof}}$ that the process was correctly done (Fig. 8, l. 42) and sends the resulting aggregate share to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 8, l. 43), which in turn verifies its correctness (Fig. 1, l. 25) and stores it (Fig. 1, l. 26). Note that, due to the additive-homomorphic properties of both the encryption and the secret sharing schemes, the resulting aggregate share can be combined with $t-1$ other aggregate shares generated by other decryptors using ciphertexts from the batch votes to reconstruct the sum of the votes in the batch.

Lastly, when a party receives (READ) from $\mathcal{E}$, it first reads all batch shares from $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ (Fig. 8, l. 6). For each batch, the party tries to reconstruct all possible combinations of t aggregate shares from this batch until a valid vote is extracted. This vote is added to the sum of votes and finally the end result is output (Fig. 8, ll. 7-18).

We note that $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ can be readily implemented in a blockchain that supports Turing-complete, stateful smart contracts, such as Ethereum [61] or Cardano [12]. To avoid possible attacks (e.g. any single party affecting the choice of decryptors), special care should be given to ensuring that the randomness used for choosing decryptors is unpredictable. To that end, a suitable distributed protocol for implementing a randomness beacon such as [28, 44] should be employed. Furthermore, various optimisations should be designed and incorporated to keep both the overall and the per-party smart contract cost low. Since our protocol has no synchrony requirements, we do not need any additional assumptions on top of the ones needed by the blockchain. To make analysis simpler, we assume that our functionalities are executed in an idealised manner and do not explicitly interact with a blockchain.

The parameter n is considered the security parameter and used as such in Theorem 9.2, which gives specific privacy guarantees. At a high level it states that, as long as $\frac{t}{n} > \frac{s}{|\mathcal{P}|}$ (where s is the number of corrupted players), the probability of any one set of decryptors throughout the entire execution containing t or more corrupted parties is negligible in n. This theorem provides a useful guideline for choosing safe values for the protocol parameters n, t given specific expectations on the number of corruptions s and total number of parties $|\mathcal{P}|$. It also confirms the following appealing intuition: for the voting system to be private, the ratio of the minimum shares needed to reconstruct a batch over the shares each vote has been split into must exceed the ratio of corrupted to total players overall.

The parameter B on the other hand can be treated separately. Changing its value is connected to trading off anonymity set size for batch frequency. Indeed, for bigger values of B more votes need to be cast to complete a batch, therefore the anonymity set each party enjoys is bigger, but at the same time each party will need to wait for more votes to be submitted by other parties before this bigger batch is completed and the party's vote is counted. Theorem 9.3 proves that a party has to wait for at most $B - 1$ more valid votes to be cast before its vote may be counted.

Furthermore, individual verifiability in our scheme is very straightforward: each voter can see their vote on the Bulletin Board, observe whether it has been incorporated into a batch, and, if so, verify the aggregation of the batch. (Recall that we do not distinguish a human operator from the computer that casts the vote.) Section 10 contains further discussion on the parameters.

Properties of our construction modelled as a functionality. In the ideal world, the protocol is replaced by the functionality $\mathcal{F}_{\text{vote}}^{B,n,t}$. As we will see later, we prove that $\Pi_{\text{vote}}^{B,n,t}$ realises the functionality $\mathcal{F}_{\text{vote}}^{B,n,t}$. The latter receives and handles inputs from all honest parties, whereas inputs from malicious parties are forwarded directly to the simulator $\mathcal{S}$. Likewise, an output addressed from a malicious party to $\mathcal{E}$ is passed by $\mathcal{S}$ to $\mathcal{F}_{\text{vote}}^{B,n,t}$ and subsequently forwarded to $\mathcal{E}$ unchanged, via the respective dummy party ITI.$\mathcal{F}_{\text{vote}}^{B,n,t}$ is responsible for handling the same messages from $\mathcal{E}$ that honest parties handle in the real world. In the initialisation phase, it asks $\mathcal{S}$ to generate a keypair for each party that is initialised. In the voting phase, when a party is instructed to vote, $\mathcal{F}_{\text{vote}}^{B,n,t}$ asks for and receives from $\mathcal{S}$ the corresponding vote number and decryptors, it checks that the vote number has not been reused and that the decryptor set does not include t or more malicious parties. It then stores the vote locally, before informing $\mathcal{S}$ of the successful vote. $\mathcal{S}$ submits the vote to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$. When a party is instructed to drain a batch, $\mathcal{F}_{\text{vote}}^{B,n,t}$ finds the oldest complete, non-drained batch for this player, if any, following steps similar to the real-world protocol, and marks this batch as drained by this party. If the batch has been drained by t parties, $\mathcal{F}_{\text{vote}}^{B,n,t}$ then adds together the honest votes of the batch, asks $\mathcal{S}$ for the sum of the votes of the malicious voters of the batch, updates the locally stored results and sends the sum of the honest votes to $\mathcal{S}$. In any case, $\mathcal{F}_{\text{vote}}^{B,n,t}$ informs $\mathcal{S}$ on the outcome of the draining. If draining succeeded, once again $\mathcal{S}$ submits the result to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$. $\mathcal{S}$ may also inform $\mathcal{F}_{\text{vote}}^{B,n,t}$ that a particular malicious party has drained a particular batch. Lastly, when a party is instructed to read the results, $\mathcal{F}_{\text{vote}}^{B,n,t}$ simply returns them as stored.

Observe that the functionality always counts in the results all honest votes that have been included in a complete, fully drained batch. Furthermore, as we will see in the proof of Theorem 9.4, $\mathcal{S}$ knows the votes of all malicious parties and reports them truthfully to $\mathcal{F}_{\text{vote}}^{B,n,t}$, which in turn includes them in the results as well. These two facts together show that we have public verifiability in the ideal world, i.e. we can be certain that no votes have been neglected in the result, as long as they are part of a drained batch. Furthermore, given that Theorem 9.4 proves indistinguishability of the real and the ideal world without any assumptions on the number of corruptions, only having to trust the underlying ideal functionalities, we deduce that the real world protocol provides public verifiability as well. Observe that the only limitation on the number of corruptions is imposed by l. 17 of Fig. 2, which is only relevant for providing explicit privacy guarantees; the indistinguis hability proof would go through if this line was missing, in which case any amount of corruptions would be allowed.

Incentivising Decryptor Participation. Our construction expects honest parties to be able to come online if needed throughout the entire protocol execution, even after casting their vote. This is so they can serve as decryptors in case they are chosen by $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$. On the other hand, some users of the protocol would presumably prefer to come online just to vote and be able to stay offline throughout the rest of the protocol; the aim of the protocol is to facilitate expressing user opinion, so we expect it to be used by parties that find utility in voting. A single party staying offline except when voting would fail to fulfill its role as decryptor, but this would not cause any problem as there is redundancy built into the secret sharing scheme as long as $n > t$. If however all parties were to follow this strategy, then no party would ever be online to act as a decryptor and therefore no batch would ever be decrypted. This situation constitutes a *tragedy of the commons*.

If parties are using a blockchain to execute the protocol and that blockchain is endowed with a cryptocurrency, then we propose the following incentive scheme to avoid this unfortunate situation. At a high level, a voter has to pay a fee to cast its vote and conversely a decryptor is paid out a set sum for its service. The two values are tuned so that each party that stays online throughout the execution gets 0 coins on expectation, but a party that only comes online to vote and stays offline for the rest of the execution loses coins.

More specifically, let X be the random variable that counts the number of batches for which a specific party is chosen as decryptor throughout the entire execution, let b be the total number of batches and p the probability that the party is chosen as the decryptor of a particular batch, the latter stemming from Fig. 6, l. 6. Since the decryptors of each batch are chosen independently, the expected value of X is $\mathbb{E}(X) = bp$. Since we have a total of $|\mathcal{P}|$ votes and each group of B votes forms a batch, it is $b = \lfloor \frac{|\mathcal{P}|}{B} \rfloor$. If we further assume that B divides $|\mathcal{P}|$ (as is recommended later), then $b = \frac{|\mathcal{P}|}{B}$. Regarding p, it is $p = \binom{|\mathcal{P}|-1}{n-1} / \binom{|\mathcal{P}|}{n} = \frac{n}{|\mathcal{P}|}$. On aggregate, $\mathbb{E}(X) = \frac{n}{B}$. Therefore, for any $c > 0$, if voting is charged with $c\frac{n}{B}$ coins and successfully decrypting a batch is compensated with c coins, then a party that is online throughout the execution gains on expectation 0 coins, whereas a party that comes online only to vote has to pay $c\frac{n}{B}$ coins. Realistic implementations should choose a sufficiently small c so that parties are not denied participation due to low available funds or due to fears of becoming a decryptor too few times to cover the cost of voting, but also c should be sufficiently large to ensure voting is expensive enough for most parties to prefer staying online to be compensated. This solution avoids the tragedy of the commons while leaving, on expectation, monetary value out of the equation.

A necessary sanity check is whether the contract holds enough money to compensate decryptors at every instant. After B votes, the contract has been paid $c\frac{n}{B} \cdot B = cn$ coins and has paid out nothing. If all n chosen decryptors handle the resulting batch before the next vote is cast, they will be compensated with cn coins in total, thus the contract has just enough money to pay them after the first batch. The same reasoning can be extended to every later batch.

Note that this incentives scheme does not hurt the privacy of the overall protocol, as votes are still encrypted in exactly the same way and payments are independent from the content of the encrypted vote. Furthermore, the instances in which each party interacts with the system and their observability does not change. The fact that this incentives scheme leverages potentially traceable coins introduces a new potential privacy goal, namely to ensure funds untraceability. Extensive research [6, 7, 9, 10, 31, 34, 54, 55, 65] has been conducted on untraceable money.

6 FORMAL DESCRIPTION OF $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$

The $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ functionality provides the core functions regarding handling vote and aggregate share submissions. When a player asks to vote, it samples decryptors (using getDecryptors$_{n,B,\mathcal{P}}$(), Fig 6), checks the validity of the vote and stores it. Similarly, when a player asks to submit an aggregate share, $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ verifies and stores it.

Functionality $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$

```
1:  Initialisation:
2:      voted ← ∅
3:      $\mathcal{V}$ ← ∅
4:      lastVoteSucceeded ← true
5:      send (GETCRS) to $\mathcal{F}_{\text{CRS}}$ and assign reply to $pk_{\text{CRS}}$
6:      for $P \in \mathcal{P}$ do
7:          send (PK, $P$) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $pk$
8:          if $pk$ is not a valid public key then halt
9:      end for

10: On (GETDECRYPTORS) by $P$:
11:     if $P \in$ voted then yield execution token
12:     add $P$ to voted
13:     if lastVoteSucceeded then
14:         decryptors ← getDecryptors$_{n,B,\mathcal{P}}$()
15:         parse decryptors as $(r, \langle P_i \rangle_{i \in [n]})$
16:     end if// else reuse previous decryptors
17:     lastVoteSucceeded ← false
18:     reply decryptors to $P$ and expect reply (VOTE, $r$, $\langle c_i, P_i \rangle_{i \in [n]}$)
        by $P$ // received $r$ must match the one stored in decryptors
19:     send (VERIFYVOTE, $P$, $pk_{\text{CRS}}$, $\langle c_i, P_i \rangle_{i \in [n]}$) to $\mathcal{F}_{\text{proof}}$ and ensure
        reply is (OK)
20:     lastVoteSucceeded ← true
21:     add (VOTE, $r$, $\langle c_i, P_i \rangle_{i \in [n]}$) to $\mathcal{V}$

22: On (READ) by $P$:
23:     reply $\mathcal{V}$

24: On (BATCHSHARE, $j$, $S$) by $P$:
25:     send (VERIFYSHARE, $P$, $pk_{\text{CRS}}$, $j$, $S$) to $\mathcal{F}_{\text{proof}}$ and ensure reply is
        (OK)
26:     add (BATCHSHARE, $j$, $S$) to $\mathcal{V}$
```

Figure 1

7 FORMAL DESCRIPTION OF $\mathcal{F}^{B,n,t}_{\text{vote}}$

The ideal-world functionality $\mathcal{F}^{B,n,t}_{\text{vote}}$ is the idealised abstraction of the intended protocol behaviour. It aggregates locally the votes of all honest players. It also expects specific messages from the adversary which let it know when a corrupt player votes or drains a batch.

Functionality $\mathcal{F}^{B,n,t}_{\text{vote}}$

```
1:  Initialisation:
2:      results ← 0
3:      votes ← ∅
4:      maliciousEntries ← ∅

5:  On $M$ by corrupted $P$:
6:      send (FORWARD, $P$, $M$) to $\mathcal{A}$

7:  On (FORWARD, $P$, $M$) by $\mathcal{A}$ where $P$ is corrupted:
8:      send $M$ to $P$

9:  On first (INIT) by $P$: // one INIT per player
10:     send (INIT, $P$) to $\mathcal{A}$ and assign reply to $pk_P$
11:     reply $pk_P$ to $P$

12: On (READ) by $P$:
13:     reply (RESULTS, results)

14: On first (VOTE, $v$) by $P$:
15:     send (GETDECRYPTORS, $P$) to $\mathcal{A}$, expect reply $(r, \langle P_k \rangle_{k \in [n]})$ by
        $\mathcal{A}$
16:     if $r$ has appeared again in votes or in a VOTED message by $\mathcal{A}$
        then halt
17:     if at least $t$ players in $\langle P_k \rangle_{k \in [n]}$ are malicious then halt
```

```
        append $(r, P, v, \langle P_k \rangle_{k\in[n]})$ to votes.
        send (VOTED, $P$, $\langle P_k \rangle_{k\in[n]}$) to $\mathcal{A}$

    On (VOTED, $r$, $\langle P_k \rangle_{k\in[n]}$) by $\mathcal{A}$:
        if $r$ has appeared in votes or in maliciousEntries then halt
        add $(r, \langle P_k \rangle_{k\in[n]})$ to maliciousEntries

    On (DRAINED, $j$, $P$) by $\mathcal{A}$ where $P$ is corrupted:
        $\mathrm{HasDrained}(P, \mathcal{B}_j) \leftarrow \mathrm{True}$
        if there are exactly $t$ distinct $P_i : \mathrm{HasDrained}(P_i, \mathcal{B}_j) = \mathrm{True}$ then
            newVotes $\leftarrow \sum_{i\in[|B|]} v : (\mathcal{B}_j.\mathrm{r}_i, v, \cdot) \in$ votes
            send (GETMALICIOUSVOTES, $j$) to $\mathcal{A}$ and expect reply maliciousVotes
            results $\leftarrow$ results + newVotes + maliciousVotes
        end if
        reply (OK)

    On (GETHONESTVOTES, $j$) by $\mathcal{A}$:
        send (READ) to $\mathcal{G}^{\mathcal{P}}_{\mathrm{VoteBox}}$, keep (VOTE, _) entries from reply and collect them into castVotes
        if the $j$-th batch in castVotes is complete then
            send (HONESTVOTES, $j$, $\sum_{i\in[|B|]} v : (\mathcal{B}_j.\mathrm{r}_i, v, \cdot) \in$ votes) to $\mathcal{A}$
        end if

    On (DRAINBATCH) by $P$:
        entries $\leftarrow$ votes.map($(r, v, \langle P_k \rangle_{k\in[n]}) \mapsto (r, \langle P_k \rangle)$) $\cup$ maliciousEntries
        $\mathcal{B} = \langle \{\mathrm{r} : \langle r_i \rangle_{i\in[B]}, \mathrm{P} : \langle P_j \rangle\} \rangle \leftarrow \mathrm{Batch}(\mathrm{entries})$
        $j \leftarrow \arg\min_{i\in[|\mathcal{B}|]} \{P \in \mathcal{B}_i.\mathrm{P} \wedge \mathrm{HasDrained}(P, \mathcal{B}_i) = \mathrm{False}\}$
        if $j$ exists then $\mathrm{HasDrained}(P, \mathcal{B}_j) \leftarrow \mathrm{True}$ else return
        if there are exactly $t$ distinct $P_i : \mathrm{HasDrained}(P_i, \mathcal{B}_j) = \mathrm{True}$ then
            newVotes $\leftarrow \sum_{i\in[|B|]} v : (\mathcal{B}_j.\mathrm{r}_i, v, \cdot) \in$ votes
            send (GETMALICIOUSVOTES, $j$) to $\mathcal{A}$ and expect reply maliciousVotes
            results $\leftarrow$ results + newVotes + maliciousVotes
        end if
        send (BATCHSHARE, $P$, $j$) to $\mathcal{A}$
```

Figure 2

8 FORMAL DESCRIPTION OF THE CONSTRUCTION & PROTOCOL

The t-out-of-n secret sharing scheme provides the following algorithms:

- $(s_1, \ldots, s_n) \leftarrow \textsc{share}(m; r)$ where r is sampled uniformly from $\mathcal{R}_{\mathrm{share}}$,
- $m \leftarrow \textsc{reconstruct}(s_1, \ldots, s_t)$.

The asymmetric encryption scheme provides these algorithms:

- $(pk, sk) \xleftarrow{\$} \textsc{keyGen}()$,
- $c \leftarrow \textsc{enc}(m, pk; r)$ where r is sampled uniformly from $\mathcal{R}_{\mathrm{enc}}$,
- $m \leftarrow \textsc{dec}(c, sk)$.

Key management is governed by the $\mathcal{G}_{\mathrm{PKI}}$ functionality.

Functionality $\mathcal{G}_{\mathrm{PKI}}$

```
On (INIT) by $P$:
    ensure this is the first time we receive (INIT) by $P$
    $(pk_P, sk_P) \xleftarrow{\$} \textsc{keyGen}()$
    reply $(pk_P)$

On (PK, $P$):
    reply $(pk_P)$

On (SK) by $P$:
    reply $(sk_P)$

On (VERIFY, $sk$, $P$):
    if $sk = sk_P$ then reply (OK) else reply (ERROR)
```

Figure 3: Key management functionality $\mathcal{G}_{\mathrm{PKI}}$.

The $\mathcal{F}_{\mathrm{CRS}}$ functionality initially generates a CRS public key and subsequently always returns it. The secret key is discarded.

Functionality $\mathcal{F}_{\mathrm{CRS}}$

```
Initialisation:
    $(pk_{\mathrm{CRS}}, \_) \xleftarrow{\$} \textsc{keyGen}()$

On (GETCRS) by $P$:
    reply $pk_{\mathrm{CRS}}$
```

Figure 4: Common Reference String functionality $\mathcal{F}_{\mathrm{CRS}}$.

The $\mathcal{F}_{\mathrm{proof}}$ functionality handles proofs of correct construction of votes and aggregate shares.

Functionality $\mathcal{F}_{\mathrm{proof}}$

```
On (PROVEVOTE, $v$, $g_0$, $pk_{\mathrm{CRS}}$, $\langle g^s_k, g^{\mathrm{CRS}}_k \rangle_{k\in[n]}$, $\langle c^s_k, c^{\mathrm{CRS}}_k, P_k \rangle_{k\in[n]}$, $sk$) by $P$:
    if $sk$ is given as input and $\textsc{pk}(sk) = pk_{\mathrm{CRS}}$ then
        store $(P, \langle c^s_k, c^{\mathrm{CRS}}_k, P_k \rangle_{k\in[n]}, pk_{\mathrm{CRS}})$
        reply (OK)
    end if
    if $v \notin \{-1, 0, 1\}$ then
        reply (ERROR)
    end if
    $\langle s_k \rangle_{k\in[n]} \leftarrow \textsc{share}(v; g_0)$
    for $k$ from 1 to $n$ do
        send (PK, $P_k$) to $\mathcal{G}_{\mathrm{PKI}}$ and assign reply to $pk_k$
```

```
12:         $c^s \leftarrow \textsc{enc}(s_k, pk_k; g_k^s)$
13:         $c^{\text{CRS}} \leftarrow \textsc{enc}(s_k, pk_{\text{CRS}}; g_k^{\text{CRS}})$
14:         if $c^s \neq c_k^s \vee c^{\text{CRS}} \neq c_k^{\text{CRS}}$ then
15:             reply ($\textsc{error}$)
16:         end if
17:     end for
18:     store $(P, \langle c_k^s, c_k^{\text{CRS}}, P_k \rangle_{k \in [n]}, pk_{\text{CRS}})$
19:     reply ($\textsc{ok}$)

20: On ($\textsc{verifyVote}$, $P, pk_{\text{CRS}}, \langle c_k^s, c_k^{\text{CRS}}, P_k \rangle_{k \in [n]}$):
21:     if $(P, \langle c_k, c_k^{\text{CRS}}, P_k \rangle_{k \in [n]}, pk_{\text{CRS}})$ is stored then reply ($\textsc{ok}$)
    else reply ($\textsc{error}$)

22: On ($\textsc{proveShare}$, $r, S, pk_{\text{CRS}}, sk$) by $P$:
23:     if $\textsc{pk}(sk) = pk_{\text{CRS}}$ then
24:         store $(P, r, S, pk_{\text{CRS}})$
25:         reply ($\textsc{ok}$)
26:     end if
27:     send ($\textsc{verify}$, $sk$, $P$) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $T$
28:     if $T = (\textsc{error})$ then reply ($\textsc{error}$)
29:     send ($\textsc{read}$) to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$, keep ($\textsc{vote}$, $(r, \langle c_k, P_k \rangle_{k \in [n]})$)
    entries from reply and collect them into
    votes $= \langle r_i, \langle c_{i,k}, P_{i,k} \rangle_{k \in [n]} \rangle_{i \in [R]}$
30:     $\mathcal{B} = \langle \{\mathsf{r} : \langle r_i \rangle_{i \in [B]}, \mathsf{P} : \langle P_j \rangle\} \rangle \leftarrow$ Batch(votes)
31:     if any entry in $\mathcal{B}_r$ contains no ciphertext for $P$ then reply
    ($\textsc{error}$)
32:     $C \leftarrow \sum_{i \in [|B|]} c : (\mathcal{B}_r.\mathsf{r}_i, \langle \ldots, (c, P), \ldots \rangle) \in$ votes
33:     if $S = \textsc{dec}(C, sk)$ then
34:         store $(P, r, S, pk_{\text{CRS}})$
35:         reply ($\textsc{ok}$)
36:     else
37:         reply ($\textsc{error}$)
38:     end if

39: On ($\textsc{verifyShare}$, $P, pk_{\text{CRS}}, r, S$):
40:     if $(P, r, S, pk_{\text{CRS}})$ is stored then reply ($\textsc{ok}$) else reply ($\textsc{error}$)
```

Figure 5

getDecryptors$_{n,B,\mathcal{P}}$() is used internally by $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ to sample the decryptors of each vote.

Stateful Function getDecryptors$_{n,B,\mathcal{P}}$()

```
1:  Initialisation:
2:      $r \leftarrow 0$

3:  Execution:
4:      $r \leftarrow r + 1$
5:      if $r \mod B = 1$ then
6:          $K_r \xleftarrow{\$} \binom{\mathcal{P}}{n}$ // $\binom{\mathcal{P}}{n}$ is the set of $n$-sized subsets of $\mathcal{P}$
7:      else
8:          $K_r \leftarrow K_{r-1}$
9:      end if
10:     return $(r, K_r)$
```

Figure 6

The Batch$_{B,t}$() function is used internally by $\mathcal{F}_{\text{proof}}$, the protocol $\Pi_{\text{vote}}^{B,n,t}$ and the functionality $\mathcal{F}_{\text{vote}}^{B,n,t}$. It groups votes into valid batches in a deterministic manner.

Algorithm Batch$_{B,t}$(votes)

```
1:  $\mathcal{B} \leftarrow \langle \rangle$ // vector
2:  for top from $B$ to max{votes.map($(r, \cdot) \mapsto r$)} do
3:      entries_up_to_top $\leftarrow$ votes.filter($(r, \cdot) \mapsto r \leq$ top)
4:      // a valid batch consists of a vector of rounds
    r $= \langle r_i \in [\text{top}] \rangle_{i \in [B]}$ ($B$ natural numbers up to top) and a vector
    of parties P $= \langle P_i \in \mathcal{P} \rangle_{i \in [m]}$ of length $t \leq m \leq n$ such that $P_i$ is
    in every vote of votes.filter($r \in$ r). If some $P \in \mathcal{P}$ appears
    multiple times in P, it must also be in each each vote of interest at
    least as many times. ($\forall i \in [m]$, $\forall$vote $\in$ votes.filter($r \in$
    r), |vote.filter($P = P_i$)| $\geq$ |P.filter($P = P_i$)|)
5:      if $\exists$ a valid batch {r : $\langle r_i \rangle_{i \in [B]}$, P : $\langle P_i \rangle_{i \in [m]}$} in
    entries_up_to_top then
6:          append the batch to $\mathcal{B}$; if there are more than one valid
    batches, choose the minimum, compared based on the
    lexicographic ordering of the concatenation of the identifiers of all
    involved parties, $P_1 P_2 \ldots P_m$. // It is impossible that exactly the
    same parties are involved in two different batches, as that would
    mean that the batches' difference is in their entries' $r$. In that case
    each of the two batches would have $B - 1$ entries with $r \neq$ top with
    at least one difference in these entries. This means that there would
    be at least $B$ distinct entries with $r \neq$ top and $t$ common parties.
    These entries would however form a batch $D$ without the entry
    that has $r =$ top. If top $= B$, the exclusion of the entry with $r = B$
    would mean that there are not enough entries to form any batch,
    else $D$ would have been already consumed in a previous iteration.
7:          remove every entry involved in the new batch from
    entries // This removal invalidates all alternative batches of the
    line above, therefore asserting the argument that there cannot be
    two or more valid, disjoint batches per iteration.
8:      end if
9:  end for
10: return $\mathcal{B}$
```

Figure 7

Protocol $\Pi_{\text{vote}}^{B,n,t}$ is executed by each protocol party. At any time, $\mathcal{E}$ can ask a party to vote, drain the next available batch, or read decrypted results. As we show in Theorem 9.4, $\Pi_{\text{vote}}^{B,n,t}$ realises $\mathcal{F}_{\text{vote}}^{B,n,t}$.

Process $\Pi_{\text{vote}}^{B,n,t}$ (self is P)

```
1:  On first ($\textsc{init}$) by $\mathcal{E}$:
2:      send ($\textsc{getCRS}$) to $\mathcal{F}_{\text{CRS}}$ and assign reply to $pk_{\text{CRS}}$
3:      send ($\textsc{init}$) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $pk$
4:      reply $pk$ to $\mathcal{E}$

5:  On ($\textsc{read}$) by $\mathcal{E}$:
 6:    send (READ) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$, keep (BATCHSHARE, $j, S$) entries from reply and collect them into shares = $\langle j, S\rangle$
 7:    results $\leftarrow 0$
 8:    for $i$ from 1 to max{shares.map($(j,S) \mapsto j$)} do
 9:        while there are $t$ $(i, S)$ entries in shares that have not been given as input to RECONSTRUCT() while handling the current READ message do
               // makes robust against malicious BATCHSHARE entries
10:            assign the shares of these $t$ entries to $(S_1, \dots, S_t)$
11:            newVotes $\leftarrow$ RECONSTRUCT$(S_1, \dots, S_t)$
12:            if newVotes $\neq \perp$ then
13:                results $\leftarrow$ results + newVotes
14:                break
15:            end if
16:        end while
17:    end for
18:    reply results to $\mathcal{E}$

19: On first (VOTE, $v$) by $\mathcal{E}$:
20:    ensure $v \in \{-1, 0, 1\}$, otherwise ignore
21:    send (GETDECRYPTORS) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ and assign reply to $(r, \langle P_k\rangle_{k\in[n]})$
22:    $g_0 \xleftarrow{\$} \mathcal{R}_{\text{share}}$
23:    $\langle s_k\rangle_{k\in[n]} \leftarrow$ SHARE$(v; g_0)$
24:    for $k$ in 1 to $n$ do
25:        send (PK, $P_k$) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $pk_k$
26:        $g^s_k \xleftarrow{\$} \mathcal{R}_{\text{enc}}$
27:        $c^s_k \leftarrow$ ENC$(s_k, pk_k; g^s_k)$
28:        $g^{\text{CRS}}_k \xleftarrow{\$} \mathcal{R}_{\text{enc}}$
29:        $c^{\text{CRS}}_k \leftarrow$ ENC$(s_k, pk_{\text{CRS}}; g^{\text{CRS}}_k)$
30:    end for
31:    send (PROVEVOTE, $v, g_0, pk_{\text{CRS}}, \langle g^s_k, g^{\text{CRS}}_k\rangle_{k\in[n]}, \langle c^s_k, c^{\text{CRS}}_k, P_k\rangle_{k\in[n]}$) to $\mathcal{F}_{\text{proof}}$ and expect reply (OK)
32:    send (VOTE, $r, \langle c^s_k, c^{\text{CRS}}_k, P_k\rangle_{k\in[n]}$) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$

33: On (DRAINBATCH) by $\mathcal{E}$:
34:    send (READ) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$, keep (VOTE, $(r, \langle c_k, P_k\rangle_{k\in[n]})$) entries from reply and collect them into votes = $\langle r_i, \langle c_{i,k}, P_{i,k}\rangle_{k\in[n]}\rangle_{i\in[R]}$
35:    $\mathcal{B} = \langle\{$r : $\langle r_i\rangle_{i\in[B]}$, P : $\langle P_j\rangle\}\rangle \leftarrow$ Batch(votes)
36:    $j \leftarrow \arg\min_{i\in[|\mathcal{B}|]}\{P \in \mathcal{B}_i.\text{P} \wedge \text{Drained}(\mathcal{B}_i) = \text{False}\}$
37:    if $j$ exists then
38:        Drained$(\mathcal{B}_j) \leftarrow$ True
39:        $C \leftarrow \sum_{i\in[|B|]} c : (\mathcal{B}_j.\text{r}_i, \langle \dots, (c, P), \dots\rangle) \in$ votes
           // $\forall r \in \mathcal{B}_j.\text{r}, \exists (c, P)$ since $P \in \mathcal{B}_j.\text{P}$
40:        send (SK) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $sk$
41:        $S \leftarrow$ DEC$(C, sk)$
42:        send (PROVESHARE, $j, S, pk_{\text{CRS}}, sk$) to $\mathcal{F}_{\text{proof}}$ and expect reply (OK)
43:        send (BATCHSHARE, $j, S$) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$
44:    end if
```

Figure 8

9 SECURITY, PRIVACY & LIVENESS GUARANTEES

Here we formulate three theorems regarding the assurances provided by our scheme.

We note that the privacy of an individual honest voter P may be still be broken if $B - 1$ malicious players are included in the same batch as P, as they can act honestly until they learn the aggregate result of the batch and then subtract their own votes from the sum. In the current protocol, $\mathcal{E}$ can arrange the order of votes as it pleases, therefore this attack against privacy is feasible. In a practical implementation however, and given that there is a constant stream of votes, such an attack could be harder to carry out. An alternative protocol could instead have $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ decide the order of players' votes, thus thwarting this attack against privacy. Such a measure would however impede the ability of each party to vote when it decides to (which is the case in a practical implementation of the current protocol) and would make the protocol prone to stalling every time the expected voter happens to be unavailable. The implications of this attack depend on the probability distribution of votes, because much larger batches (anonymity sets) are required if the votes are highly biased. We therefore define privacy relative to an idealised reporting mechanism that tallies each batch.

Definition 9.1 (Privacy). In a real-world execution with parties in $\mathcal{P}$ of which at most s are malicious, let E be the event under which at least one of the sets of decryptors returned by getDecryptors$_{n,B,\mathcal{P}}$ contains at least t malicious parties. We say that $\Pi^{B,n,t}_{\text{vote}}$ is *private* if $\Pr[E] \leq \text{negl}(n)$ in $\text{EXEC}^{\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}, \mathcal{F}_{\text{proof}}}_{\Pi^{B,n,t}_{\text{vote}}, \mathcal{A}, \mathcal{E}}$.

THEOREM 9.2.

$$\forall \textit{ PPT } \mathcal{A}, \mathcal{E}, \forall \textit{ set of parties } \mathcal{P},$$
$$\forall 0 < \alpha \leq 1 : \alpha|\mathcal{P}| \in \mathbb{N}, n = \alpha|\mathcal{P}|,$$
$$\forall 0 < \gamma \leq 1 : \gamma n \in \mathbb{N}, t = \gamma n,$$
$$\forall 0 \leq \beta < 1 : \beta|\mathcal{P}| \in \mathbb{N}, s = \beta|\mathcal{P}|, B \in \mathbb{N},$$
$$\textit{If } \frac{t}{n} > \frac{s}{|\mathcal{P}|} \textit{ then } \Pi^{B,n,t}_{\text{vote}} \textit{ is private.}$$

Note that this does not preclude the possibility of the adversary learning all individual votes in specific cases. For example, if all honest parties are unanimous, then the adversary can deduce the vote of each just by looking at the aggregate tally.

PROOF OF THEOREM 9.2. In a real-world execution with parties in $\mathcal{P}$ of which at most s are malicious, let E be the event under which at least one of the sets of decryptors returned by getDecryptors$_{n,B,\mathcal{P}}$ contains at least t malicious parties. We will show that

$$\forall \text{ PPT } \mathcal{A}, \mathcal{E}, \forall \text{ set of parties } \mathcal{P},$$
$$\forall 0 < \alpha \leq 1 : \alpha|\mathcal{P}| \in \mathbb{N}, n = \alpha|\mathcal{P}|,$$
$$\forall 0 < \gamma \leq 1 : \gamma n \in \mathbb{N}, t = \gamma n,$$
$$\forall 0 \leq \beta < 1 : \beta|\mathcal{P}| \in \mathbb{N}, s = \beta|\mathcal{P}|, B \in \mathbb{N},$$
$$\text{If } \frac{t}{n} > \frac{s}{|\mathcal{P}|} \text{ then } \Pr[E] \leq \text{negl}(n) \text{ in } \text{EXEC}^{\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}, \mathcal{F}_{\text{proof}}}_{\Pi^{B,n,t}_{\text{vote}}, \mathcal{A}, \mathcal{E}}.$$

getDecryptors$_{n,B,\mathcal{P}}$ is called at most $|\mathcal{P}|$ times. The first time and every B times going forwards, a new, uniformly random subset of n players is chosen (Fig. 6, l. 6), for a maximum total of $\lceil \frac{|\mathcal{P}|}{B} \rceil$ independent choices. Each random choice of n decryptors can be modelled as n consecutive single-ball draws without replacement and with replacement scheme $R(\text{white}) = R(\text{black}) = 0$ (i.e. after drawing any ball, no balls are added back to the urn) from an urn that initially has 1 white and 1 black ball for each honest and each malicious player respectively, for a total of s black and $|\mathcal{P}| - s$ white balls. The random variable b = "number of black balls drawn after n draws" follows a hypergeometric distribution [52]. From the above we deduce that $\Pr[E] \le \Pr[b \ge t] \lceil \frac{|\mathcal{P}|}{B} \rceil$. It is

$$E[b] = n\frac{s}{|\mathcal{P}|} \text{ and } \forall q \ge 0, \Pr[b \ge E[b] + qn] \le e^{-2q^2 n} \quad .$$

We want to upper bound $\Pr[b \ge t]$, therefore $t = E[b] + qn = n\frac{s}{|\mathcal{P}|} + qn \Leftrightarrow q = \frac{t}{n} - \frac{s}{|\mathcal{P}|}$, thus q is always positive due to the theorem prerequisite. It is

$$\Pr[b \ge t] \le e^{-2(\frac{t}{n} - \frac{s}{|\mathcal{P}|})^2 n} = e^{-2(\gamma - \beta)^2 n} \quad . \tag{1}$$

Therefore $\Pr[E] \le \lceil \frac{n}{\alpha B} \rceil e^{-2(\gamma-\beta)^2 n} \le \text{negl}(n)$. □

Theorem 9.3 (Liveness). *In a real-world execution, every honestly cast vote becomes part of a valid batch in $\mathcal{V}$ of $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ after at most $B - 1$ more votes for the topic at hand have been successfully cast.*

We note that no synchrony assumptions are made, therefore all party activations are controlled by $\mathcal{E}$. The latter may thus choose to never activate enough honest parties to complete a batch (with a vote message) or enough honest decryptors to decrypt it (with a drainBatch message). Nevertheless, in a realistic software implementation these two actions would be initiated spontaneously, not at the whim of $\mathcal{E}$.

Proof of Theorem 9.3. In a real-world execution, sending a getDecryptors and then a vote message to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ is the only way for P to vote, whether P is honest or malicious (since $\mathcal{V}$, which contains all votes, is stored locally by $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$). $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ only accepts votes for which the decryptors have been generated by $\text{getDecryptors}_{n,B,\mathcal{P}}$ (Fig. 1, ll. 14-18). $\text{getDecryptors}_{n,B,\mathcal{P}}$ in turn keeps choosing the same decryptors until a batch of submitted votes is complete (Fig. 6). In particular, a batch is complete every exactly B calls to $\text{getDecryptors}_{n,B,\mathcal{P}}$, as in case a player does not cast its vote successfully (by either getting its decryptors but not submitting its vote before another player tries to vote, i.e. not providing the expected reply of Fig. 1, l. 18, or by submitting an invalid vote, i.e. having Fig. 1, l. 19 fail), $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ reuses the same decryptors (as l. 20 of Fig. 1 was not run after l. 17 was run), therefore a party has to wait at most $B - 1$ successfully cast votes before its vote is included in a valid batch. □

Theorem 9.4 shows that the adversary cannot alter the tallies.

Theorem 9.4 (Security).

$$\forall \; PPT \; \mathcal{A}, \exists \; PPT \; \mathcal{S} : \forall \; PPT \; \mathcal{E} \; \textit{it is}$$

$$EXEC^{\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}, \mathcal{F}_{\text{proof}}, \mathcal{G}_{\text{PKI}}}_{\Pi^{B,n,t}_{\text{vote}}, \mathcal{A}, \mathcal{E}} \approx EXEC^{\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}, \mathcal{G}_{\text{PKI}}}_{\mathcal{S}, \mathcal{E}} \quad .$$

Proof of Theorem 9.4.

Algorithm $\text{findShare}(S_1, \ldots, S_{t-1}, x_t, V)$

Finds $S_t = (x_t, y_t) : \text{RECONSTRUCT}(S_1, \ldots, S_t) = V$ and the source polynomial of the secret, $P(x)$. Works for Shamir's Secret Sharing scheme [51].

```
1: for $i$ from 1 to $t-1$ do
2:     parse $S_i$ as $(x_i, y_i)$ // $x_i, y_i$ are finite field elements
3: end for
4: Solve linear equation $V = \sum_{i=1}^{t} y_i \prod_{m=1, m \ne j}^{m=t} \frac{x_m}{x_m - x_i}$ to find the only
   unknown, $y_t$ // $y_t$ is a finite field element
5: for $i$ from 1 to $t$ do
6:     $l_i(x) = \prod_{m=1, m \ne i}^{t} \frac{x - x_m}{x_i - x_m}$
7: end for
8: return $((x_t, y_t), \sum_{i=1}^{t} y_i l_i)$
```

Figure 9

Simulator $\mathcal{S}$ – all messages by $\mathcal{F}^{B,n,t}_{\text{vote}}$, unless otherwise noted

$\mathcal{S}$ takes over the interface of $\mathcal{F}_{\text{CRS}}$ and uses pk_{CRS} in $\mathcal{F}_{\text{CRS}}$ as generated by $\mathcal{S}$ in l. 3.

```
1: Initialisation:
2:     drained ← $\underbrace{(0, \ldots, 0)}_{\lceil |\mathcal{P}|/B \rceil}$ // 1 entry for each possible batch
3:     $(pk_{\text{CRS}}, sk_{\text{CRS}}) \xleftarrow{\$} \text{KEYGEN}()$

4: On (FORWARD, $P$, $M$):
5:     send $M$, addressed from $P$, to internal $\mathcal{A}$

6: On $M$ for corrupted party $P$ by $\mathcal{A}$:
7:     if $M = (\text{VOTE}, r, \langle c^s_k, c^{\text{CRS}}_k, P_k \rangle_{k \in [n]})$ then
8:         send $(\text{VERIFYVOTE}, P, pk_{\text{CRS}}, \langle c^s_k, c^{\text{CRS}}_k, P_k \rangle_{k \in [n]})$ to $\mathcal{F}_{\text{proof}}$

9:         if reply by $\mathcal{F}_{\text{proof}}$ is (OK) then
10:            send $(\text{VOTED}, r, \langle P_k \rangle_{k \in [n]})$ to $\mathcal{F}^{B,n,t}_{\text{vote}}$ and expect reply
   (OK)
11:            if this vote completes the $j$-th batch then
12:                assign number of corrupted decryptors in $j$-th batch
   to drained$_j$
13:            end if
14:
   $v_P \leftarrow \text{RECONSTRUCT}(\text{DEC}(sk_{\text{CRS}}, c^{\text{CRS}}_1), \ldots, \text{DEC}(sk_{\text{CRS}}, c^{\text{CRS}}_t))$
15:        end if
16:    else if $M = (\text{BATCHSHARE}, j, S)$ then
17:        send $(\text{VERIFYSHARE}, P, pk_{\text{CRS}}, j, S)$ to $\mathcal{F}_{\text{proof}}$
18:        if reply by $\mathcal{F}_{\text{proof}}$ is (OK) then
19:            send $(\text{DRAINED}, j, P)$ to $\mathcal{F}^{B,n,t}_{\text{vote}}$ and expect reply (OK)
20:        end if
21:    end if
22:    send $(\text{FORWARD}, P, M)$ to $\mathcal{F}^{B,n,t}_{\text{vote}}$

23: On (INIT, $P$):
24:     send (INIT) to $\mathcal{G}_{\text{PKI}}$ as $P$ and assign reply to $pk_P$ // "as $P$"
   messages are implicitly routed through $\mathcal{F}^{B,n,t}_{\text{vote}}$
25:     reply $pk_P$ to $\mathcal{F}^{B,n,t}_{\text{vote}}$
```

```
26: On (getDecryptors, $P$):
27:     send (getDecryptors) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ as $P$ and forward reply to $\mathcal{F}^{B,n,t}_{\text{vote}}$

28: On (voted, $P$, $\langle P_k \rangle_{k\in[n]}$):
29:     for $k$ from 1 to $n$ do
30:         send (pk, $P_k$) to $\mathcal{G}_{\text{PKI}}$ and assign reply to $pk_k$
31:         $c^s_k \leftarrow \text{enc}(pk_k, 0)$
32:         $c^{\text{CRS}}_k \leftarrow \text{enc}(pk_{\text{CRS}}, 0)$
33:         send (proveVote, $\bot, \bot, \bot, \bot, \langle c^s_k, c^{\text{CRS}}_k, P_k \rangle_{k\in[n]}, sk_{\text{CRS}}$) to $\mathcal{F}_{\text{proof}}$ and expect reply (ok)
34:     end for
35:     if this vote completes the $j$-th batch then
36:         assign number of corrupted decryptors in $j$-th batch to $\texttt{drained}_j$
37:     end if
38:     send (vote, $r$, $\langle c^s_k, c^{\text{CRS}}_k, P_k \rangle_{k\in[n]}$) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ as $P$

39: On (getMaliciousVotes, $j$):
40:     send (read) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ and keep votes from reply
41:     calculate the $j$-th batch $\mathcal{B}_j$
42:     reply with sum of votes $v_P$ of all corrupted voters in $\mathcal{B}_j$

43: On (batchShare, $P$, $j$):
44:     if all voters of the $j$-th batch are corrupted then
45:         execute ll. 34-35 and 39-42 of Fig. 8 as $P$ // copy actions of $\Pi^{B,n,t}_{\text{vote}}$
46:         reply (batchShare, $j$, $S$) // use $S$ as calculated in l. 45
47:     end if
48:     find index $k$ of decryptor $P$ in $j$-th batch
49:     $\texttt{drained}_j \leftarrow \texttt{drained}_j + 1$
50:     if $\texttt{drained}_j < t$ then
51:         $s_{P,j} \xleftarrow{\$} \mathcal{R}_{\text{SSS}}$ // $\mathcal{R}_{\text{SSS}}$ is the finite field of Shamir's Secret Sharing
52:         $S \leftarrow (k, s_{P,j})$
53:         send (proveShare, $j$, $S$, $pk_{\text{CRS}}$, $sk_{\text{CRS}}$) to $\mathcal{F}_{\text{proof}}$ as $P$ and expect reply (ok)
54:         reply (batchShare, $j$, $S$)
55:     else if $\texttt{drained}_j = t$ then
56:         send (getHonestVotes, $j$) to $\mathcal{F}^{B,n,t}_{\text{vote}}$, add to the reply the sum of corrupted votes in the $j$-th batch (as stored in l. 14) and assign result to $V_j$
57:         send (read) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$, from the reply extract the $t-1$ aggregate shares of the $j$-th batch and assign them to $S_1, \dots, S_{t-1}$
58:         $(S_t, Q_j) \leftarrow \text{findShare}(S_1, \dots, S_{t-1}, k, V_j)$
59:         send (proveShare, $j$, $S_t$, $pk_{\text{CRS}}$, $sk_{\text{CRS}}$) to $\mathcal{F}_{\text{proof}}$ as $P$ and expect reply (ok)
60:         reply (batchShare, $j$, $S_t$)
61:     else // $\texttt{drained}_j > t$
62:         $S \leftarrow (k, Q_j(k))$
63:         send (proveShare, $j$, $S$, $pk_{\text{CRS}}$, $sk_{\text{CRS}}$) to $\mathcal{F}_{\text{proof}}$ as $P$ and expect reply (ok)
64:         send (batchShare, $j$, $S$) to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ as $P$
65:     end if
```

Figure 10

The contents of messages that involve corrupted parties are fully controlled by the internal, simulated adversary $\mathcal{A}$. Simulator $\mathcal{S}$ (Fig. 10) extracts some data from some of these messages (Fig. 10, l. 12) and reads information from $\mathcal{F}_{\text{proof}}$ without changing its state (Fig. 10, ll. 8 and 17). $\mathcal{S}$ also informs $\mathcal{F}^{B,n,t}_{\text{vote}}$ whenever a corrupted player successfully votes (Fig. 10, l. 10) or drains a batch (Fig. 10, l. 19), with which information $\mathcal{F}^{B,n,t}_{\text{vote}}$ decides when a batch is fully drained and thus whether it can disclose the aggregate votes for this share to $\mathcal{S}$. As we will see later, this information is needed to ensure indistinguishability and $\mathcal{F}^{B,n,t}_{\text{vote}}$ could anyway obtain it reading $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ directly. Thus the way $\mathcal{S}$ handles corrupted players' messages cannot provide a distinguishing advantage.

In the real world, an honest party P that receives (init) sends (init) to $\mathcal{G}_{\text{PKI}}$ and forwards its reply pk to $\mathcal{E}$. In the ideal world, the $\mathcal{F}^{B,n,t}_{\text{vote}}$ notifies $\mathcal{S}$ that P received (init). $\mathcal{S}$ in turn sends (init) as P to $\mathcal{G}_{\text{PKI}}$ and forwards its reply to $\mathcal{F}^{B,n,t}_{\text{vote}}$, which stores and forwards it to $\mathcal{E}$. Since in both cases the response is generated by $\mathcal{G}_{\text{PKI}}$ with the same input by the same player, the reply received by $\mathcal{E}$ is perfectly indistinguishable in the two cases.

In the real world, an honest party that receives (vote, v) first gets the decryptors from $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$ and then shares its vote and encrypts each share for each decryptor and with the CRS public key, proves correct ciphertext generation to $\mathcal{F}_{\text{proof}}$ and submits the vote to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$. In the ideal world, an honest party that receives (vote, v) allows $\mathcal{S}$ to generate and send the vote to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$. $\mathcal{S}$ chooses decryptors by asking $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$, as in the real world. This means that $\mathcal{F}^{B,n,t}_{\text{vote}}$ will not trigger the halt of Fig. 2, l. 16, as the local variable `votes` of $\mathcal{F}^{B,n,t}_{\text{vote}}$ only contains honestly generated votes, the same holds for voted messages by $\mathcal{S}$ and every honestly generated vote has a unique round. The same argument precludes the triggering of the halt of Fig. 2, l. 21. Also the halt of Fig. 2, l. 17 will only trigger with negligible probability by virtue of Theorem 9.2. Since it doesn't learn the plaintext vote, $\mathcal{S}$ then generates ciphertexts of 0 and proceeds to fake a proof of correctness of these, exploiting its knowledge of the secret corresponding to the CRS. It then sends the generated vote to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$. The round and decryptors of the vote are generated exactly as in the real world, so they are perfectly indistinguishable. The ciphertexts' distribution is negligibly close to uniform independent of the input to enc due to the IND-CPA property of the encryption scheme so the ciphertexts are indistinguishable. Lastly, in both worlds $\mathcal{F}_{\text{proof}}$ verifies the same statement, namely that the prover either knows the secret corresponding to the key that the prover provided as CRS (possible in the real world only if the prover maliciously sets its own key as CRS, which is irrelevant for honest verifiers who only accept proofs made with the pk_{CRS} given by $\mathcal{F}_{\text{CRS}}$) or that the prover has generated the ciphertexts honestly and in both worlds the proofs are correct. Therefore the state changes caused by a (vote, v) message to an honest party are indistinguishable between the two worlds.

In the real world, an honest party that receives (drainBatch) reads all published votes, deterministically chooses a completed batch for which it is a decryptor (if any), sums the ciphertexts encrypted for itself, decrypts the sum to obtain one aggregate share, proves correct decryption and sends the share to $\mathcal{G}^{\mathcal{P}}_{\text{VoteBox}}$. In the ideal world, when an honest party receives (DRAINBATCH), $\mathcal{F}_{\text{vote}}^{B,n,t}$ is aware of the same votes that it would know in the real world, since $\mathcal{F}_{\text{vote}}^{B,n,t}$ knows all honest votes by virtue of the dummy parties relaying them, as well as all malicious votes' rounds and decryptors since $\mathcal{S}$ sends a VOTED message with such data whenever a malicious player votes correctly. $\mathcal{F}_{\text{vote}}^{B,n,t}$ can therefore proceed to calculate the same batches and deterministically find the same batch for the party to drain as in the real world, if any. In case this drain is the t-th for this batch, $\mathcal{F}_{\text{vote}}^{B,n,t}$ asks $\mathcal{S}$ for the sum of malicious votes for this batch and updates the total vote accordingly. It finally asks $\mathcal{S}$ to form a valid share, which in turn does so and sends it to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$.

$\mathcal{S}$ produces the share in the following fashion. In case all voters of this batch are malicious, they have produced all shares and ciphertexts correctly (otherwise their vote would have been ignored by $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$), so the ciphertexts correspond to the actual votes of the malicious players. In this case $\mathcal{S}$ follows the same steps that an honest decryptor would follow in the real world, therefore introducing no opportunity for distinguishability. In case however at least one voter is honest, $\mathcal{S}$ had faked its ciphertexts when the former voted. In order to generate the needed share now, $\mathcal{S}$ exploits the fact that it is free to decrypt the sum of the honest player's ciphertexts to any share, as long as its x component is equal to the index of the player in the batch's decryptors and fake a proof of correct decryption, by virtue of knowing the secret of the CRS. If the sum of total corrupted decryptors in this batch plus the number of honest decryptors that have drained this batch (including the current one) is less than t (call this number d), then the current drain will not contribute the t-th share for this batch, therefore it will not add anything to any party's existing knowledge. Also no party knows the value that the aggregate share would have in the real world, since at least one of the voters is honest and thus the shares that it encrypted for honest decryptors are never published. Note also that there is at least one honest decryptor in this batch, as $\mathcal{S}$ receives the current message, BATCHSHARE, by $\mathcal{F}_{\text{vote}}^{B,n,t}$ only when an honest decryptor drains a batch. Therefore $\mathcal{S}$ generates a random y value for the share, uses the secret part of the CRS to fake a proof of correct share and sends it to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$.

In case $d = t$, the share that will be created by $\mathcal{S}$ has to be such that reconstructing the t shares returns an aggregate vote matching that of the real world. To that end, $\mathcal{S}$ learns the sum of honest voters' votes in this batch by $\mathcal{F}_{\text{vote}}^{B,n,t}$ and adds the corrupted voters' votes in this batch to get the aggregate batch vote. Note that having $\mathcal{F}_{\text{vote}}^{B,n,t}$ disclose this sum does not leak any more information that what a malicious party in the real world could deduce after the aggregate share currently under generation would be made public. It then uses the aggregate vote and the $t-1$ existing batch shares to solve a linear equation and obtain the y value of the new share, such that reconstruction leads to the correct value. It also calculates the polynomial that corresponds to the aggregate shares for future use (Fig. 9). It then fakes a proof of correct share generation and sends it to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$. In this way, $\mathcal{S}$ has managed to only change data that cannot be known by malicious parties in the real world and keep all other data consistent between the two worlds.

In case $d > t$, $\mathcal{S}$ simply uses the previously determined polynomial to generate a new aggregate share that can be combined with the existing ones to correctly reconstruct the aggregate vote. It then fakes a proof of correct share generation and sends it to $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$.

In the real world, an honest party that receives (READ) reads all aggregate shares from $\mathcal{G}_{\text{VoteBox}}^{\mathcal{P}}$ and for each batch that has at least t such shares, the party reconstructs the aggregate vote and adds it to the sum. Once done, it returns the total. In the ideal world, $\mathcal{F}_{\text{vote}}^{B,n,t}$ simply returns the results as it has stored them locally. These two values necessarily coincide, since $\mathcal{F}_{\text{vote}}^{B,n,t}$ only updates the results once for each batch, exactly when the latter can be reconstructed for the first time. $\mathcal{F}_{\text{vote}}^{B,n,t}$ knows the honest votes of this batch, as it receives them from the honest dummy parties, and the correct malicious votes are given honestly by $\mathcal{S}$. Therefore the two worlds are indistinguishable. □

9.1 Public Verifiability

We here note that in no part of the security proof did we put any specific bounds on the number of corrupted players except, in the interests of privacy, to require that no set of decryptors contained t or more malicious players, a property enforced by $\mathcal{F}_{\text{vote}}^{B,n,t}$ (Fig. 2, l. 17). The indistinguishability argument would work even if this property were not enforced, allowing for a truly arbitrary number of corruptions. Furthermore, by simple inspection of $\mathcal{F}_{\text{vote}}^{B,n,t}$, we can see that honest votes are counted correctly once in an opened batch, irrespective of the number of corrupted parties. This means that our construction enjoys public verifiability, i.e. it has the property that, informally, any observer (whether it participated in the process or not) can check that the votes of other parties that belong to fully opened batches have not been ignored or counted wrongly.

9.2 Multi-Topic Continuous Opinion Aggregation Guarantees

As discussed earlier, a multi-topic continuous opinion aggregation scheme consists of one independent execution of $\mathcal{F}_{\text{vote}}^{B,n,t}$ per topic. Both the *batch privacy* and the *liveness* properties hold for each $\mathcal{F}_{\text{vote}}^{B,n,t}$ execution and these executions do not share any state, therefore the adversary cannot leverage information from one execution to break the guarantees of another. Thus the properties carry over to each vote batch of the combined scheme. Regarding security, Theorem 9.4 ensures that each $\mathcal{F}_{\text{vote}}^{B,n,t}$ instance can be realised by a $\Pi_{\text{vote}}^{B,n,t}$ protocol, as long as no state is shared, e.g. parties must use independently generated keys for each $\Pi_{\text{vote}}^{B,n,t}$ execution. It can be however proven that privacy, liveness and security also hold if parties reuse public keys across $\Pi_{\text{vote}}^{B,n,t}$ executions for efficiency.

10 RECOMMENDED PARAMETERS

Theorem 9.2 provides a valuable result regarding privacy in the asymptotic case, i.e. as n increases to infinity. In practice though it is useful to calculate the probability of a bad event – namely that a uniformly random set of decryptors has t or more malicious parties, in which case they would be able to reconstruct the corresponding vote directly, i.e. without waiting for the completion of a batch, and thus break privacy – for specific values of the system parameters n, t, s and number of total players $|\mathcal{P}|$. In particular, when choosing a uniformly random n-sized subset of $\mathcal{P}$, the probability of obtaining a decryptor set with at least t malicious parties is given by the expression $P[\text{bad event}] = \frac{\sum_{i=t}^{\min(s,n)} \binom{s}{i}\binom{|\mathcal{P}|-s}{n-i}}{\binom{|\mathcal{P}|}{n}}$ and the probability of one or more bad events occurring throughout an entire execution is upper bounded by $\lceil \frac{|\mathcal{P}|}{B} \rceil \Pr[\text{bad event}]$. We visualised the value of this probability for various values of the parameters when $|\mathcal{P}| = 10000$ (Fig. 11). [1]. We also determined that the bad event only happens one or more times throughout an execution with probability less than 0.001 for $|\mathcal{P}| \geq 8200$, $n = \lfloor \frac{|\mathcal{P}|}{100} \rfloor$, $s = \lfloor \frac{|\mathcal{P}|}{4} \rfloor$ and $t = \lfloor \frac{n}{2} \rfloor$. The provided code can be used to plot and calculate exactly these probabilities for an expected number of users in order to choose suitable parameters.

Regarding the batch size B, we recommend a value of 11, which is the size of the anonymity set of each Monero [46] transaction[2]. This recommendation should however be taken with some reservation, as a higher value may be needed due to the increased privacy requirements of votes as opposed to coins, or a lower value may be preferred if batch completion is too slow. Experimentation with a realistic implementation would be needed to specify the best value for B, if any. Furthermore, having B be a divisor of $|\mathcal{P}|$ ensures that no incomplete batch remains at the end of the execution, therefore that all votes are considered.

11 FUTURE WORK

A number of directions for further exploration are discussed next. One possible enhancement to the scheme would be to instantiate a single, efficient protocol for continuous opinion aggregation, avoiding the need to execute a separate copy of $\mathcal{F}_{\text{vote}}^{B,n,t}$ per topic. To achieve this improvement, a way to extend the additive-homomorphic properties of the secret sharing and the encrpytion schemes from a single, scalar integer to a vector of integers would be needed. A simple trick would be to multiply the vote of each topic with an increasing power of 2, such that each topic has enough bits between each power of 2 and that of the next topic to represent all practically plausible results, and add the votes together to form a single, albeit longer, integer. This would however introduce the limitation that each party would need to vote for all topics at once. Lifting this limitation, possibly employing a range proof for each vote to ensure its topic does not clash with that of a previously submitted vote, would be of interest.

Two further goals regarding the number and availability of players are firstly allowing dynamic participation, i.e. not requiring the set of all players to be known from the onset of the protocol – which, among others, needs a method to eliminate the initialisation phase – and secondly allowing participants to vote again in case their batch is not drained by at least t decryptors after a set amount of time (invalidating their old vote in the process), thus further enhancing liveness. Achieving these two targets while maintaining good performance properties may also require employing a reputation system that rewards well-behaving, available parties and punishes misbehaving ones.

Furthermore, the current protocol could be improved by making the security model more robust. There are two such directions: on the one hand dynamic corruptions could be permitted, which would

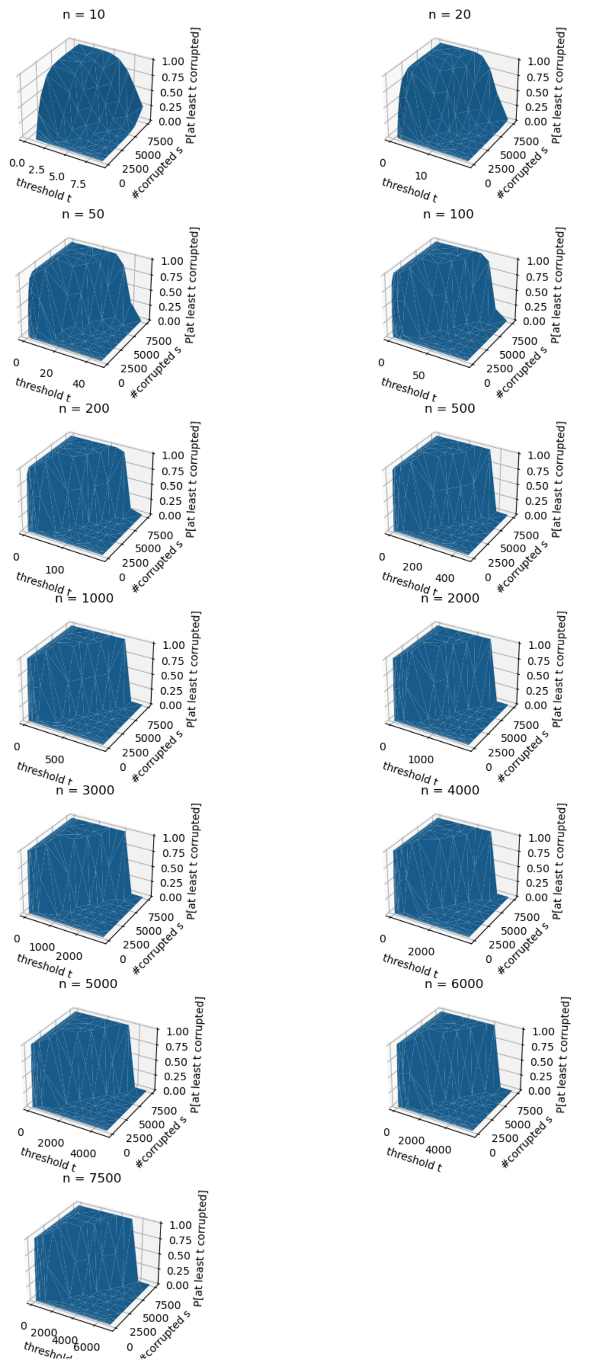

Figure 11: Visualisation of probabilities of bad event for $|\mathcal{P}| = 10000$ and various parameter values

[1]https://gitlab.com/anonymized-submission/democratic-decision-making-parameters
[2]https://www.getmonero.org/resources/moneropedia/ring-size.html

likely need to make the order of votes and/or the corresponding sets of decryptors less predictable. On the other hand, proving security in a framework that supports composition, such as Universal Composition [11] or Constructive Cryptorgraphy [42], would enable composing this protocol with other, independent protocols that either run concurrently in the same machines or use our protocol as a subroutine.

12 CONCLUSION

In this work we formally defined and analysed the privacy and timely progress of a novel decentralised voting scheme for opinion aggregation. We leveraged a variety of underlying primitives to create a protocol that enables a large number of parties to express their opinion on a particular subject while maintaining their privacy within an anonymity set of a given size. Votes are revealed in batches that only disclose a single number, the votes' summary opinion. Our simulations showed that tuning the parameters allows for a variety of tradeoffs between, among others, the speed with which vote batches are revealed and the size of the anonymity sets. Our scheme may be extended to incorporate a variety of features, such as parallel voting of multiple subjects and an unbounded number of players.

REFERENCES

[1] Zeinab Abbassi, Nidhi Hegde, and Laurent Massoulié. 2014. Distributed content curation on the Web. *ACM Transactions on Internet Technology (TOIT)* 14, 2-3 (2014), 9.

[2] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*. USENIX Association, Renton, WA, 335–348. http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf

[3] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2013. Steering user behavior with badges. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, International World Wide Web Conferences Steering Committee / ACM, no address given, 95–106.

[4] Georgios Askalidis and Greg Stoddard. 2013. A theoretical analysis of crowd-sourced content curation. In *The 3rd Workshop on Social Computing and User Generated Content*, Vol. 16. no publisher given, no address given.

[5] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2020. Crowd Verifiable Zero-Knowledge and End-to-End Verifiable Multiparty Computation. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*. Springer, Cham, 717–748. https://doi.org/10.1007/978-3-030-64840-4_24

[6] Aritra Banerjee, Michael Clear, and Hitesh Tewari. 2021. zkHawk: Practical Private Smart Contracts from MPC-based Hawk. In *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. IEEE, no address given, 245–248. https://doi.org/10.1109/BRAINS52497.2021.9569822

[7] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. Cryptology ePrint Archive, Report 2014/349. http://eprint.iacr.org/.

[8] Kelly Bergstrom. 2011. "Don't feed the troll": Shutting down debate about community expectations on Reddit.com. *First Monday* 16, 8 (2011).

[9] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, no address given, 947–964. https://doi.org/10.1109/SP40000.2020.00050

[10] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In *Financial Cryptography and Data Security*, Joseph Bonneau and Nadia Heninger (Eds.). Springer International Publishing, Cham, 423–443.

[11] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, no address given, 136–145. https://doi.org/10.1109/SFCS.2001.959888

[12] Manuel M. T. Chakravarty, Roman Kireev, Kenneth MacKenzie, Vanessa McHale, Jann Müller, Alex ander Nemish, Chad Nester, Michael Peyton Jones, Simon Thompson, Rebecca Valentine, and Philip Wadler. 2019. Functional blockchain contracts. https://iohk.io/en/research/library/papers/functional-blockchain-contracts/.

[13] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. 2016. D-DEMOS: A Distributed, End-to-End Verifiable, Internet Voting System. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, no address given, 711–720. https://doi.org/10.1109/ICDCS.2016.56

[14] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. 2016. SoK: Verifiability Notions for E-Voting Protocols. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, no address given, 779–798. https://doi.org/10.1109/SP.2016.52

[15] Véronique Cortier, David Galindo, and Mathieu Turuani. 2018. A Formal Analysis of the Neuchatel e-Voting Protocol. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE Computer Society, no address given, 430–442. https://doi.org/10.1109/EuroSP.2018.00037

[16] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Springer, no address given, 103–118. https://doi.org/10.1007/3-540-69053-0_9

[17] Elizabeth C. Crites, Mary Maller, Sarah Meiklejohn, and Rebekah Mercer. 2020. Reputable List Curation from Decentralized Voting. *Proc. Priv. Enhancing Technol.* 2020, 4 (2020), 297–320. https://doi.org/10.2478/popets-2020-0074

[18] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Springer, no address given, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[19] Anish Das Sarma, Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2010. Ranking mechanisms in twitter-like forums. In *Proceedings of the third ACM international conference on Web search and data mining*. ACM, ACM, no address given, 21–30.

[20] Gianluca Dini. 2003. A secure and available electronic voting service for a large-scale distributed system. *Future Gener. Comput. Syst.* 19, 1 (2003), 69–85. https://doi.org/10.1016/S0167-739X(02)00109-7

[21] T. Elgamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (1985), 469–472. https://doi.org/10.1109/TIT.1985.1057074

[22] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. 1992. A Practical Secret Voting Scheme for Large Scale Elections. In *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*. Springer, no address given, 244–251. https://doi.org/10.1007/3-540-57220-1_66

[23] Arpita Ghosh and Preston McAfee. 2011. Incentivizing high-quality user-generated content. In *Proceedings of the 20th international conference on World wide web*. ACM, ACM, no address given, 137–146.

[24] Jens Groth. 2004. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *International Conference on Financial Cryptography*. Springer, Springer, no address given, 90–104.

[25] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Springer, no address given, 305–326. https://doi.org/10.1007/978-3-662-49896-5_11

[26] Mangesh Gupte, MohammadTaghi Hajiaghayi, Lu Han, Liviu Iftode, Pravin Shankar, and Raluca M Ursu. 2009. News posting by strategic users in a social network. In *International Workshop on Internet and Network Economics*. Springer, Springer, no address given, 632–639.

[27] Rolf Haenni, Eric Dubuis, Reto E. Koenig, and Philipp Locher. 2020. CHVote: Sixteen Best Practices and Lessons Learned. In *Electronic Voting - 5th International Joint Conference, E-Vote-ID 2020, Bregenz, Austria, October 6-9, 2020, Proceedings*. Springer, no address given, 95–111. https://doi.org/10.1007/978-3-030-60347-2_7

[28] Runchao Han, Jiangshan Yu, and Haoyu Lin. 2020. RandChain: Decentralised Randomness Beacon from Sequential Proof-of-Work. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1033. https://eprint.iacr.org/2020/1033

[29] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. 2009. VeryVote: A Voter Verifiable Code Voting System. In *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*. Springer, no address given, 106–121. https://doi.org/10.1007/978-3-642-04135-8_7

[30] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. ACM, no address given, 61–70. https://doi.org/10.1145/1102199.1102213

[31] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th USENIX*

Security Symposium (USENIX Security 18). USENIX Association, Baltimore, MD, 1353–1370. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[32] Andreas M Kaplan and Michael Haenlein. 2010. Users of the world, unite! The challenges and opportunities of Social Media. *Business horizons* 53, 1 (2010), 59–68.

[33] Chris Karlof, Naveen Sastry, and David A. Wagner. 2005. Cryptographic Voting Protocols: A Systems Perspective. In *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. USENIX Association, no address given. https://www.usenix.org/conference/14th-usenix-security-symposium/cryptographic-voting-protocols-systems-perspective

[34] Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. 2021. KACHINA - Foundations of Private Smart Contracts. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, no address given, 1–16. https://doi.org/10.1109/CSF51468.2021.00002

[35] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. 2018. On the Security Properties of e-Voting Bulletin Boards. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. Springer, no address given, 505–523. https://doi.org/10.1007/978-3-319-98113-0_27

[36] Aggelos Kiayias, Benjamin Livshits, Andrés Monteoliva Mosteiro, and Orfeas Stefanos Thyfronitis Litos. 2019. A Puff of Steem: Security Analysis of Decentralized Content Curation. In *International Conference on Blockchain Economics, Security and Protocols, Tokenomics 2019, May 6-7, 2019, Paris, France*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, no address given, 3:1–3:21. https://doi.org/10.4230/OASIcs.Tokenomics.2019.3

[37] Aggelos Kiayias and Moti Yung. 2002. Self-tallying Elections and Perfect Ballot Secrecy. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*. Springer, no address given, 141–158. https://doi.org/10.1007/3-540-45664-3_10

[38] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2017. An Efficient E2E Verifiable E-voting System without Setup Assumptions. *IEEE Secur. Priv.* 15, 3 (2017), 14–23. https://doi.org/10.1109/MSP.2017.71

[39] Dor Konforty, Yuval Adam, Daniel Estrada, and Lucius Gregory Meredith. 2015. Synereo: The Decentralized and Distributed Social Network. *Self-published* 0 (2015). https://pdfs.semanticscholar.org/253c/c4744e6b2b87f88e46188fe527982b19542e.pdf Accessed: 2019-01-02.

[40] Oksana Kulyk, Stephan Neumann, Melanie Volkamer, Christian Feier, and Thorben Koster. 2014. Electronic voting with fully distributed trust and maximized flexibility regarding ballot design. In *2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*. IEEE, IEEE, no address given, 1–10.

[41] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. Vol. 24. Springer, no address given, 277–346. https://doi.org/10.1007/978-3-319-57048-8_6

[42] Ueli Maurer. 2011. Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In *Theory of Security and Applications - Joint Workshop, TOSCA 2011, Saarbrücken, Germany, March 31 - April 1, 2011, Revised Selected Papers*. Springer, no address given, 33–56. https://doi.org/10.1007/978-3-642-27375-9_3

[43] Avner May, Augustin Chaintreau, Nitish Korula, and Silvio Lattanzi. 2014. Filter & follow: How social media foster content curation. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42. ACM, ACM, no address given, 42–55.

[44] Peter Mell, John Kelsey, and James M. Shook. 2017. Cryptocurrency Smart Contracts for Distributed Consensus of Public Randomness. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*. Springer, no address given, 410–425. https://doi.org/10.1007/978-3-319-69084-1_31

[45] Arash Molavi Kakhki, Chloe Kliman-Silver, and Alan Mislove. 2013. Iolaus: Securing online content rating systems. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, International World Wide Web Conferences Steering Committee / ACM, no address given, 919–930.

[46] Shen Noether and Adam Mackenzie. 2016. Ring Confidential Transactions. *Ledger* 1 (2016), 1–18. https://ledgerjournal.org/ojs/index.php/ledger/article/view/34

[47] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–238.

[48] Emilee Rader and Rebecca Gray. 2015. Understanding user beliefs about algorithmic curation in the Facebook news feed. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, ACM, no address given, 173–182.

[49] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. 2016. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Springer, no address given, 176–192. https://doi.org/10.1007/978-3-662-53357-4_12

[50] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *In CRYPTO*. Springer-Verlag, no address given, 148–164.

[51] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176

[52] Matthew Skala. 2013. Hypergeometric tail inequalities: ending the insanity. arXiv:1311.5939 [math.PR]

[53] Katarina Stanoevska-Slabeva, Vittoria Sacco, and Marco Giardina. 2012. Content Curation: a new form of gatewatching for social media?. In *Proceedings of the 12th international symposium on online journalism*. no publisher given, no address given.

[54] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. 2019. Zkay: Specifying and Enforcing Data Privacy in Smart Contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1759–1776. https://doi.org/10.1145/3319535.3363222

[55] Shifeng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. 2017. RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*. Springer, no address given, 456–474. https://doi.org/10.1007/978-3-319-66399-9_25

[56] Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. 2009. Sybil-Resilient Online Content Voting. In *NSDI*, Vol. 9. USENIX Association, no address given, 15–28.

[57] Unknown. 2013. Slido. (2013). https://www.sli.do/.

[58] Unknown. 2018. Steem Whitepaper. (2018). https://steem.io/steem-whitepaper.pdf Accessed: 2019-04-02.

[59] Unknown. 2021. BitClout White Paper. (2021). https://bitcloutresource.com/bitclout-white-paper/.

[60] Bimal Viswanath, Mainack Mondal, Krishna P Gummadi, Alan Mislove, and Ansley Post. 2012. Canal: Scaling social network-based Sybil tolerance schemes. In *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, ACM, no address given, 309–322.

[61] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. (2014). https://ethereum.github.io/yellowpaper/paper.pdf.

[62] Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B Gibbons, and Feng Xiao. 2009. Dsybil: Optimal sybil-resistance for recommendation systems. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, IEEE Computer Society, no address given, 283–298.

[63] Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. 2019. A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, no address given. https://www.ndss-symposium.org/ndss-paper/a-treasury-system-for-cryptocurrencies-enabling-better-collaborative-intelligence/

[64] Yingwu Zhu. 2010. Measurement and analysis of an online content voting network: a case study of Digg. In *Proceedings of the 19th international conference on World wide web*. ACM, ACM, no address given, 1039–1048.

[65] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *CoRR* abs/1506.03471 (2015). arXiv:1506.03471 http://arxiv.org/abs/1506.03471