Itugen: Transparent and Anonymous voting without Tallying Authorities

Guillaume Drevon[1], Vincenzo Iovino[2], and Aleksander Kampa[1]

[1]Sikoba Research
{gd,ak}@sikoba.com
[2]University of Salerno,
vinciovino@gmail.com

6 December 2019

Abstract

We propose itugen, a new verifiable e-voting system that enjoys very strong security guarantees. Our scheme strictly divides ballot allocation from voting, and the voting system ensures both privacy and verifiability. Authorities are trusted only to guarantee the validity and independence of the ballots but cannot break the privacy of any individual voter or subvert the result of the election even if they collude.

Our system is based on isekai, a versatile and powerful implementation of the major recent variants of succinct zero-knowledge proofs including SNARKs, Aurora, Ligero and Bulletproofs. Integrating these mechanisms in a modular way, our e-voting scheme can be instantiated from a number of currently-mature *post-quantum secure* primitives, with the possibility of adding new schemes in the future.

Keywords: e-voting, zero-knowledge, system design.

1 Introduction

E-voting technologies, in particular Internet voting, can help increase voter turnout in elections while reducing the cost of organising them. This could lead to major democratic changes in the coming digital society.

In this paper, we propose and implement itugen, a new e-voting system that we believe to be the first to simultaneously satisfy the following properties:

- **Easy tallying.** The ballots appear on the bulletin board in clear text, so the tally can be easily computed and checked "by hand", as in traditional paper-based elections.
- **Anonymity.** A vote cannot be associated with any of the eligible voters. However, only eligible voters are allowed to vote, and they can cast one and only one valid vote.
- **Individual and universal verifiability.** Each eligible voter can quickly check that their vote was counted. Moreover, everyone, even a third party who did not participate in the election, can check that the election process was run faithfully.
- **No trusted authority needed for privacy.** There is no authority or participant in the electronic election that is able to guess what a voter voted for. Moreover, no

authority or participant is even able to detect whether or not a given voter cast a ballot.

- **Post-quantum security.** Our e-voting system is built from modular, post-quantum secure primitives, thus guaranteeing security against foreseeable future advances in quantum computing.

The itugen system is inspired by ring signatures [RST01] and, indeed, can be seen as a systematic implementation of ring signatures while offering additional benefits. In fact, according to our definition, an e-voting scheme is a ring signature scheme; in this case, the privacy guaranteed by our e-voting security model implies the signature's anonymity, and the system's verifiability implies the signature's linkability. Ring signatures have long been seen in the e-voting community as a useful tool for e-voting.

We take advantage of recent progress on succinct ZK proofs [Mic00, Kil95, GGPR13, BBB+18] to provide the first implementation of an efficient e-voting scheme with the aforementioned properties.

Specifically, our main ingredient is isekai, a versatile and easy-to-use verifiable computation library that includes implementations of SNARKs [GGPR13], IOP-based SNARKs like Aurora and Ligero [BSCS16, AHIV17, BSCR+19] and Bulletproofs [BBB+18]. The design of isekai is modular and allows users to instantiate builds from *post-quantum* primitives.

This paper is structured as follows. We first provide an overview of e-voting in Section 2. We then present our main tool, isekai, in Section 3. In Section 4, we formalize the notion of an anonymous e-voting scheme and its security model, and in Section 5 we present the itugen e-voting scheme.

2 E-voting Overview

2.1 Basic properties of e-voting systems

E-voting protocols must fulfill three basic security requirements:

- **Eligibility.** Only eligible voters registered for the election should be able to cast a vote. A malicious party or any administrator or authority participating in the election ceremony should fail in subverting the election result by casting ballots on behalf of other voters or adding spurious ballots.
- **Privacy.** The secrecy of each ballot's contents and the identity of its holder must be absolutely preserved at the technical level. We will later discuss this property in more detail.
- **Verifiability.** Verifiability (also called *integrity* in the literature) should prevent a corrupt participant in the protocol from altering or falsifying the tally. *Individual* verifiability guarantees to each voter that they can confirm that their individual ballot was counted, while *universal* verifiability ensures that anyone can confirm the overall tally (and in some cases, tallying metadata about any given anonymized ballot). We assume that each voter casts a ballot to a"public bulletin board" (PBB), a publicly-accessible set of records such as a blockchain or a public distributed ledger.

There are also other important properties of e-voting systems, such as usability and coercion-resistance. A discussion of these is outside of the scope of this paper.

In many e-voting schemes [Cha81, CGS97, DJ01, RS06, Adi08, CCC+09, RT09, JCJ10], privacy is achieved by assuming that an authority is *trusted* to tally the votes. Using secret sharing techniques, the trust is usually distributed among several authorities, but it is still required that not all authorities collude. Nevertheless, verifiability should be guaranteed even in case of malicious authorities or of the entire software system being uniformly compromised (for instance, by malware slipping into the common codebase, database or network).

Many e-voting schemes also make use of Non-Interactive Zero-Knowledge Proofs (NIZK) [BFM88, DMP88, Gol01] to provide verifiability. NIZK must provide two properties: soundness and zero-knowledge (ZK). Soundness prevents a corrupt prover from proving a false statement, i.e. a statement for which no witness exists. Zero-knowledge ensures that the verifier does not learn any additional information about the witness.

2.2 Standard approaches to e-voting: the example of Helios

The popular Helios e-voting scheme [Adi08] exemplifies the standard approach to constructing e-voting systems. The parties participating in Helios are multiple voters and one or multiple authorities. For simplicity, let us assume there is a single authority.

First, the authority sets up a public key for each voter using its chosen encryption scheme while retaining the corresponding secret key. A voter "computes" their ballot by inputting the public key of the authority and their intended vote and sends the ballot to a write-only public bulletin board (PBB). Ballots on the PBB can be public because the authority's private key is required to de-anonymize it and link it back to the voter's identity. The authority uses its secret key to confirm the validity of each entry and compute the tally from all ballots on the PBB.

Compting the tally can be done in Helios in two ways.

- **Homomorphic tallying** - In this setting, we assume the encryption scheme to be homomorphic. From all voters' ballots published on the PBB, the authority can compute a single ballot B encrypting the result t of the election. The authority can then announce t and add, for verifiability, a ZK proof that B encrypts the claimed result.
- **Shuffle-based tallying** - In the shuffle-based setting, we assume the encryption scheme to have the property that each ciphertext can be *re-randomized*. The authority can then re-randomize and randomly permute all ballots, adding a proof that it did this operation correctly, and finally decrypt each ciphertext in the permuted list, giving a corresponding proof of correct decryption.

Other e-voting systems, e.g. [RS06, RT09, JCJ10], add more properties and capabilities but still require authorities to be trusted for keeping the relationship between individual ballots and their holders private.

3 Isekai, a verifiable computation middleware

One of the main ingredients of itugen is isekai [ise], an open-source verifiable computation middleware developed by Sikoba Research [Sik] and written in Crystal language [Cry].

There are already several verifiable computation libraries being developed with different properties and unique capabilities. New implementations are also under development, but

without any common standards in place, benchmarking is effectively impossible, and deciding which verifiable computation system to use is difficult. This is where isekai comes in. Isekai provides a framework that allows users to choose the programming language and verifiable computation system they want, allowing them to test multiple options without having to make difficult choices at the beginning of a project.

Many verifiable computation systems work by transforming a program into a system of constraints that are satisfied by the result of the program. The proof system then applies some cryptographic functions to this set of equations. This constraint system is easily derived from an arithmetic circuit representation of a program, which is independent of the programming language of the program itself, as well as of the proof system. Isekai is able to transform any computation written in C or C++ into an equivalent arithmetic circuit representation. Not all programming language features are supported, but isekai is able to handle a fairly large subset of C and C++ structures, such as arrays, pointers, loops, function calls, integer/binary operations and control-flow graphs that can be obtained by compiling a C99 code, without goto, break, continue or return statements.

In practice, the input program is first converted into LLVM [LLV] bitcode[1] and then into an internal representation, which is in turn transformed into a circuit representation. Isekai is structured so that adding support for additional programming languages requires only that the transformation be changed into the internal representation, a process greatly simplified when there is an LLVM frontend for the language[2].

The language circuit and its execution trace will be the input to multiple proof systems, and a user can select the one that suits him well. Each proof system provides a proof of computational integrity for the circuit. The proofs, by default, are non-interactive, meaning that a verifier does not have to interact further with a prover to become convinced about the integrity. However, the proof systems we plan to support are quite different in their features, concretely:

- The language circuit must be converted into an arithmetic or Boolean circuit of some kind, with a very limited set of operations and all variables being from the same finite field. The field can be prime or binary.
- Some proof systems require a one-time additional setup for circuits (one or many) that is done by a trusted party.
- Proof size can be constant (large or small) or grow sublinearly with the circuit size.
- Verification time varies from constant to linear in the circuit size.

A typical life-cycle for the proof of the language circuit is the following:

1. The language circuit is converted to an arithmetic circuit.
2. If needed, a trusted setup is performed to create a global circuit description (GCD), which is published so that proofs can be verified.
3. The program is executed and its trace is recorded.
4. The program trace is converted to an arithmetic trace.
5. If needed, part of the arithmetic trace is published hidden in the form of cryptographic commitments.

[1] https://llvm.org/docs/BitCodeFormat.html

[2] Languages with LLVM compliers include C, C++, C#, Crystal, D, Haskell, Julia, Kotlin, Lua, Objective-C and Rust

6. The proof generator takes the arithmetic trace and GCD and outputs a proof, which is sent to a verifier.
7. The verifier checks the proof.

4 Definitions

4.1 Cryptographic primitives

The cryptographic ingredients we will use in our construction are as follows:

- A public-key encryption scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Setup}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$, whose private key can be secretly shared among multiple parties.
- A Merkle tree [Mer88] built from a collision-resistant hash function H.
- A succinct non-interactive ZK proof of knowledge system $\mathsf{Isekai} = (\mathsf{Isekai.Prov}, \mathsf{Isekai.Ver})$ that is supported by isekai. Note that if isekai is set up with a SNARK in the CRS[3] model, the proof system also includes a CRS generator algorithm. For simplicity, we will assume that the e-voting scheme is instantiated by isekai with a transparent succinct proof system like [BSBHR18, BBB$^+$18]. Without loss of generality, we can therefore assume that Isekai does not include a CRS-generation algorithm.
- A digital signature scheme $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.Sign}, \mathsf{DS.Ver})$, such that each verification-key PK output by $\mathsf{DS.Setup}$ corresponds to a unique secret key SK.

We assume the reader to be familiar with these primitives.

4.2 Anonymous e-voting scheme

In this section, we formalize the notion of an anonymous e-voting scheme and its security model. We first make some assumptions.

- **PBB** We assume the availability of a public bulletin board (PBB). That is, we assume that each participant can write a message to the PBB *anonymously* and such message is publicly visible to every other participant and cannot be deleted from the PBB or altered. The PBB could be implemented by means of a public blockchain or distributed ledger.
- **PKI assumption** We assume the availability of a public key infrastructure (PKI) in which each voter is given a unique secret key for a digital signature scheme $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.Sign}, \mathsf{DS.Ver})$ whose corresponding public key is made public on the PBB.
- **Identifier of the election** We assume a unique public identifier `id` is chosen for each election and published on PBB. The string `id` can, for instance, include the time when polling starts.

An anonymous e-voting (or simply e-voting) scheme is a tuple of efficient algorithms

[3] Comon Reference String

$$\mathsf{EVOTE} = (\mathsf{EVOTE.Setup}, \mathsf{EVOTE.Cast}, \mathsf{EVOTE.Tally}, \mathsf{EVOTE.Verify})$$

with the following syntax:

- $\mathsf{EVOTE.Setup}(1^\lambda)$: the setup algorithm, taking as input the security parameter λ, sets up the system, outputting a public key PK made public on the PBB and a secret key SK retained by the authority. The authority is trusted only for guaranteeing ballot independence and not for privacy or verifiability.

  This definition could be extended to support a set of linked authorities but, for simplicity, we can skip this detail and assume a single authority.

- $\mathsf{EVOTE.Cast}(\mathtt{id}, \mathsf{PK}, \mathsf{SK}_V, v)$: the casting algorithm is run by a voter who, taking as input the identifier of the election and the public key PK, her secret key SK_V for DS and her voting preference v, computes a *ballot* B that is written on the PBB.

  It may look suspicious that the verification algorithm needs the secret key but, as we will see later, the public key in the setup is only needed to guarantee ballot independence in the casting phase, after which the corresponding secret key is made available on the PBB.

- $\mathsf{EVOTE.Tally}(\mathtt{id}, \mathsf{PK}, \mathsf{SK}, B_1, \ldots, B_N)$: the tally algorithm, taking as inputs the election identifier, the public/secret key pair, and all ballots written on the PBB, computes the tally t of the election.

- $\mathsf{EVOTE.Verify}(\mathtt{id}, \mathsf{PK}, \mathsf{SK}, B_1, B_N, t)$: the verification algorithm, taking as inputs the election identifier, the public/secret key pair, all ballots written on the PBB, and the claimed tally t of the election, outputs 0 or 1.

  Moreover, when the algorithm is executed with an input of just $(\mathtt{id}, \mathsf{PK}, \mathsf{SK}, B)$ for a single ballot B, the algorithm outputs 1 or 0. This is used for individual verifiability.

Note that we are implicitly assuming that the input to the algorithms also includes the voters' public keys published on the PBB.

4.3 Security model

In our security model, we make a simplifying assumption. We assume that each voter submits their vote independently from other voters. We will see later how ballot independence can be guaranteed. In this simplified model, there is no authority at all, and we can assume that the secret key output by the setup procedure is an empty string.

We require an e-voting scheme EVOTE to satisfy the following properties:

Short ballots. The algorithm $\mathsf{EVOTE.Cast}$ should output a ballot of a length at most logarithmic in the length of the list of all voters' public keys and $\mathsf{EVOTE.Verify}$, when used for individual verifiability, should run in a time at most logarithmic to the length of the list of all voters' public keys.

Privacy. Privacy (or anonymity) requires that no efficient adversary Adv can win with more than non-negligible advantage in the following game $\mathsf{Priv}^{\mathcal{A},\mathsf{EVOTE},\lambda,n,\mathtt{id}}$, played between $\mathcal{A}$ and a challenger $\mathcal{C}$ (implicitly defined by the game), parameterized by security parameter λ, by the number of eligible voters n and by identifier $\mathtt{id}$.

$\mathsf{Priv}^{\mathcal{A},\mathsf{EVOTE},\lambda,n,\mathtt{id}}$:

- **Setup Phase.** $\mathcal{C}$ runs $\mathsf{DS.Setup}$ n times to generate n pairs of public and secret keys $(\mathsf{PK}_i, \mathsf{SK}_i)_{i=1}^n$. Furthermore, $\mathcal{C}$ runs $\mathsf{EVOTE.Setup}(1^\lambda)$ to compute the pair $(\mathsf{PK}, \mathsf{SK})$, and $\mathcal{A}$ is given PK.
- **Challenge Phase.** $\mathcal{A}$ outputs as its challenge two subsets S_0, S_0 of $\{1, \ldots, n\}$ of the same cardinality, $m \leq n$, and a list of m strings $(v_1, \ldots, v_m)$.
- **Cast Phase.** $\mathcal{C}$ chooses a random bit $b \leftarrow \{0, 1\}$ and for $i = 1, \ldots, m$ generates $B_i = \mathsf{EVOTE.Cast}(\mathtt{id}, \mathsf{PK}, \mathsf{SK}_{S_{b,i}}, v_{S_{b,i}})$.
- **Guess Phase.** $\mathcal{A}$ is given $(B_1, \ldots, B_m)$ and outputs its guess b'.
- **Winning Condition.** $\mathcal{A}$ wins the game if $b' = b$.

The previous definition can be naturally extended to the case of adversaries adaptively attacking the scheme in an unbounded number of elections but for simplicity we do not do it.[4]

Eligibility. The eligibility requirement states that no efficient adversary $\mathcal{A}$, taking as input only the public key of the election output by $\mathsf{EVOTE.Setup}$ (without any voter's secret key as input) is able to compute a ballot B that is accepted by $\mathsf{EVOTE.Verify}$ executed in the individual-verifiability mode.

Verifiability. Giving a definition of *fully satisfactory* verifiability is not an easy task and is out of the scope of this work. We mention that our e-voting scheme in Section 5 has the property that a ballot output by the cast algorithm consists of a pair $(v, \mathtt{aux})$, and the tally is computed just from the strings v, viewing them as the intended vote of the voter. In this setting, verifiability boils down to ensuring that no eligible voter should be able to cast two ballots that get counted; this will be clear from our construction.

5 Itugen: our e-voting scheme

In this section we present our e-voting scheme, itugen.

5.1 A first solution without ballot independence

As a warmup, we first present a construction that does not guarantee ballot independence. This simplified scheme suffers from the drawback that, at any point during the polling period, a voter can see the *partial* tally and may therefore be influenced by it. We will later show how to resolve this issue.

Our e-voting scheme

$$\mathsf{Itugen} = (\mathsf{Itugen.Setup}, \mathsf{Itugen.Cast}, \mathsf{Itugen.Tally}, \mathsf{Itugen.Verify})$$

will be implicitly described by describing how the different election phases work.

[4] Our scheme itugen of Section 5 also satisfies this more general security property.

Setup Phase. Let n be the number of eligible voters. For each $i = 1, \ldots, n$, we assume voter that i registers in the system with a public key PK_i according to the signature scheme DS and retains privately the corresponding secret key SK_i. All public keys $(\mathsf{PK}_1, \ldots, \mathsf{PK}_n)$ of all eligible voters are published on the PBB.

The setup algorithm $\mathsf{Itugen.Setup}$ builds a Merkle tree M from $(\mathsf{PK}_1, \ldots, \mathsf{PK}_N)$ and publishes it on the PBB. Let us call R the root of M. The setup procedure outputs R as a public key. Note that the voters' individual public keys are the leaves of the Merkle tree M.

Recall that we also assume that a public identifier $\mathtt{id}$ is chosen for this election and published on PBB. The string $\mathtt{id}$ can be, for instance, the day on which the election is run.

Cast Phase. Each eligible voter V who decides to cast a vote v proceeds as follows.

Let v be the intended vote that V wants to cast. The string v can be either a valid voting option (e.g. the name of a candidate) or an arbitrary string representing an invalid vote. (Note that this possibility of submitting an invalid ballot is also given to voters in traditional paper-based elections.)

Let SK_V be the secret key of V corresponding to public key PK_V, and let $h_V = H(\mathsf{SK}_V, \mathtt{id})$ and $\sigma_V = \mathsf{DS.Sign}(\mathsf{SK_V}, \mathsf{v}||\mathsf{h_V})$.

Let p be the Merkle path in M that proves that R is the root of a tree that has PK_V as a leaf. Note that the length of p is logarithmic in M.

Let $C^{R,v,h_V,\mathtt{id}}$ be the following Boolean circuit with one output gate. $C^{R,v,h_v,\mathtt{id}}$ depends on the constants $R, v, h_V, \mathtt{id}$, takes as input a triple $w = (p, \mathsf{SK}_V, \sigma_V)$ and outputs 1 if and only if *all* the following conditions are verified:

1. The string p is a Merkle path from R to PK_V.
2. SK_V is a secret key for DS corresponding to public key PK_V. (Here, we are implicitly assuming that the digital signature scheme DS is such that there is an efficient way to check that SK_V is the unique secret key corresponding to PK_V.)
3. $h_v = H(\mathsf{SK}_V, \mathtt{id})$.
4. $\sigma_V = \mathsf{DS.Sign}(\mathsf{SK_V}, \mathsf{v}||\mathsf{h_v})$.

Note that the constants $R, v, h_V, \mathtt{id}$, and thus $C^{R,v,h_v,\mathtt{id}}$, are public information while w is only known to V. The voter V uses $\mathsf{Isekai.Prov}$ to compute a proof π_V of the fact that $C^{R,v,h_v,\mathtt{id}}$ is satisfied by witness w.

V publishes their ballot $B_V \stackrel{\triangle}{=} (v, h_V, \pi_V)$ on the PBB. Note that for ease of exposition, we use the subscript V in the strings written on the PBB, but such strings cannot be associated with the voter V.

The algorithm $\mathsf{Itugen.Cast}$ is implicit from the previous description.

Discarding invalid votes and duplicates. When a voter sends a ballot (v, h_V, π_V) to the PBB, we assume that if the proof π_V is not accepted by $\mathsf{Isekai.Ver}(C^{R,v,h_v,\mathtt{id}}, \pi_V)$, an error symbol $\perp$ is written next to the ballot to indicate that v should not be considered for the tally.

Moreover, if two ballots $B_1 = (v_1, h_1, \pi_1)$ and $B_2 = (v_2, h_2, \pi_2)$ with $h_1 = h_2$ are found, they should be considered a *duplicate* vote. Indeed, the purpose of the hash $H(\mathsf{SK}_V, \mathtt{id})$ is to link

the voter V to the election $\mathtt{id}$ so as to be able to detect whether V voted multiple times without revealing the identity of V. Depending to the policy of a given election encoded in the setup phase, both ballots can be discarded, or the second one can be discarded as invalid, or the first can be superseded by the second.

Tallying Phase. The tally can now be publicly computed from all votes v in each ballot. Note that v can still be an invalid voting option, but we have the assurance that one of the voters (we do not know which) intended to write v on the PBB.

Verification Phase. The validity of the election process can be checked by any party, even a third party not participating in the election, simply by verifying that the ZK proofs in each ballot are verified and that there are no two ballots with the same hash (see above). This can be done at any point in time, assuming that the information on the PBB is immutable, even while votes are still being cast.

The algorithms $\mathsf{Itugen.Verify}$ and $\mathsf{Itugen.Tally}$ are implicit from the previous description.

5.2 Achieving ballot independence.

To solve the issue of ballot independence, we assume there are a set of authorities trusted only for independence but not for privacy or verifiability.

The authorities set up a public key PK for $\mathcal{E}$, sharing the corresponding secret key SK among them; that is, the i-th authority retains a share SK_i of SK. At casting phase, a voter produces a ballot B computed as before but, instead of sending B directly to the PBB, they send B encrypted by PK. At the end of the election, the authorities publish their shares of SK on the PBB, from which each ballot can be decrypted. The election then continues as before.

Removing authorities altogether via time-sensitive cryptography. One alternative that does not require authorities is to use time-sensitive primitives like the time-locks used by Rivest *et al.* [RSW96]. Using time locks, it is possible to encrypt a message so that it cannot be decrypted by anyone until a pre-determined amount of time has passed but can be decrypted after running a computer for a certain amount of time.

5.3 Security analysis

Eligibility and verifiability. An eligibility guarantee is implied by the difficulty of forging signatures for adversaries that do not have any valid secret keys and by the soundness of the SNARK system.

Recall that each ballot $B_V = (v, h_V, \pi_V)$ corresponding to some voter V includes the voter V's vote v in the clear. Therefore, verifiability boils down to guaranteeing that each voter cannot submit two valid votes.

Suppose as a counterexample that a voter V, having public key PK_V corresponding to the unique secret key SK_V, is able to cast two ballots $B_1 = (v_1, h_1, \pi_1)$ and $B_2 = (v_2, h_2, \pi_2)$ such that π_1 and π_2 are accepted proofs and $h_1 \neq h_2$, i.e. that the ballots are not considered duplicates. By the extractability property of the SNARK, the unique secret-key property of DS and the definition of circuits $C^{R,v_i,h_i,\mathtt{id}}$, $i = 1, 2$, we can show that $h_1 = H(\mathsf{SK}_V, \mathtt{id}) = h_2$, contradicting the fact that $h_1 \neq h_2$.

Privacy. The privacy property is implied by the zero-knowledge property of the SNARK. Indeed, a ballot (computationally) hides the witness and thus the path in the Merkle tree from the root to the public key of the voter.

6 Conclusion

E-voting schemes without authorities trusted for privacy have been proposed in other limited scenarios, such as ones that assume all eligible voters have to cast a vote (see [GIR16] and references therein). With itugen, we propose a scheme that offers complete transparency as well as anonymity without relying on trusted authorities and without any limiting assumptions.

Because itugen does not rely on any central authority, a practical implementation of it will require a PBB that is tamper-proof and offers very high availability. As these are core features of distributed ledger technology (see for example [WGP+17] [BMTZ17], [LS18]), itugen should naturally tend to use a blockchain as its PBB.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 103–112. ACM Press, May 1988.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[BSCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 103–128. Springer, 2019.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2016.

[CCC+09] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: end-to-end verifiability by voters of

optical scan elections through confirmation codes. *IEEE Trans. Information Forensics and Security*, 4(4):611–627, 2009.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, May 1997.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cry] Crystal language. `https://crystal-lang.org/`.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, February 2001.

[DMP88] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 52–72. Springer, August 1988.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, May 2013.

[GIR16] Rosario Giustolisi, Vincenzo Iovino, and Peter Rønne. On the possibility of non-interactive voting in the public-key setting. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, 2016.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Techniques*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[ise] isekai. `https://github.com/sikoba/isekai`.

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63. Springer, 2010.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, August 1995.

[LLV] Llvm. `https://llvm.org`.

[LS18] Esteban Landerreche and Marc Stevens. On immutability of blockchains. In *Proceedings of the 1st ERCIM Blockchain Workshop 2018*, Reports of the European Society for Socially Embedded Technologies, 2018.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, August 1988.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[RS06] Peter Y. A. Ryan and S. A. Schneider. Prêt à voter with re-encryption mixes. Technical Report CS-TR-956, University of Newcastle, 2006.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, December 2001.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. *Massachusetts Institute of Technology*, 1996.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In *IN: WORKSHOP ON SECURITY PROTOCOLS*, 2009.

[Sik] Sikoba research. `https://research.sikoba.com`.

[WGP+17] Ingo Weber, Vincent Gramoli, Alexander Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. On availability for blockchain-based systems. In *SRDS*, pages 64–73. IEEE Computer Society, 2017.