PeaceFounder: centralised E2E verifiable evoting via pseudonym braiding and history trees

Janis Erdmanis[0000−0002−8963−5963]

janiserdmanis@protonmail.org

Abstract. PeaceFounder is a centralised E2E verifiable e-voting system that leverages pseudonym braiding and history trees. The immutability of the bulletin board is maintained replication-free by voter's client devices with locally stored consistency-proof chains. Meanwhile, pseudonym braiding done via an exponentiation mix before the vote allows anonymisation to be transactional with a single braider at a time. In contrast to existing E2E verifiable e-voting systems, it is much easier to deploy as the system is fully centralised, free from threshold decryption ceremonies, trusted setup phases and bulletin board replication. Furthermore, the body of a vote is signed with a braided pseudonym, enabling unlimited ballot types.

Keywords: e-voting · E2E-V · anonymous channel · ElGamal · exponentiation mix · braiding · history tree

Introduction

End-to-end (E2E) verifiability and vote privacy are universally recognised as essential requirements for e-voting systems to ensure democratic election results [12,13,2,20]. E2E verifiability allows voters to confirm that their votes are correctly recorded and counted without trusting entities out of their control while maintaining their vote privacy[1]. To break the link between the vote and the voter, E2E verifiable voting systems typically use reencryption shuffle with known examples such as Helios [1], CHVote [16], Belenios [5] or homomorphic tallying such as ElectionGuard [21], many of which are available under open-source licenses. When votes are collected in those systems at the end of the vote, they undergo a reencryption shuffle or homomorphic tallying and end with a threshold decryption ceremony to arrive at the final tally while allowing voters to track their encrypted votes and the public to verify the final tally via robust zero-knowledge proofs [28,29,17].

Despite their security benefits, deploying E2E verifiable e-voting systems is challenging. The threshold decryption ceremony, crucial for maintaining privacy, requires multiple independent parties to participate. This introduces logistical

[1] PeaceFounder is strictly E2E verifiable only after the votes are published on the bulletin board; before that, any delegated auditor can audit evidence internally, and hence, the trust is in control of voters even then (see Table 1)

complexities: if the threshold is too low, it risks leaking the decryption key, allowing the adversary to see how each voter voted. However, if the threshold is large, a corrupt minority could sabotage the decryption of the election results. Additionally, these systems typically assume the presence of an immutable bulletin board to prevent the discarding of encrypted votes from undesirable voters, and they rely on internal eligibility audits to ensure participation privacy and receipt-freeness. These technical intricacies make deploying E2E verifiable e-voting systems less feasible for small and medium-sized communities, leading to a preference for simpler black box systems [3,14]. Wheras entrusting the voting process to a service provider, who may reduce deployment costs, risks compromising privacy.

The Selene system represents a significant advancement in verifiability [24]. It offers a voter assigned tracking number and displays their votes in plain next to them after the vote. This transparency helps voters feel more confident in the system, as they can see how their vote is counted [30]. The tracking number is not published before the vote and is deniable thus is receipt free. However, it's worth noting that the threshold decryption ceremony still needs to be deployed along with the bulletin board, making it generally suitable only for state-like elections.

Haenni & Spycher proposed a system using verifiable identity exponentiation mixes (braids) to anonymise voters' public keys (identity pseudonyms), eliminating the need for a threshold decryption ceremony [18]. In such a system, the votes are signed with a public key on a braided relative generator (pseudonym) and delivered through an anonymous channel to the ballotbox in plain. The lack of a decryption ceremony ensures that after voter cast their vote, the announcement of the election result depends only on a single entity that collects the votes, making it far more robust than depending on a multiparty ceremony. Furthermore, the braiding is transactional and can be delegated to entirely untrusted parties, enhancing the anonymity threshold of the number of parties that need to be compromised to link the vote to the voter.

However, the benefits of such a system have yet to be realized, as it relies on a trusted bulletin board that does not discard unfavourable votes. The use of replication based buletin board distributed between multiple independent parties, offers only minor improvements over existing end-to-end verifiable systems that rely on a threshold decryption ceremony. Furthermore, despite over a decade of developments [23,22,24], only a single implementation has been attempted with UniVote [15]. However, it relies on a threshold decryption ceremony to ensure the imparity of the bulletin board and, hence, is hard to deploy. This gap has led to the development of a new voting system, PeaceFounder, which addresses many nuances related to usability and deployability.

The innovative approach by PeaceFounder combines pseudonym braiding [18] with a history tree enabled bulletin board [7]. When voters cast their vote, their devices receive inclusion proof of the vote, which can later be verified to be binding to the tally with consistency proof. Having only a few voters who request their device to check the proofs allows for the swift identification of a

corrupt server with publicly verifiable evidence. Thus, the immutability of the bulletin board is guaranteed. To ensure global consistency and avoid split view attacks, the requests are made through an anonymous channel, which is already a present assumption in the voting system by Haenni & Spycher [18]. Thus, once the server has assured that the vote is recorded, there is no way for it to be removed. This allows the system to be fully centralised and, thus, makes it easy to self-host as trusted bulletin board assumption is weakened to its availability, which can be easily accounted for.

The system is designed around organizational voting demes to simplify member registration and coordination of braiders. A deme is an arbitrary organisation where members can be continuously registered or terminated and can vote upon proposals as they become available. Membership termination is achieved by encoding terminated members in a bitmask within a proposal to prevent them from voting at the endpoint, and employing a full braiding reset when trust credit exceeds a threshold, thereby efficiently utilizing braiding resources as demes grow larger. This enables braiding to be done asynchronoulsy with other self-hosted demes worldwide and relaxes their availability requirements, significantly advancing voters' anonimity in the public evidence.

Primitive Review

Braids

PeaceFounder uses Haenni & Spycher's proposed construction to form a verifiable exponentiation mix (braid) from ElGamal reencryption shuffle and proof of decryption [18]. ElGamal reencryption shuffle has been well understood and offers zero-knowledge proofs with the following cryptographic assumptions [28,17]:

- Hardness of discrete logarithm problem (DL) for privacy and soundness;
- Hardness of decisional Diffie-Hellman problem (DDH) for privacy;
- Cryptographically secure hash function.[2]

We shall define braiding as a cryptographic schema that shuffles input pseudonyms $y_i = g^{x_i}$ and exponentiates them with a secret factor s, resulting in output pseudonyms $y'_i = (g^{x_i})^s = (g^s)^{x_i} = h^{x_i}$ on a new relative generator $h = g^s$. Private key owners x_i can use the relative generator h to issue cryptographic signatures with the new pseudonyms without being linked to the input pseudonyms. To ensure integrity, braiding is supported with a zero-knowledge proof, represented as a knot, as shown in Fig. 1.

Let's denote Elgamal reencryption under a public key pk as $\mathsf{Enc}_{pk}(a, b) = (a * g^r, b * pk^r)$ where $a, b, g, pk \in G$ are elements of a cryptographic group for which discrete logarithm (DL) and decisional Diffie-Hellman problem (DDH) are hard, such as modular prime groups or elliptic curves. We can define shuffle for the ElGamal vector. $\{(a_i, b_i)\}$ as as $\mathsf{Shuffle}_{pk}(\{(a_i, b_i)\}) = \Pi(\{\mathsf{Enc}_{pk}(a_i, b_i)\})$

[2] Note that a cryptographically secure hash function is weaker than a random oracle assumption

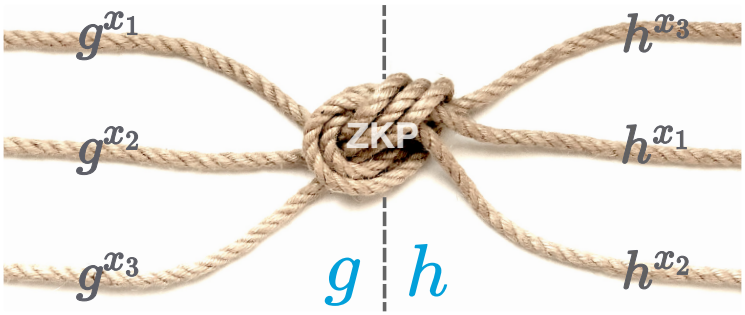

Fig. 1. Illustration of a braid with an associated zero-knowledge proof represented as a knot. Input pseudonyms g^{x_i} are linked to output pseudonyms h^{x_i} via a secret exponentiation factor s via $(g^{x_i})^s = (g^s)^{x_i} = h^{x_i}$ since $h = g^s$.

where, with curly brackets, we denote a vector and Π It's a permutation. In many scenarios, the permutation can be replaced by sorting the output elements, alleviating possible sources of errors.

The braid proof is constructed from zero-knowledge proof of shuffle and proof of decryption. For a vector of members' pseudonyms $\{y_i\}$ as elements from a cryptographic group $y_i \in G$ on a relative generator $g \in G$. The braider computes a vector of output pseudonyms with the following steps:

1. Generates a secret exponentiation factor s;
2. Computes a new relative geenrator $h \leftarrow g^s$;
3. Calculates ElGamal reencryption shuffle on the pseudonym set as $\{(a_i, b_i)\} \leftarrow \mathsf{Shuffle}_h(\{(1, y_i)\})$;
4. Decrypts $c_i \leftarrow b_i^{-s}$;
5. Computes the resulting pseudonyms as $y'_i = a_i / c_i$.

Step 3 is supplemented with a zero-knowledge proof of shuffle [28,17], whereas steps 2 and 4 have proof of correct decryption [4]. Note that in the case of elliptic curves, the identity element can be represented as $(0, 0)$ which is not an element of the curve, and thus, an exception needs to be taken when instantiating it. This procedure was first introduced in the work of Haenni & Spycher. A Verificatum compatible proof of shuffle [29] for step 3 and a custom implementation for proof of decryption in steps 2 and 4 are used, both implemented in Julia and available in the *ShuffleProofs* library [9].

History Trees

History trees, as first proposed by Crosby & Walach [7], an extension to Merkle trees with an unbalanced number of entries, have emerged as a novel solution for ensuring the immutability of bulletin boards. The most prominent examples include transparency logs for public key certificates and the recent enhancement of package distribution for Go programming language libraries [27].

Inclusion proofs provide efficient backtracking hash chain proofs of any record's inclusion in the ledger with respect to the current tree root commit. Clients can get inclusion proofs along with records, which can assure that the record is authentic, much like having a signature issued directly on the record. Consistency proofs, on the other hand, safeguard the ledger's immutability over time. It proves that a current bulletin board commit retains all records from its previous commit, with new records appended. This proof is efficient; thus, multiple clients with unpredictable queries can ensure the ledger's immutability.

However, an attack vector exists where a corrupt authority, capable of tracing clients, can present different views based on the query source, known as a split view attack. To prevent that, the orthodox solution, as proposed by Sigsum [27], is to have a list of witnesses from which clients can ask for signatures on the authority-issued commits to ensure global consistency.

An alternative approach used here is relying on an anonymous channel. Since the server does not know who requests proof, if it tries to cheat, it risks sending an inconsistent ledger commit to the client, who can make them public, resulting in judicial actions. To achieve this, special care must be taken without revealing the client's internal state. Thus, the client with a fresh anonymous session first requests the current chain commit and only then sends its local commit index to retrieve a consistency proof. Note that in more advanced designs, as employed in Sigsum [27,6], the client can directly request necessary hashes from the server and thus also obfuscate commit index for which the proof is retrieved. Therefore, a globally consistent view through an anonymous channel can be achieved with a carefully crafted requests.

The use of anonymous channels to ensure global consistency is preferable as it does not require adding trust assumptions about the honesty of the witnesses. Also, client witness preferences can give away information that can help a corrupt authority to track the voter. In practice, an anonymous channel is never perfect and needs to be considered in relation to a potential adversary's capabilities. Nevertheless, the TOR project [8] has gained substantial popularity to hide in the crowd, and the recent Arti project [26] offers excellent opportunities for custom client application integrations.

The structure of the bulletin board

The bulletin board for PeaceFounder is split into braidchain and ballotbox ledgers as shown in Fig. 2. The braidchain ledger (unrelated to blockchain) contains membership registration certificates, membership termination records, braid records and proposal records. The braidchain ledger is responsible for ensuring correct inputs for membership pseudonyms and generators in braids and an anchoring state within the proposal. The anchored state contains an anchor index, tree root hash, generator, and termination bitmask for honest voters' clients to stop terminated members from voting before the braiding is reset with identity pseudonyms. Records can be included in the braidchain ledger if they are compatible with its current state, which includes:

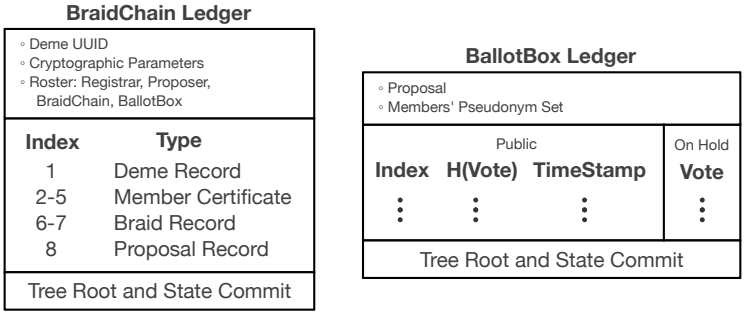

Fig. 2. Ilustration of buletin board structure. On the left, a braidchain ledger is shown. It is initialised with a `DemeSpec` record that sets cryptographic parameters and authorises entities to issue records by putting them in the roster. As new records are added, they undergo a consistency check with the current state. If they are consistent, they are included, and the state is updated and finalised by the braidchain controller issued commit. When the proposal record is included in the braidchain ledger, a corresponding ballotbox ledger is initialised with an anchored member pseudonym set. Any vote that contains a correct proposal hash and is issued with a pseudonym from a member pseudonym set is eligible for inclusion. To ensure fairness and allow revoting, only the vote hash and ballotbox controller commitments are public during the vote, forming a cast receipt that the ballot box controller subsequently commits in the history tree.

- A rooster issued by a guardian that includes authorised registrar, proposer, braider, braidchain, and ballotbox identities, which can be initialised and changed with a `DemeSpec` record;
- A set of member identities;
- A set of blocked identities to prevent repeated registration after termination;
- A set of current member pseudonyms;
- A current relative generator;

The state can be referenced by an index, which is used to anchor a set of member pseudonyms for the proposal that can participate in the vote.

For every recorded proposal in the braidchain, there is a ballotbox ledger initialised with an anchored state generator and member pseudonym set. During the vote, only receipts with vote hash and server commitments are published to ensure fairness and pseudonym secrecy. The history tree is constructed from hashes of those receipts. Thus, the client can be assured about the integrity of the received response with corresponding inclusion proof and check that their vote and votes cast by others are still included by following up with consistency proof later.

BraidChain Ledger

Registration The registration to the deme happens through the recording of a membership certificate in the braidchain ledger. The protocol for registration is as follows:

1. The potential member requests the registrar for an invite, delivered by email or shown on the screen;
2. Upon pasting the invite into the member's device, it retrieves a `DemeSpec` record listing all relevant cryptographic parameters, generates a key pair and sends an identity pseudonym to the registrar with HMAC authorised request;
3. Upon receiving the request, the registrar checks that the one-time token is valid and issues an `Admission` record for the identity pseudonym;
4. The client requests the current braidchain state commit where it gets the current relative generator;
5. Using the current relative generator, the client generates a pseudonym and constructs a `Membership` record containing the pseudonym, generator and `Admission` record and seals it with the identity pseudonym;
6. Upon the braidchain controller receiving the membership record, it verifies that `Admission` record is issued by an authorised registrar, that identity is not in roll or a blacklist, the current generator matches what is encoded in the membership record, and that braiding currently does not occur. If all checks pass, the membership certificate is recorded, and the inclusion proof is returned to the client as confirmation.

All requests within this protocol are idempotent in case of network failures and requests can be replayed and followed up. Steps 4-6 are necessary for a catchup phase, as braiding and membership registration are noncommutative. In case of failure, the client repeats these steps until success unless the user aborts that.

To verify that the intended recipient has used the invite and avoid assuming channel untapability, a document signed by members' digital identity can be used to confirm the registration, which is sent back to the registrar for internal records. The document must contain the registration index at which the membership certificate is recorded in the braidchain ledger and the invite code, which includes the `DemeSpec` hash and the server route. This can be done within a reasonably short time window, instructing the member via email or other communication channels after registration. If the member fails to comply with such a request, the membership can be terminated.

Termination Membership termination is necessary to keep in sync with the organisation or allow the reissuing of credentials if those have been lost or compromised. Terminating membership is a challenging feat with the PeaceFounder system. The members' pseudonyms are entirely anonymous and can't be revoked unless the key owner wants to cooperate, which we assume is not the case. Resetting braiding from identity pseudonyms with a base generator, which allows removing terminated members, faces scalability issues. The probability that some members would need to be terminated grows linearly with the group size. Upon reset, every member loses their anonymity threshold, protecting their privacy, and thus needs to be rebraided. However, relying on the availability of on-demand braiders and coping with a diminished registration experience—where

members may face delays in recording their membership certificates due to braiding locks—is an impractical solution.

An alternative approach involves notifying affected members' clients of their membership termination. If the trust credit, assuming honesty from members' clients, surpasses a specific threshold, a full braid reset is initiated. With this hybrid way, we recognise that the result of the vote is no longer perfect and is subject to an error bar caused by adverse influence, which has attained members' keys. Nevertheless, a small threshold can significantly reduce availability requirements for external braiding and enable more braidings, increasing members' privacy when they vote. Also, in cases where such a compromise is unacceptable, the system can be extended by requiring members to register with certified and tamper-resistant devices/smartcards.

The registrar must inform the clients that they have been terminated in a privacy-preserving manner, and it should not be possible to filter out undesirable news from the registrar. If the client requests the vote on whether their membership is terminated, it will destroy the anonymous channel over which the vote is being cast. Doing that with separate channels would also not be an option, as that produces a strong time correlation.

To resolve this issue, the state anchor in the proposal contains a bitmask of all terminated membership registration indices. If the member's client finds their index within the bitmask, it notifies them that their membership has been terminated. With 1kB of the bitmask, it can support 8192 braidchain records, and employing compression can significantly improve this. As memberships are terminated transparently and can be audited, it leaves no room for a corrupt authority to selectively stop voters from voting on specific proposals. On the other hand, anonymity is protected by not knowing in which bit the client is interested.

The termination for auditing purposes is done through a `Termination` record issued by the registrar and included in the braidchain. It contains a membership index and a terminated identity pseudonym. When recorded, the identity pseudonym is removed from the member identities and added to the blacklist. The latter is essential to prevent the reuse of already-issued admissions for re-registering to the braidchain and to avoid creating duplicate termination records. The termination record is also used to abort the registration protocol when admission is issued, but the membership record is not registered. In such cases, the membership index is set to 0 and does not affect the termination bitmask.

Braiding Braiding is a cryptographic scheme also known as a verifiable exponentiation mix in which input pseudonyms are exponentiated with a secret factor and shuffled. Braiding as input takes pseudonyms, a generator, and cryptographic parameters, produces an output generator and pseudonyms, and ensures integrity with zero-knowledge proofs. The pseudonyms are unlinkable unless a secret exponentiation factor is known or cryptographic assumptions DL or DDH are broken. As such, it does not require client interaction and is executed solely with a single entity.

The anonymity of the member's pseudonym for its cast vote directly depends on the number of parties participating in creating braids. We shall define the anonymity threshold as a measure for each pseudonym, which tells us the least number of parties that must be compromised to uncover the link between the voter and the vote. The measure, though, needs to be looked into in the context of whether parties are independent, and thus, it shows us only the maximum security level for pseudonym anonymity one could expect.

The anonymity threshold for newly registered members is less than that of existing members who have already been braided. This is an acceptable compromise because the benefit for the adversary to infiltrate the system and spy on the members is already significantly reduced. Thus, such an attack would not be cost-effective and hence diminish such risks. Conversely, the authority should do its best to increase the anonymity threshold by attempting to braid with as many different parties as possible.

Because membership registration to the deme happens continuously, with the ability to catch up with the current braidchain generator as described earlier, the availability constraints for braiders are relaxed, allows braiding to occur asynchronously from member registration and proposal submission. This allows for establishing braiding relationships with different parties during the lifetime of the deme, which can be initiated when another party is ready around the globe. To make the process fair and more open, braiding can be done through a trusted third party that acts as a broker, keeping account of each deme braiding contributions, making it easy to obtain a sufficiently large anonymity threshold for members' pseudonyms of the deme.

A braid reset can be done to remove terminated memberships from member pseudonyms that adversaries may be using. For such braid, the input pseudonyms are taken as member identity pseudonyms and the base generator as the input generator. As this operation degrades the anonymity of members, it can only be done in a self-braiding issued by a local braider. After such braid, braiding can be continued with external braiders.

The braidchain controller accepts a braid if input pseudonyms and input generator are equal to the current member pseudonyms and current generator unless reset when member identity pseudonyms and base generator is used instead. Additionally, before the recording, the braid proof is verified with respect to cryptographic parameters with which the deme is initialised. The braid itself is signed by the braider with external cryptographic parameters specified in the external deme `DemeSpec` record provided with the braid. This way, the authenticity and independence of every braid can be evaluated from public records that this external deme publishes, such as a public bulletin board for auditing and some social proofs that assure the authenticity of the deme. Moreover, the braiding offers an opportunity to advertise one's deme and forms a decentralised index.

Proposing A vote is officially announced with a proposal recorded in a braidchain ledger. The proposal record contains metadata such as title, description,

and unique unified identifier (UUID), which can be used to reference proposals in preparation phases/drafts outside the voting system. In addition, UUID can enable the reference of the vote in a ballotbox broker, which can manage vote collection for multiple demes, decoupling it from a particular deme's braidchain ledger.

The proposal also contains an anchor of the braidchain state, which sets the relative generator, member termination bitmask, and braidchain tree root. As votes include a hash of the proposal on which they issue their votes, which includes the braidchain tree root, voters act as witnesses for the braidchain ledger, cementing its immutability. It also enables efficient auditing; knowing the ballotbox tree root announced along with the tally provides all the necessary information to audit the bulletin board records that led to the resulting tally. Furthermore, the tree root is shown on voters' devices, which, in turn, also ensures global consistency of the braidchain ledger for the voters' clients.

Lastly, the proposal contains the opening time, closing time, ballot itself and ballotbox collector identity authorised to issue ballotbox commits. The ballotbox collector identity is authorised with `DemeSpec` record and is included in the `Proposal` record. As members' clients are thin, it enables them to know from which authority they need to wait for confirmation without needing to keep any other records.

An important part of election security is ensuring that every client gets the same list of proposals. One way is to use an anonymous channel and request a list of proposals from the server. An alternative approach is to have clients' devices audit the braidchain using consistency proofs. Consequently, if a corrupt authority attempted a split view attack, it would be unable to reconcile the discrepancies at the ballot box when the tally with the ballot box tree root is published and hence would face judicial actions.

To use such property, all proposals are tied into a linked list. The current braidchain state commit contains the index of the last recorded proposal. The recorded proposal includes an index of the previous one and so on. When retrieving proposals, the client then, along with the proposals, asks for inclusion proofs, and thus, all is assured. To ensure that the last state commit contains an index to the previous proposal, the client keeps a record of the commits, assuring that no new proposal is inserted before them. If it is, then proof of blame can be constructed and announced publicly for everyone to verify.

When a proposal record is submitted to the braidchain, it undergoes several checks: it confirms there are no duplicate UUIDs, verifies that a current `DemeSpec` record authorises the collector's identity, ensures the anchor index points to a braid, checks all state parameters for accuracy, and finally, verifies that an authorised proposer issues the proposal record. Afterwards, the last proposal index in the state commit is updated, and the ballotbox ledger is initialised with the member pseudonyms to which the proposal is anchored.

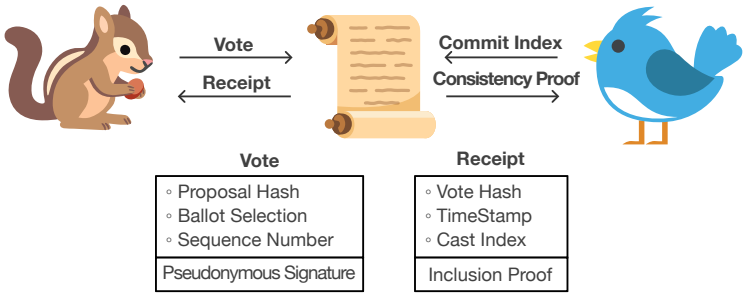

Fig. 3. Ilustration of ballotbox ledger. Every vote signed by a valid pseudonym and associated with a valid proposal hash gets recorded in the ballotbox ledger, even if it is superseded or malformed. Upon recording, a receipt containing an inclusion proof is returned; if the same vote is already recorded, a receipt for it is returned instead. A timestamp ensures that malware cannot show a receipt linked to someone else's vote. Meanwhile, a cast index helps locate the specific vote on the ledger. Few voters make follow-up queries to update their local consistency proof chain which ensures their vote's inclusion as well as votes made by others.

BallotBox Ledger

When the proposal's opening time arrives, it becomes available for vote submissions to the ballotbox ledger as ilustrated in Fig. 3. The vote starts with a ballotbox controller generating a seed and placing it in the state commit that prevents voters from casting a vote before the ballotbox opens. The vote contains a proposal hash to ensure that every voter votes on the same proposal, the seed received in a commit, a sequence number and the voter's selection. A vote is recorded as long as the proposal hash matches and is issued with a valid seal from one of the members' pseudonyms. This holds regardless of whether an earlier vote with a higher sequence number has been recorded or if the selection is filled inconsistently with the ballot.

A cast receipt and inclusion proof in the response ensures that the vote is successfully cast. If the vote is already recorded, an inclusion proof references that instead and hence ensures that the casting of the vote is idempotent. The cast receipt contains a commitment to pseudonym alias, vote hash, timestamp and cast index. A timestamp ensures that malware on the device can't show a receipt linked to someone else's vote. Meanwhile, the cast index helps to locate the specific vote on the ledger deterministically.

The alias is an integer that allows referencing a pseudonym within anchored braid output member pseudonyms. As the clients don't keep the braid record on their devices, the ballotbox controller calculates it for them. Correctness is ensured with a commitment $\mathsf{Hash}(\text{Alias}|\text{Vote})$ included in the receipt, so the pseudonym casting the vote is not leaked. The alias is delivered along a cast receipt in response to the vote submission to the ledger.

To ensure that their vote is cast as intended, a receipt with a tracking code is shown on the voters' client devices as shown in Fig. 4. The tracking code is

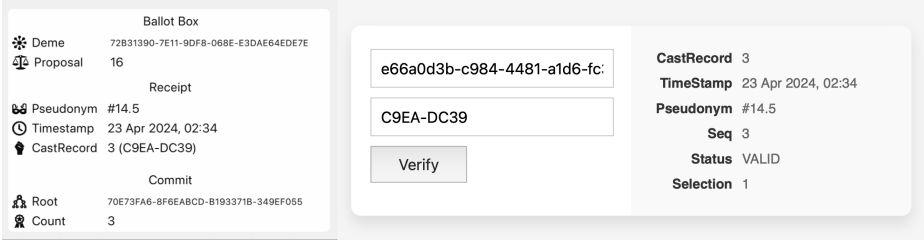

Fig. 4. Ilustration of cast receipt and tracking code. After casting a vote, the voter sees a receipt on their client, as shown on the left, produced by [10]. The receipt contains a pseudonym alias, timestamp, and cast index for the ballotbox ledger row. Commit lists locally stored tree root and ledger index, which gets updated upon refresh with the client requesting consistency proof from the ballot box controller. On the right, a web form where a voter can put in a tracking code and see that their vote is cast as intended is shown, which is used to make HMAC-authorised requests and uses a proxy to ensure untraceability. The tracking code loses valididty after a short period to preserve voters' ability to revote undetected.

temporal and valid for a short period since the vote inclusion in the ballotbox ledger to preserve voters' ability to revote undetected. It is derived by truncating $\mathsf{Hash}(0|Vote)$ and displaying that to the voter in Crockford base32 encoding. To see the cast vote, within the provided time window, the voter can input the tracking code within the corresponding static website form and get the response about the vote using the tracking code as a token. To allow making such requests to unprotected ballotbox servers without TLS, the HTTP requests and responses are HMAC authorised and routed through a proxy to protect voters from a local network observer. A 5-byte tracking code, as shown on the screen, requires around 5TB lookup table, which, if unacceptable in a particular context, can be augmented with the use of TLS.

If a voter is unsatisfied with their cast vote, changes their mind, or is influenced by coercion or bribery, they have the option to revote. A sequence number ensures that only the last cast vote from the device counts. If two votes with the same pseudonym and sequence number are made, the first one recorded at the ballotbox ledger gets counted. Thus, if a vote is cast with a compromised key, then when the voter casts their vote from their device, its status would be overridden when checked with a tracking code.

However, the ability to see the status of a cast vote allows adversaries to submit their coerced votes at the end of the voting process and verify their validity. To prevent this, the vote selection can be asymmetrically encrypted so it is possible to mark a vote as coerced. When checked with the tracking number, such a vote would be shown as valid unless superseded with another later-issued coerced vote.[3]

[3] Details on how the vote selection is encrypted and verifiably decrypted will be outlined in a separate short paper to not convolute existing exposition.

Another method to detect votes cast outside the device is with a tally bitmask that lists counted votes, included in the ballotbox commit when votes are announced. The voter's device then retrieves it with the consistency proof and checks whether their last cast vote has been included in the final tally. Consequently, the client alerts the voter if their cast vote has not been counted.

A history tree consistency proofs with an anonymous channel prevents a corrupt ballotbox controller from returning a valid inclusion proof while dropping the vote from the ledger. After the client gets inclusion proof, it can subsequently request consistency proof with respect to the current braidchain ledger commit at any time. As the requests are made through an anonymous channel, only a few voters need to follow up with consistent proofs to ensure the immutability of the ledger. Furthermore, the official announcement of the tally also includes the ballotbox tree root, which is replicated between different mediums. That further cements global consistency for the voter's clients as anyone can check that the announced tree root agrees with what is shown on their devices. If it differs, a commit can be exported and combined with one referenced in the announced tally as a proof of a corrupt ballotbox controller, and judicial actions can follow.

Such a check does, however, assume that the voter's device is not infected with malware, deceiving voters into believing their vote has been counted. For instance, it could cast a valid vote following the voter's own submission or vote on behalf of the voter if it anticipates that the voter will not participate. The latter can be detected if the malware mispredicts participation and the voter checks the cast vote via tracking number or on the ballotbox when the votes are published. However, using a tracking number falls short when the vote can be indicated as coerced, treating the voter as a coercer. Therefore, both attacks can only be detected if the votes are published on the bulletin board and the voter checks them.

To prevent malware from deceiving voters into checking another person's vote, the receipt contains a timestamp when the vote has been included in the ledger and it's cast index. Voters can verify their own vote by locating it on the public bulletin board using their cast index and comparing the timestamp with the time they cast their vote. The method remains secure even during a widespread, coordinated malware attack, as the malware cannot link multiple voters to a single vote because it cannot influence the timing of when each voter casts their vote. On the other hand, the pseudonym alias allows the voter to locate other cast votes more easily and check if it corresponds to the times when it cast votes.

Unfortunately, voters cannot prove if their votes have been compromised by third-party malware. It is the voter's responsibility to verify their vote and ensure that their devices and software are free from malware, reporting any detected incidents. Certified devices or smartcards, which require a PIN for issuing votes, can mitigate such attack vectors when used with a tracking code, provided the certified hardware is free from backdoors. Privacy from a corrupt vendor can be safeguarded by generating a private key with a trusted observer in a two-party protocol, as outlined in [4]. As client side operations for signing a vote are

straightforward, requiring only a DSA signature on a relative generator listed in a proposal, makes certified tamper resistant hardware viable for larger demes and can be explored with further research.

The final attack vector that a corrupt ballotbox controller can use is manipulating availability and ignoring undesirable votes without returning a response. A network provider can also exploit such an attack if the votes' selections are not encrypted. Also, an adversary anticipating an undesirable tally may perform a DDOS attack on the ballotbox controller, preventing voters from submitting their votes and arriving at the official tally.

To defend against such incidents, voters can cast their votes through proxies or monitors. These intermediaries can verify the eligibility of the vote and log issues to record the vote in the ballotbox ledger. To ensure the effectiveness of these logs, each proxy collaborates with others when issues arise, attempting to form a consensus on unrecordable votes. This consensus can then serve as evidence of corrupt authority, potentially leading to judicial action. The imparity of proxies can be ensured by asymmetrically encrypted vote selection, which will be outlined in a subsequent paper.

Auditing

To audit the announced tally, an auditor must first check that every voter has arrived at the same tree root as the one announced along with the tally. Only a few voters need to be checked, and the sample does not need to be random either because using an anonymous channel ensures a globally consistent view. Alternatively, this check can be performed passively by ensuring that no inconsistent commits have been made post-vote if a few voters have compared their ballotbox tree root on their device with the one announced.

The next step is to check if the ballotbox controller had not purposefully discarded votes from being recorded in the ballotbox ledger. To do so, the auditor needs to be aware of the channels members voice their complaints and the proxy/monitors they would use in case of such incidents. The auditor must verify that the monitor lists any reported incidents and cross-check this information with other monitors that the proxy used to route the vote. This is crucial to prevent a conspiring minority from undermining the trust in the election results.

Finally, the bulletin board can be audited. In the most simple use, it can be run by one command `peacefounder-audit all <buletinboard>` which can be conveniently integrated into continuous integration pipelines [11]. When such command is used, the most recent commit of the ballotbox is taken from which auditing follows in the following order:

1. Checks that the tree root of all ballotbox records corresponds to that of the commit;
2. Checks that the tree root encoded in the proposal anchor corresponds to that of the braidchain at the anchored state index;
3. Checks that every record in the braidchain has been correctly authorised, followed by auditing of the seals;

4. Checks that every braid has correct input pseudonyms and generator;
5. Verifies every braid proof;
6. The braidchain audit concludes by checking that every proposal state anchor is correct and points to a braid record;
7. Checks the eligibility of each vote in the ballotbox and the validity of its seal. This consequently also audits pseudonym aliases as shorthand for locating pseudonyms within the anchored braid output pseudonym vector;
8. Finally, the state, including a tally for the ballotbox, is computed and compared with the tally in the ballotbox commit.

The outcome of such an audit ensures that the auditor only needs to verify that the tree root of the ballotbox commit matches the one announced with the official tally, thereby ensuring the integrity of the vote, with each member having at most one vote. Additionally, the auditor can check that members' pseudonyms used to cast votes on proposals have a high anonymity threshold achieved through braiding among different independent parties, thus ensuring the anonymity of the voter.

The final part of the audit is to verify the authenticity of every member of the deme eligible to vote. Those records are kept with the registrar and are not available publically, as that would violate the right of privacy for freedom of association, which is also necessary to satisfy GDPR policy. Therefore, to audit those records, the registrar needs to provide access to them, which can potentially expose members of the organisation. To limit potential leakage, a verifiably random sample of members can be created using verifiable randomness [25]. Fortunately, this type of audit can be conducted asynchronously from the voting process and reused for multiple proposals, provided the deme's members remain largely unchanged.

Properties

The PeaceFounder voting system offers all key properties for E2E verifiability [2]. Recorded as cast, cast as intended, and tallied as cast are ensured while also enabling public verification of each vote's eligibility without compromising the anonymity of the voters. It also offers a limited form of receipt freeness until votes are publicly available on the bulletin board. That can be delayed to weaken the link between coercer/briber and their subjects.

These various properties are not universal and can be individually compromised when either the voter's client, the bulletin board, the registrar or braiders are controlled by an adversary (see [13] for analogous analysis of existing online voting systems). The properties for a trust also depend on whether votes are already published on the bulletin board, which can be delayed for coercion/bribery prevention purposes. The matrix of properties and involved parties is shown in Table 1.

When a voter casts their vote, the device shows the receipt, which contains the cast index and timestamp when it was recorded. These can be compared with published receipts on the bulletin board. A malicious voter's device may deceive

Property	Client	BBoard	Registrar	Braider
Recorded as cast	🕵	🕵	🕵	🕵
Cast as Intended	✅* 🕵	🕵 ✅*	🕵 🕵	🕵 🕵
Tallied as recorded	✅*	✅*	🕵	🕵
Eligibility	🕵	🕵	✅	🕵
Anonimity	✅	🕵	🕵	🕵 one ✅
Receipt freeness (temporal)	✅	✅	🕵	🕵

Table 1. Conditions under which E2E verifiability properties are ensured for the PeaceFounder voting system. Trusted entities are represented with a green checkmark, whereas the man in a trench coat represents entities an adversary can control while ensuring a particular property. After the votes are published on the bulletin board, voters no longer need to trust entities indicated with an asterisk but lose receipt freeness.

the voter by showing another receipt from the bulletin board when multiple votes are made simultaneously. However, the deception would be exposed if the voter compares the pseudonym alias of the vote with that published on the bulletin board or checked with the tracking code after the cast. Therefore, the property *recorded as cast* is unconditionally satisfied, even during the vote when the cast vote has sufficient time separation from others.

After casting their vote, voters receive a tracking code displayed by their client device. They can enter this code into a web form and retrieve their vote within a 15-minute window from the time it was recorded, as illustrated in Fig. 4, ensuring that their vote is cast as intended and is recorded as cast. Such a check, however, does assume that the client's device is not infected with malware and does not coordinate its efforts with the bulletin board. Thus, either of them needs to be trusted during the vote. After the votes are published on a public bulletin board, they can be verified there, ensuring they are *cast as intended* unconditionally.

A follow-up consistency proof check of a few voters, conducted through anonymous channels on voters' client devices, ensures that their votes and votes cast by others remain cast as recorded. It is reinforced by announcing the final tally along with the ballotbox tree root from which it was computed, easily replicable across various mediums. The tally can be announced before votes are published on the bulletin board. In such cases, third-party internal audits can easily verify the resulting tally, rendering the honesty of the bulletin board optional. Furthermore, risk-limiting tallies can be used to limit leakage [19]. Therefore, *tallied as recorded* holds unconditionally in the presence of trusted third-party internal audits, and once the votes are published on the bulletin board, those audits are no longer necessary.

The eligibility of every vote is guaranteed by a seal issued with a pseudonym, which is listed under the members' pseudonyms in the proposal-anchored braid. Auditing the braidchain ledger, which is always public, assures that there is one

corresponding membership certificate for every pseudonym. The other part is assuring that memberships are issued to authentic members, which can be only done by internal audits to preserve members' right to privacy for freedom of association. The members consent to their association with a digitally signed document containing the invite and their registration index, which is kept with the registrar. Therefore, *eligibility* holds unconditionally as long as internal audits of the registrar are performed.

The voter's anonymity comprises two properties: the unlinkability of the vote to the voter and the untractability of the voter's identity when it casts a vote. Unlinkability is achieved through the use of braided pseudonyms. Since braiding is transactional and independent of registration, it can be conducted with untrusted parties, provided the braid proofs are valid. This significantly broadens the range of entities that can perform braiding. Everyone can verify the authenticity of the braids, which are issued by separate parties and sealed by the braiders, thereby preventing any corrupt authority from falsifying anonymisation. On the other hand, untraceability is assured with the help of an anonymous channel [8]. Therefore, *anonymity* is maintained as long as at least one braider is honest and no adversary has compromised the anonymous channel used by the voter during the vote.

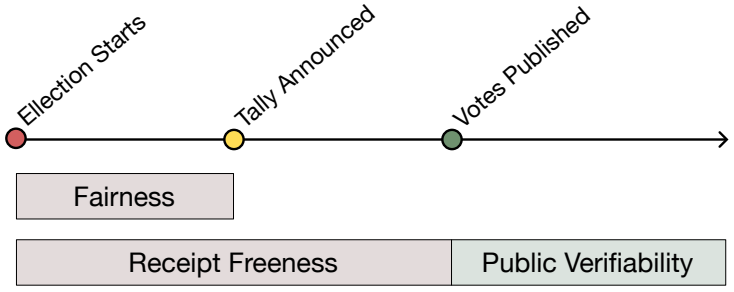

Fig. 5. Illustration of temporal receipt freeness in the Peacefounder voting system. During the election period, the system maintains both receipt-freeness and fairness. However, after the tally is published, newly submitted votes lose their fairness. The votes themselves can be published on the buletin board later to extend the time period for receipt-freeness, reducing the effectiveness of coercers and bribers. This comes at the expense of delaying the audit process for publically verifying that votes have been tallied as cast.

The receipt freeness, ensured by the ability to revote and mark votes as coerced, is temporary and is lost once the votes are published on a public bulletin board as shown in Fig. 5. However, an adversary must compel voters to commit to a specific vote and prove its ownership during the vote. This can be achieved by proxying their vote through them, the voter disclosing the combination of cast index and pseudonym alias, or marking the ballot in a unique way, known

as the Italian attack. If neither of those steps is done, after votes are published, the voter is free to take any receipt from the bulletin board and claim that to be their own to coercer/briber.

A measure that the authority can take is to delay the publishing of the votes and use internal third-party audits to verify the tally. This approach reduces the likelihood of bribery, as voters must trust that bribers will remember to compensate them once the votes are published. It also makes it more difficult for bribers to create network effects during voting. In cases of coercion, delaying the publication gives subjects time to organise and document potential consequences, encouraging them to defy coercers' orders by either marking their votes as coerced or opting to revote later.

Conclusion

This paper introduces an E2E verifiable voting system with centralised responsibility and decentralised accountability via voter devices. Any misconduct results in publishable cryptographic evidence identifying the responsible party - the registrar, proposer, bulletin board, or braider authorities - allowing judicial actions to be taken. The system is fully centralised, enabling a single party to set up, deploy, and collect votes. It does not require bulletin board replication and monitoring for immutability, as this is ensured by voters' devices through consistency proofs.

Privacy is secured through braiding with multiple independent parties. As it is transactional, it extends the scope beyond that of existing systems, which require coordination of threshold decryption ceremonies that depend on a trusted quorum assumption for robustness. Revoting and marking votes as coerced are possible; however, receipt-freeness is temporary and lost once votes are published on the bulletin board. Therefore, resistance to coercion and bribery depends on the assumption that voters will not commit to distant outcomes, ensuring they can freely revote or tag votes as coerced.

In the future, encryption of the vote selections will be addressed so that votes can be marked as coerced, and the imparity of proxies/monitors will be maintained. Proof of participation may be implemented using blind signatures to ensure that eligible members vote, particularly in scenarios where authorities seek to enforce voter participation. Extending the protocol with smartcards is also an exciting avenue to explore to prevent malware interference and ensure that terminated members cannot cast votes. Privacy from a corrupt hardware issuer is already a solved problem via the two-party protocol outlined in [4].

The fact that the vote selections are encoded plainly offers a straight path to support unlimited ballot types, preferential, budget constraint limited, and cardinal ballots. Also, apart from ballots, anonymised pseudonyms can be used to do internal whistleblowing to ensure integrity within the organisation. More daring features can be explored, like fluid voting, where voters can change their vote within given periods, which can make a representative lose their seat, bridging the gap between representative and direct democracy. Or doing ballot shard-

ing where the long ballot is sharded into smaller ones assigned to member pseudonyms in a verifiable lottery would elevate the impact of individual voters' choices, reduce decision fatigue and hence could address the voting paradox. All these features can enable a better democracy through vote while ensuring integrity and voters' privacy in foolproof deployments.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: Proceedings of the 17th Conference on Security Symposium. p. 335–348. SS'08, USENIX Association, USA (2008)
2. Benaloh, J., Rivest, R.L., Ryan, P.Y.A., Stark, P.B., Teague, V., Vora, P.L.: End-to-end verifiability. ArXiv **abs/1504.03778** (2015), https://api.semanticscholar.org/CorpusID:14314175
3. Brunet, J., Essex, A.: Online voting in ontario municipalities: A standards-based review. In: Volkamer, M., Duenas-Cid, D., Rønne, P., Ryan, P.Y.A., Budurushi, J., Kulyk, O., Rodriguez Pérez, A., Spycher-Krivonosova, I. (eds.) Electronic Voting. pp. 52–68. Springer Nature Switzerland, Cham (2023)
4. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) Advances in Cryptology — CRYPTO' 92, vol. 740, pp. 89–105. Springer Berlin Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7, http://link.springer.com/10.1007/3-540-48071-4_7, series Title: Lecture Notes in Computer Science
5. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A simple private and verifiable electronic voting system. In: Guttman, J.D., Landwehr, C.E., Meseguer, J., Pavlovic, D. (eds.) Foundations of Security, Protocols, and Equational Reasoning, vol. 11565, pp. 214–238. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-19052-1_14, http://link.springer.com/10.1007/978-3-030-19052-1_14, series Title: Lecture Notes in Computer Science
6. Cox, R.: Transparent logs for skeptical clients (2019)
7. Crosby, S.A., Wallach, D.S.: Efficient data structures for tamper-evident logging. In: Proceedings of the 18th Conference on USENIX Security Symposium. p. 317–334. SSYM'09, USENIX Association, USA (2009)
8. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation onion router. In: 13th USENIX Security Symposium (USENIX Security 04). USENIX Association, San Diego, CA (Aug 2004)
9. Erdmanis, J.: ShuffleProofs.jl: Verificatum compatible verifier and prover for NIZK proofs of shuffle. https://github.com/PeaceFounder/ShuffleProofs.jl (2022)
10. Erdmanis, J.: PeaceFounder project. https://github.com/PeaceFounder (2024)
11. Erdmanis, J.: PeaceFounderDemo: bulletin board demo audited in continuous integration pipeline. https://github.com/PeaceFounder/PeaceFounderDemo (2024)
12. Finogina, T.: Why does e-voting have to be perfect? (short article) p. 377 (10 2021)
13. Finogina, T., Cucurull Juan, J., Costa, N.: Selective comparison of verifiable online voting systems. Security and Privacy (2024). https://doi.org/https://doi.org/10.1002/spy2.394, https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.394
14. Goodman, N., Spycher-Krivonosova, I., Essex, A., Brunet, J.: Verifiability experiences in ontario's 2022 online elections. In: Volkamer, M., Duenas-Cid, D.,

Rønne, P., Ryan, P.Y.A., Budurushi, J., Kulyk, O., Rodriguez Pérez, A., Spycher-Krivonosova, I. (eds.) Electronic Voting. pp. 87–105. Springer Nature Switzerland, Cham (2023)
15. Haenni, R.: UniVote protocol specification (2013), https://e-voting.bfh.ch/app/download/5874743461/specification.pdf?t=1507600656
16. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: Chvote protocol specification. Cryptology ePrint Archive, Paper 2017/325 (2017), https://eprint.iacr.org/2017/325
17. Haenni, R., Locher, P., Koenig, R., Dubuis, E.: Pseudo-code algorithms for verifiable re-encryption mix-nets. In: Financial Cryptography and Data Security, vol. 10323, pp. 370–384. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-70278-0_23, http://link.springer.com/10.1007/978-3-319-70278-0_23, series Title: Lecture Notes in Computer Science
18. Haenni, R., Spycher, O.: Secure internet voting on limited devices with anonymized dsa public keys. In: Proceedings of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections. p. 8. EVT/WOTE'11, USENIX Association, USA (2011)
19. Jamroga, W., Roenne, P.B., Ryan, P.Y.A., Stark, P.B.: Risk-limiting tallies, http://arxiv.org/abs/1908.04947
20. Jamroga, W., Ryan, P.Y.A., Schneider, S., Schürmann, C., Stark, P.B.: A declaration of software independence. In: Dougherty, D., Meseguer, J., Mödersheim, S.A., Rowe, P. (eds.) Protocols, Strands, and Logic: Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday, pp. 198–217. Lecture Notes in Computer Science, Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-91631-2_11, https://doi.org/10.1007/978-3-030-91631-2_11
21. Josh Benaloh, Michael Naehrig: ElectionGuard: Design specification, https://github.com/microsoft/electionguard/releases/download/v2.0/EG_Spec_2_0.pdf
22. Rakeei, M.A., Giustolisi, R., Lenzini, G.: Secure internet exams despite coercion. In: DPM/CBT@ESORICS (2022), https://api.semanticscholar.org/CorpusID:251066859
23. Ryan, P.Y.A.: Crypto Santa, pp. 543–549. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49301-4_33, https://doi.org/10.1007/978-3-662-49301-4_33
24. Ryan, P.Y.A., Rønne, P.B., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. In: Clark, J., Meiklejohn, S., Ryan, P.Y., Wallach, D., Brenner, M., Rohloff, K. (eds.) Financial Cryptography and Data Security, vol. 9604, pp. 176–192. Springer Berlin Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_12, http://link.springer.com/10.1007/978-3-662-53357-4_12, series Title: Lecture Notes in Computer Science
25. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460. IEEE (2017). https://doi.org/10.1109/SP.2017.45, http://ieeexplore.ieee.org/document/7958592/
26. TOR Project: Arti: An implementation of Tor, in Rust. https://gitlab.torproject.org/tpo/core/arti (2024)

27. Valsorda, F.: Modern transparency logs. In: Proceedings of the Real World Crypto Symposium. International Association for Cryptologic Research, Toronto, Canada (Mar 2024), https://iacr.org/submit/files/slides/2024/rwc/rwc2024/68/slides.pdf
28. Wikström, D.: A sender verifiable mix-net and a new proof of a shuffle. In: Roy, B. (ed.) Advances in Cryptology - ASIACRYPT 2005. pp. 273–292. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
29. Wikstrom, D.: How to implement a stand-alone verifier for the verificatum mix-net. verificatum.org (2011)
30. Zollinger, M.L., Estaji, E., Ryan, P.Y.A., Marky, K.: "just for the sake of transparency": Exploring voter mental models of verifiability. In: Krimmer, R., Volkamer, M., Duenas-Cid, D., Kulyk, O., Rønne, P., Solvak, M., Germann, M. (eds.) Electronic Voting, vol. 12900, pp. 155–170. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-86942-7_11, https://link.springer.com/10.1007/978-3-030-86942-7_11, series Title: Lecture Notes in Computer Science