Cicada: A framework for private non-interactive on-chain auctions and voting

Noemi Glaeser*[1,2], István András Seres†[3], Michael Zhu‡[4], and Joseph Bonneau§[4,5]

[1]University of Maryland
[2]Max Planck Institute for Security and Privacy
[3]Eötvös Loránd University
[4]a16z crypto research
[5]New York University

March 14, 2024

Abstract

Auction and voting schemes play a crucial role in the Web3 ecosystem. Yet currently deployed implementations either lack privacy or require at least two rounds, hindering usability and security. We introduce Cicada, a general framework for using linearly homomorphic time-lock puzzles (HTLPs) to enable provably secure, non-interactive private auction and voting protocols. We instantiate our framework with an efficient new HTLP construction and novel packing techniques that enable succinct ballot correctness proofs independent of the number of candidates. We demonstrate the practicality of our approach by implementing our protocols for the Ethereum Virtual Machine (EVM).

1 Introduction

Auctions and voting are essential applications of Web3. For example, decentralized marketplaces run auctions to sell digital goods like non-fungible tokens (NFTs) [Ope23] or domain names [XWY+21], while decentralized autonomous organizations (DAOs) deploy voting schemes to enact decentralized governance [Opt23]. Most auction or voting schemes currently deployed on blockchains, e.g., NFT auctions on OpenSea or Uniswap governance [FMW22], lack bid/ballot privacy. This can negatively influence user behavior, for example, by vote herding or discouraging participation [EL03, GY18, SY03]. The lack of privacy can also cause surges in congestion and transaction fees as users try to outbid each other to participate, a negative externality for the entire network.

Existing private voting protocols [PE23, GG22, PE] achieve privacy at the cost of introducing a trusted authority who is still able to view all submissions. Alternatively, the only deployed private *and* trustless auction we are aware of [XWY+21] deploys a two-round commit-reveal protocol: in the first round, every party commits to their bid, and in the second round they open the commitments and the winner can be determined. Other protocols relying on more heavyweight cryptographic building blocks have been proposed in the literature. We summarize several approaches for private voting and auctions in Table 1. Unfortunately, each approach suffers from at least one of the following limitations, hindering widespread adoption:

Interactivity. Interactivity is a usability hurdle that often causes friction in the protocols' execution. Mandatory bid/ballot reveals are also a target for censorship. A malicious party can bribe the block proposers to exclude certain bids or ballots until the auction/voting ends [PRF23].

Trusted third party (TTP). Many protocols use a trusted coordinator to tally submissions during the voting/bidding phase to achieve essential security (e.g., availability, liveness) or privacy [PE23] guarantees. We argue that in most decentralized applications, relying on a trusted third party or a threshold of them is at odds with the ethos of decentralization and trust-minimization.

*nglaeser@umd.edu. The majority of this work was done at a16z crypto research.
†seresistvanandras@gmail.com. The majority of this work was done at a16z crypto research.
‡mzhu@a16z.com
§jbonneau@gmail.com

Approach	Non-interactive	No TTPs	Efficient	Tally privacy	Everlasting privacy
Commit-reveal [FOO93, GY18]	○	●	●	●	○
Zero-knowledge (shuffle) proofs [Adi08, PE23]	○	○	●	●	●
Fully homomorphic TLPs [MT19]	●	●	○	●	○
Fully homomorphic encryption [Gen09, CGGI16, DPLNS17]	●	○	○	●	●
Multi-party computation [BDJ+06, AOZZ15]	○	◐	●	●	●
TLPs + homomorphic encryption [CJSS21]	●	◐	●	●	○
HTLPs (our approach)	●	●*	●	●	◐

Table 1: Qualitative comparison of major cryptographic approaches for designing private auction/voting schemes. An asterisk indicates that those schemes can be instantiated with a transparent setup using class groups, cf. Section 7.1. Everlasting ballot privacy can be added to our approach via an extension (Section 8.1).

Inefficiency. Compute and storage costs are substantial bottlenecks in decentralized applications running on a public blockchain. Some approaches [CGGI16, DPLNS17] avoid the previous pitfalls by relying on complex cryptographic primitives such as fully-homomorphic encryption (FHE), whose overheads are impractical in the blockchain setting.

1.1 Our high-level goals

We aim to build voting and auction protocols that possess the following distinguishing features compared to prior work.

Trust-minimization. In our protocols, we do not want to assume (a quorum of) trusted third parties. The cryptographic voting literature extensively applies trusted parties, for example, to operate a public bulletin board or tally votes. The classic tools used in prior work, e.g. (fully) homomorphic encryption, inherently imply a handful of trusted parties to decrypt the ballots/bids. Apart from the liveness and safety of the blockchain consensus, we solely employ standard cryptographic assumptions.

One-round protocol. We argue that usability is one of the major challenges of deploying privacy-preserving voting and auction protocols in practice. Multi-round protocols, e.g., commit-reveal-style protocols, have incentive issues and considerable usability hurdles. We solve these pressing issues with efficient one-round protocols.

Ballot/bid privacy. Last but not least, we want to achieve ballot/bid privacy. Our approach naturally provides privacy until the end of the voting/bidding phase. Additionally, we show novel cryptographic techniques in Section 8.1 how we can achieve everlasting ballot and bid privacy without sacrificing our two previously stated goals.

Coercion resistance, i.e., the adversary's inability to coerce voters to cast specific ballots demanded by the adversary, is a crucial property of voting schemes. In the privacy-respecting e-voting literature, it is well-known that receipt-freeness (since the voters cannot prove how they voted) implies coercion-resistance [BT94]. However, we consider receipt-freeness as a non-goal in our protocol design. Still, we sketch an extension to our framework in Section 8.3, where we achieve coercion resistance via a different pathway than receipt freeness. We leave it to future work to achieve the property of receipt freeness for on-chain voting schemes.

1.2 Our approach

In this work, we introduce Cicada, a general framework for practical, privacy-preserving, and trust-minimized protocols for both auctions and voting. Cicada uses time-lock puzzles (TLPs) [RSW96] to achieve *privacy and non-interactivity* in a trustless and efficient manner. Intuitively, the TLPs play the role of commitments to bids/ballots that any party can open after a predefined time, avoiding the reliance on a second *reveal* round. Since solving a TLP is computationally intensive, ideally, we would solve only a sublinear number of TLPs (in the number of voters/bidders) for efficiency. This is achieved using *homomorphic* TLPs (HTLPs): bids/ballots encoded as HTLPs can be "squashed" into a sublinear number of TLPs. Although fully homomorphic TLPs are not practically efficient, Malavolta and Thyagarajan [MT19] introduced efficient additively and multiplicatively homomorphic TLP constructions. This is enough for simple constructions like first-past-the-post (FPTP) voting, but it has been an open challenge to use HTLPs to realize more complicated auction and voting protocols, e.g., cumulative voting.

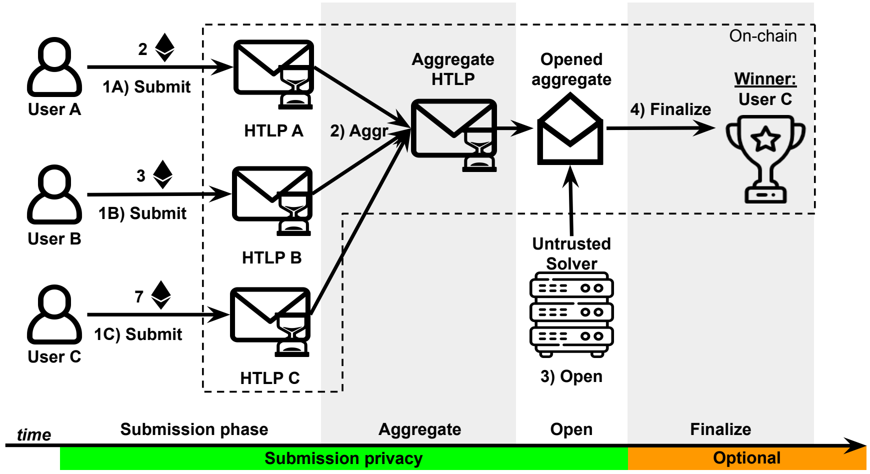

Figure 1: Our model of an HTLP-based auction/voting scheme. *(1) Submission phase:* users generate their bids/ballots as HTLPs and post them to a public bulletin board, e.g., a blockchain. *(2) Aggregation:* an on-chain contract homomorphically combines submissions into a single aggregate puzzle. *(3) Opening:* after the submissions have been aggregated, an off-chain entity solves the aggregate HTLP using T sequential steps and submits the solution to the contract. *(4) Finalize:* The smart contract may do some final computation over the solution to compute the result and announces the winner. Submission privacy is ensured only until the start of the Open phase. In Section 8.1, we show how voters can optionally achieve everlasting ballot privacy.

Our contributions. We show how to use HTLPs to build *non-interactive* protocols for complex auction and voting schemes. Our protocols are both practically efficient, private until the end of the voting/bidding phase, and provably secure, overcoming the following challenges:

Efficient proofs for bid/ballot correctness. Users need to prove that their bids/ballots are well-formed according to the auction/voting protocol rules. Designing protocols that admit concretely efficient proofs for bid/ballot correctness is challenging as we work in groups of unknown order, e.g., $\mathbb{Z}_N^*$. In particular, we wish to minimize the proof size and the verification cost since proofs are stored and verified on-chain.

EVM-friendliness. We provide open-source, freely available implementations tailored to the popular Ethereum Virtual Machine (EVM) with word size 256 bits. Our most efficient protocols work in $\mathbb{Z}_N^*$ for $N \approx 2^{1024}$, groups which are not natively supported by EVM. We implement several gas-efficient libraries to support modular arithmetic in such groups of unknown order. We demonstrate in Section 7 that these protocols can be run today on Ethereum Layer 1. Our non-interactive protocols are particularly well-suited to the EVM since, unlike prior works, we do not need to keep bids, ballots, and proofs in persistent storage as they are not required for any subsequent round.

2 System Model

Our system design is illustrated in Figure 1. We envision three types of participants:

Users. We simply refer to voters or bidders as *users*. Users submit bids or ballots, which we generically call *submissions*. We assume some external process to establish the set of authorized users (which may be open to all). Once users place their submissions, no further action is required of them.

On-chain coordinator. We refer to the tallier/auctioneer as the *coordinator*, typically implemented as a smart contract that collects submissions. The coordinator transparently calculates the winner(s). In the case of an auction, they might also transfer (digital) assets to the winner(s). In an election, they might grant special privileges to the winner's public key.

Off-chain solver. Since our protocols apply HTLPs, we assume an untrusted *solver* who unlocks the final HTLP(s) off-chain and submits the solution(s) to the coordinator with proofs of correct opening attached. This could, in principle, be any party, although, in practice, it will likely be one of the parties participating in (or administering) the vote/auction, or a paid marketplace [AM23, TGB+21].

An adversary may attempt to read ballots/bids before the submission phase is complete. This is prevented by the security properties of (H)TLPs (Section 3.3) assuming the delay T is longer than the submission phase.

3 Preliminaries

3.1 Notation

We use $[n]$ to denote a range of positive integers $\{1, \ldots, n\}$. For other ranges (mostly zero-indexed), we explicitly write the (inclusive) endpoints, e.g., $[0, n]$. Concatenation of vectors $\mathbf{x}, \mathbf{y}$ is written as $\mathbf{x}||\mathbf{y}$. We will use n as the number of users, m as the number of candidates, and w as the maximum weight to be allocated to any one candidate in a ballot/bid ($n, m, w \in \mathbb{N}$). For simplicity and without loss of generality, we assume the user identities are unique integers $i \in [n]$. We generally use $i \in [n]$ to index users and $j \in [m]$ for candidates. We use a calligraphic font, e.g., $\mathcal{S}$ or $\mathcal{X}$, to denote sets or domains. When we apply an operation to two sets of equal size ℓ we mean pairwise application, e.g., $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$ means $z_i = x_i + y_i \ \forall i \in [\ell]$. The output y of a randomized algorithm Alg is written as $y \xleftarrow{R} \mathsf{Alg}(x)$ and randomly sampling an element x from a set $\mathcal{X}$ is written as $x \xleftarrow{R} \mathcal{X}$. Many of our algorithms take some public parameters pp as input, and we also drop this input when pp is clear from context.

3.2 Voting and auction protocols

We recall the specifics of FPTP, approval, range, and cumulative voting, along with single-item sealed bid auctions. The cryptographically relevant details of these schemes (i.e., the valid ballots' structure: their domain, Hamming weight, and norm) are summarized in Table 2. In Section 5, we create private voting protocols for these schemes of interest.

Majority, approval, range, and cumulative voting. In the classic majority (i.e., first-past-the-post or FPTP) voting scheme, users can cast 0 (oppose) or 1 (support) for a given candidate (or cause). A slight generalization of FPTP is approval voting, where users can submit a binary vote to multiple candidates, i.e., the cast ballot s can be seen as $s \in \{0, 1\}^m$, where m is the number of causes. A different generalization is range voting(also known as score voting), where users can give each candidate up to some weight w(thus, FPTP is the special case where $w = 1$). A related scheme is cumulative voting, where users can distribute w votes (points) among the candidates(now approval voting is the special case where $w = 1$).

Ranked-choice voting. In a ranked-choice voting scheme, voters can signal more fine-grained preferences among m candidates. In the Borda count version [Eme13], each voter can cast $m - 1$ points to their first-choice candidate, $m - 2$ points to their second-choice candidate, etc. In general, they can cast $m - k$ points to their kth choice. Several other counting functions exist for ranked voting, but in this work, we only focus on Borda counts. Our protocols can easily be adapted to other counting functions, such as the Dowdall system [FG14] via minor modifications.

Quadratic voting. In quadratic voting [LW18], each user's ballot is a vector $\mathbf{b} = (b_1, \ldots, b_m)$ such that $\langle \mathbf{b}, \mathbf{b} \rangle = \|\mathbf{b}\|_2^2 \leq w$. Once again, the winner is determined by summing all the ballots and determining the candidate with the most points. Thus, this is also an additive voting scheme. However, proving ballot well-formedness efficiently in this particular case benefits greatly from the novel application of the residue numeral system (RNS) to private voting (see Section 5.1).

Single-item sealed-bid auction. In a sealed-bid auction for a single item (e.g., an NFT or domain name), users submit secret bids to the auction contract. The domain of the bids might be constrained, e.g., $b \in \{0, 1\}^k$ (in our implementations $k \approx 8 - 16$; see Section 7.2). Therefore, bidders must prove that their bid is well-formed, i.e., falls into that domain. Once all secret bids are revealed, the contract selects the highest bidder and awards them the auctioned item. The price the winner must pay depends on the auction scheme: e.g., highest bid in a first price auction, second-highest in a Vickrey auction.

3.3 (Homomorphic) Time-lock puzzles

A time-lock puzzle (TLP) [RSW96] consists of three efficient algorithms $\mathsf{TLP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve})$ allowing a party to "encrypt" a message to the future. To recover the solution, one needs to perform a computation that is believed to be inherently sequential, with a parameterizable number of steps.

	Submission domain	Hamming wt	Norm
Voting schemes			
Cumulative	$[0, w]^m$	$\leq m$	$\leq w$
Range	$[0, w]^m$	$\leq m$	$\leq wm$
Ranked-choice (Borda)	$\pi([0, m-1])$	$m-1$	$m(m-1)/2$
Quadratic (Section 6.2)	$[0, \sqrt{w}]^m$	$\leq m$	$\\|\mathbf{b}\\|_2^2 = \langle \mathbf{b}, \mathbf{b} \rangle = w$
Single-item sealed-bid auction	$[0, w]$	1	$\leq w$
Bayesian truth serum (B.1)	$[0, 1]^m \times \mathbb{N}^m$	$1, 1$	$1, \leq m$

Table 2: Requirements for the domain, Hamming weight, and norm of a vector $\mathbf{b}$ in order for it to be a valid submission in various voting/auction schemes. $\pi(S)$ denotes the set of permutations of S. The norm is an ℓ_1 norm unless otherwise specified. m is the number of candidates and w is the maximum weight which can be assigned to any one candidate.

Definition 1 (Time-lock puzzle [RSW96]). *A time-lock puzzle scheme* TLP *for solution space* $\mathcal{X}$ *consists of the following three efficient algorithms:*

$\mathsf{TLP.Setup}(1^\lambda, T) \xrightarrow{R} \mathsf{pp}$**.** *The (potentially trusted) setup algorithm takes as input a security parameter* 1^λ *and a difficulty (time) parameter* T*, and outputs public parameters* pp.

$\mathsf{TLP.Gen}(\mathsf{pp}, s) \xrightarrow{R} Z$**.** *Given a solution* $s \in \mathcal{X}$*, the puzzle generation algorithm efficiently computes a time-lock puzzle* Z*.*

$\mathsf{TLP.Solve}(\mathsf{pp}, Z) \to s$**.** *Given a TLP* Z*, the puzzle solving algorithm requires at least* T *sequential steps to output the solution* s*.*

Informally, we say that a TLP scheme is *correct* if TLP.Gen is efficiently computable and TLP.Solve always recovers the original solution s to a validly constructed puzzle. A TLP scheme is *secure* if Z hides the solution s and no adversary can compute TLP.Solve in fewer than T steps with non-negligible probability. For the formal definitions, we refer the reader to [MT19].

Homomorphic TLPs. Malavolta and Thyagarajan [MT19] introduce *homomorphic* TLPs (HTLPs). An HTLP is defined with respect to a circuit class $\mathcal{C}$ and has an additional algorithm Eval. defined as:

$\mathsf{HTLP.Eval}(\mathsf{pp}, C, Z_1, \ldots, Z_m) \to Z_*$**.** Given the public parameters, a circuit $C \in \mathcal{C}$ where $C : \mathcal{X}^m \to \mathcal{X}$, and input puzzles $Z_1, \ldots, Z_m$, the homomorphic evaluation algorithm outputs a puzzle Z_*.

Correctness requires that the puzzle obtained by homomorphically applying the circuit C to m puzzles should contain the expected solution, namely $C(s_1, \ldots, s_m)$, where $s_i \leftarrow \mathsf{HTLP.Solve}(Z_i)$. Again, we refer the reader to [MT19] for the formal definition. Moving forward, we will use $\boxplus$ for homomorphic addition and $\cdot$ for scalar multiplication of HTLPs. For the homomorphic application of a linear function f, we write $f(Z_1, \ldots, Z_m)$.

Malavolta and Thyagarajan [MT19] construct two HTLPs with, respectively, linear and multiplicative homomorphisms in groups of unknown order. For our purposes we are only interested in the former, which is based on the Paillier cryptosystem [Pai99]. It uses $N = pq$ a strong semiprime, $g \xleftarrow{R} \mathbb{Z}_N^*$ and $h = g^{2^T}$, and has solution space $\mathbb{Z}_N$:

$$Z := (g^r, h^{r \cdot N}(1+N)^s) \in \mathbb{J}_N \times \mathbb{Z}_{N^2}^* \quad (1)$$

To recover s, a solver must recompute $h^r = (g^r)^{2^T}$, which is believed to be inherently sequential in a group of unknown order.

As an alternative, we introduce a novel linear HTLP based on the exponential ElGamal cryptosystem [CGS97] over a group of unknown order. This construction requires a small solution space $\mathcal{X} \subset \mathbb{Z}_N$, i.e., $\mathcal{X} = \{s : s \in \mathbb{J}_N \wedge \ s \ll N\}$. Let $g, y \xleftarrow{R} \mathbb{Z}_N^*$ and again $h = g^{2^T}$, and construct the puzzle as

$$Z := (g^r, h^r y^s) \in (\mathbb{Z}_N^*)^2 \quad (2)$$

This scheme is only practical for small $\mathcal{X}$ since, in addition to recomputing h^r, recovering s requires brute-forcing the discrete modulus of y^s. We discuss the efficiency trade-off between these two constructions in Section 7.2 and relegate the construction details to Appendix A.1.

Non-malleability. Introducing a homomorphism raises the issue of puzzle malleability, i.e., the possibility of "mauling" one puzzle (whose solution may be unknown) into a puzzle with a related solution. This could lead to issues when HTLPs are deployed in larger systems, prompting research into non-malleable TLPs [FKPS21]. In our case, we define and enforce non-malleability at the system level (Section 4).

3.4 Non-interactive zero-knowledge proofs

A proof system $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is defined with respect to relation $\mathcal{R}_\mathcal{L}$ with NP language $\mathcal{L}$ with statement-witness pairs $(x; \omega) \in \mathcal{R}_\mathcal{L}$. We will use *non-interactive zero-knowledge* proofs (NIZKs) to enforce well-formedness of user submissions while maintaining their secrecy. This prevents users from "poisoning" the aggregate HTLP maintained by the on-chain coordinator. For efficiency, we make use of custom NIZKs(see Section 6). We refer to [Tha23] for the formal security definitions of NIZKs (soundness and zero-knowledge).

Applied NIZKs in Groups of Unknown Order. Since submissions will be instantiated as HTLPs in our application and all known HTLP constructions use groups of unknown order, our proofs of well-formedness must also operate over these groups. Previous ballot correctness proofs [Gro05] and sigma protocols [Sch89, CP92] generally operate in groups of prime order and cannot directly be applied in groups of unknown order [BCM05]. To circumvent these impossibility results, we follow the blueprint of [BBF19] and instantiate our protocols in generic groups of unknown order [DK02] with a common reference string. We detail our protocols in Section 6.

4 Syntax of Time-Locked Voting and Auction Protocols

We now introduce a generic syntax for a time-locked voting/auction protocol. Any such protocol is defined with respect to a base *scoring function* $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ (e.g., second-price auction, range voting), which takes as input n submissions (bids/ballots) $s_1, \ldots, s_n$ in the submission domain $\mathcal{X}$ and computes the election/auction result $\Sigma(s_1, \ldots, s_n) \in \mathcal{Y}$. It is useful to break down the scoring function into the "tally" or aggregation function $t : \mathcal{X}^n \to \mathcal{X}'$ and the finalization function $f : \mathcal{X}' \to \mathcal{Y}$, i.e., $\Sigma = f \circ t$. For example, in first-past-the-post voting, the tally function t is addition, and the finalization function f is arg max over the final tally/bids.

Definition 2 (Time-locked voting/auction protocol)**.** *A time-locked voting/auction protocol* $\Pi_\Sigma = (\mathsf{Setup}, \mathsf{Seal}, \mathsf{Aggr}, \mathsf{Open}, \mathsf{Finalize})$ *is defined with respect to a base voting/auction protocol* $\Sigma = f \circ t$*, where* $t : \mathcal{X}^n \to \mathcal{X}'$ *and* $f : \mathcal{X}' \to \mathcal{Y}$*.*

$\mathsf{Setup}(1^\lambda, T) \xrightarrow{R} (\mathsf{pp}, \mathcal{Z})$**.** *Given a security parameter* λ *and a time parameter* T*, output public parameters* pp *and an initial list of HTLP(s)* $\mathcal{Z}$ *that corresponds to the running tally or bid computation.*

$\mathsf{Seal}(\mathsf{pp}, i, s) \xrightarrow{R} (\mathcal{Z}_i, \pi_i)$**.** *User* $i \in [n]$ *wraps its submission* $s \in \mathcal{X}$ *in a (list of) HTLP(s)* $\mathcal{Z}_i$*. It also outputs a proof of well-formedness* π_i*.*

$\mathsf{Aggr}(\mathsf{pp}, \mathcal{Z}, i, \mathcal{Z}_i, \pi_i) \to \mathcal{Z}'$**.** *Given a list of (tally) HTLPs* $\mathcal{Z}$*, time-locked submission* $\mathcal{Z}_i$ *of user* i*, and proof* π_i*, the transparent contract potentially aggregates the sealed submission homomorphically into* $\mathcal{Z}$ *to get an updated (tally)* $\mathcal{Z}' = t(\mathcal{Z}, \mathcal{Z}_i)$*.*

$\mathsf{Open}(\mathsf{pp}, \mathcal{Z}) \to (\mathcal{S}, \pi_{\mathsf{open}})$**.** *Open* $\mathcal{Z}$ *to solution(s)* $\mathcal{S}$*, requiring* T *sequential steps, and compute a proof* π_{open} *to prove correctness of* $\mathcal{S}$*.*

$\mathsf{Finalize}(\mathsf{pp}, \mathcal{Z}, \mathcal{S}, \pi_{\mathsf{open}}) \to \{y, \perp\}$**.** *Given proposed solution(s)* $\mathcal{S}$ *to* $\mathcal{Z}$ *with proof* π_{open}*, the coordinator may reject* $\mathcal{S}$ *or compute the final result* $y = f(\mathcal{S}) \in \mathcal{Y}$*.*

We note that the $\mathsf{Setup}(\cdot)$ algorithm in our protocols may use private randomness. In particular, our constructions use cryptographic groups (RSA and Paillier groups) that cannot be efficiently instantiated without a trusted setup (an untrusted setup would require gigantic moduli [San99]). This trust can be minimized by generating the group via a distributed trusted setup, e.g., [BF01, CHI+21, DM10]. Alternatively, the HTLPs in our protocols could be instantiated in class groups [TCLM21], which do not require a trusted setup; however, HTLPs in class groups are less efficient and verifying them on-chain would be prohibitively costly (see Section 7.1).

A time-locked voting/auction protocol Π_Σ must satisfy the following three security properties:

Correctness. Π_Σ is *correct* if, assuming setup, submission of n puzzles, aggregation of all n submissions, and opening are all performed honestly, $\mathsf{Finalize}$ outputs a winner consistent with the base protocol Σ.

Definition 3 (Correctness)**.** *We say a voting/auction protocol* Π_Σ *with* $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ *is* correct *if for all* $T, \lambda \in \mathbb{N}$ *and submissions* $s_1, \ldots, s_n \in \mathcal{X}$*,*

$$\Pr\left[\begin{array}{c} \mathsf{Finalize}(\mathsf{pp}, \mathcal{Z}_{\mathsf{final}}, \mathcal{S}, \pi_{\mathsf{open}}) \\ = \Sigma(s_1, \ldots, s_n) \end{array}\middle|\begin{array}{c} (\mathsf{pp}, \mathcal{Z}) \xleftarrow{R} \mathsf{Setup}(1^\lambda, T) \wedge \\ (\mathcal{Z}_i, \pi_i) \xleftarrow{R} \mathsf{Seal}(\mathsf{pp}, i, s_i) \ \forall i \in [n] \wedge \\ \mathcal{Z}_{\mathsf{final}} \leftarrow \mathsf{Aggr}(\mathsf{pp}, \mathcal{Z}, \{i, \mathcal{Z}_i, \pi_i\}_{i \in [n]}) \wedge \\ (\mathcal{S}, \pi_{\mathsf{open}}) \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{Z}_{\mathsf{final}}) \end{array}\right] = 1$$

where the aggregation step is performed over all n submissions in any order.

Submission privacy. The scheme satisfies *submission privacy* if the adversary cannot distinguish between two submissions, i.e., bids or ballots. Note that this property is only ensured up to time T.

Definition 4 (Submission privacy)**.** *We say that a voting/auction protocol* Π_Σ *with* $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ *is* submission private *if for all* $T, \lambda \in \mathbb{N}, i \in [n]$ *and all PPT adversaries* $\mathcal{A}$ *running in at most* T *sequential steps, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[b = b' \middle| \begin{array}{c} (\mathsf{pp}, \mathcal{Z}) \xleftarrow{R} \mathsf{Setup}(1^\lambda, T) \wedge \\ b \xleftarrow{R} \{0,1\} \wedge \\ (\mathcal{Z}_i, \pi_i) \xleftarrow{R} \mathsf{Seal}(\mathsf{pp}, i, b) \wedge \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathcal{Z}, i, \mathcal{Z}_i, \pi_i) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Non-malleability. Notice that submission privacy alone does not suffice for security: even without knowing the contents of other puzzles, an adversary could submit a value that depends on other participants' (sealed) submissions. For example, in an auction, one could be guaranteed to win by homomorphically computing an HTLP containing the sum of all the other participants' bids plus a small value ϵ. Therefore, we also require *non-malleability*, which requires that no participant can take another's submission and replay it or "maul" it into a valid submission under its own name.

Definition 5 (Non-malleability)**.** *We say that a voting/auction protocol* Π_Σ *with* $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ *is* non-malleable *if for all* $T, \lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$ *running in at most* T *sequential steps, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following probability is bounded by* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{c} \mathsf{Aggr}(\mathsf{pp}, \mathcal{Z}, i, \mathcal{Z}_i, \pi_i) \neq \mathcal{Z} \wedge \\ (i, \cdot, \mathcal{Z}_i, \pi_i) \notin \mathcal{Q} \end{array}\middle|\begin{array}{c} (\mathsf{pp}, \mathcal{Z}) \xleftarrow{R} \mathsf{Setup}(1^\lambda, T) \wedge \\ (i, \mathcal{Z}_i, \pi_i) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Seal}}(\mathsf{pp}, \cdot, \cdot)}(\mathsf{pp}, \mathcal{Z}) \end{array}\right]$$

where $\mathcal{O}_{\mathsf{Seal}}(\mathsf{pp}, \cdot, \cdot)$ *is an oracle which takes as input any* $j \in [n]$ *and* $s_j \in \mathcal{X}$ *and outputs* $(\mathcal{Z}_j, \pi_j) \xleftarrow{R} \mathsf{Seal}(\mathsf{pp}, j, s_j)$*, and* $\mathcal{Q}$ *is the set of queries and responses* $(j, s_j, \mathcal{Z}_j, \pi_j)$ *to the oracle.*

A note on anonymity. We consider user anonymity an orthogonal problem. In the applications we have in mind, users can increase their anonymity by using zero-knowledge mixers [PSS19] or other privacy-enhancing overlays, e.g., zero-knowledge sets [Eth19]. Additionally, users can decouple their identities from their ballots by applying a verifiable shuffle [Nef01], although the on-chain verification of a shuffle proof might be prohibitively costly for larger elections. In Section 8.1 we describe how our protocols can be extended to achieve bid privacy even after the election ends, thus disclosing nothing besides a user's (non-)participation.

5 The Cicada framework

5.1 Efficient vector encoding for HTLPs

In many voting schemes, a ballot consists of a vector indicating the voter's relative preferences or point allocations for all m candidates. To avoid solving many HTLPs, it is desirable to encode this vector into a single HTLP, which requires representing the vector as a single integer.

Definition 6 (Packing scheme)**.** *A setup algorithm* PSetup *and pair of efficiently computable bijective functions* (Pack, Unpack) *is called a* packing scheme *and has the following syntax:*

- $\mathsf{PSetup}(\ell, w) \to \mathsf{pp}$*. Given a vector dimension* ℓ *and maximum entry* w*, output public parameters* pp*.*
- $\mathsf{Pack}(\mathsf{pp}, \mathbf{a}) \to s$*. Encode* $\mathbf{a} \in (\mathbb{Z}^+)^\ell$ *as a positive integer* $s \in \mathbb{Z}^+$*.*
- $\mathsf{Unpack}(\mathsf{pp}, s) \to \mathbf{a}$*. Given* $s \in \mathbb{Z}^+$*, recover a vector* $\mathbf{a} \in (\mathbb{Z}^+)^\ell$*.*

For correctness *we require* $\mathsf{Unpack}(\mathsf{Pack}(\mathbf{a})) = \mathbf{a}$ *for all* $\mathbf{a} \in (\mathbb{Z}^+)^\ell$*.*

The classic approach to packing [Gro05, HS00] uses a *positional numeral system (PNS)* to encode a vector of entries bounded by w as a single integer in base $M := w$(see Construction 1 below). Instead, we will set $M := nw+1$ to accommodate the homomorphic addition of all n users' vectors: each voter submits a length-m vector with entries $\leq w$. Summing over n voters, the result is a length-m vector with a maximum entry value nw; to prevent overflow, we set $M = nw + 1$.

Construction 1 (Packing from Positional Numeral System)**.**

- $\mathsf{PSetup}(\ell, w) \to M$: *Return* $M := w + 1$*.*
- $\mathsf{Pack}(M, \mathbf{a}) \to s$: *Output* $s := \sum_{j=1}^{|\mathbf{a}|} a_j M^{j-1}$*.*
- $\mathsf{Unpack}(M, s) \to \mathbf{a}$: *Let* $\ell := \lceil \log_M s \rceil$*. For* $j \in [\ell]$*, compute the* j*th entry of* $\mathbf{a}$ *as* $a_j := s \mod M^{j-1}$*.*

We also introduce an alternative approach in Construction 2 which is based on the *residue numeral system (RNS)*. The idea of the RNS packing is to interpret the entries of $\mathbf{a}$ as prime residues of a single unique integer s, which can be found efficiently using the Chinese Remainder Theorem (CRT). In other words, for all $j \in [\ell]$, s captures a_j as $s \bmod p_j$.

Construction 2 (Packing from Residue Numeral System)**.**

- $\mathsf{PSetup}(\ell, w) \to \mathbf{p}$: *Let* $M := w + 1$ *and sample* ℓ *distinct primes* $p_1, \ldots, p_\ell$ *s.t.* $p_j \geq M \ \forall j \in [\ell]$*. Return* $\mathbf{p} := (p_1, \ldots, p_\ell)$*.*
- $\mathsf{Pack}(\mathbf{p}, \mathbf{a}) \to s$*: Given* $\mathbf{a} \in (\mathbb{Z}^+)^\ell$*, use the CRT to find the unique* $s \in \mathbb{Z}^+$ *s.t.* $s \equiv a_j \pmod{p_j} \ \forall j \in [\ell]$*.*
- $\mathsf{Unpack}(\mathbf{p}, s) \to \mathbf{a}$*: return* $(a_1, \ldots, a_\ell)$ *where* $a_j \equiv s \mod p_j \ \forall j \in [\ell]$*.*

A major advantage of this approach is that, in contrast to the PNS approach, which is only homomorphic for SIMD (single instruction, multiple data) addition, the RNS encoding is fully SIMD homomorphic: the sum of vector encodings $\sum_{i\in[n]} s_i$ encodes the vector $\mathbf{a}_+ = \sum_{i\in[n]} \mathbf{a}_i$, and the product $\prod_{i\in[n]} s_i$ encodes the vector $\mathbf{a}_\times = \prod_{i\in[n]} \mathbf{a}_i$. Note that as in the PNS approach, we set $M = nw + 1$ to accommodate homomorphic addition of submissions; homomorphic multiplication, however, would require $M = w^n + 1$, and the primes in $\mathbf{p}$ would therefore be larger as well. Although the RNS has found application in error correction [KPT+22, TC14], side-channel resistance [PFPB19], and parallelization of arithmetic computations [AHK17, BDM06, GTN11, VNL+20], to our knowledge it has not been applied to voting schemes. We show in Section 6.2 that RNS is in fact a natural fit for some voting schemes, e.g., quadratic voting, leading to more efficient proofs of ballot correctness.

5.2 Our framework

We present Cicada, our framework for non-interactive private auctions/elections, in Figure 2. Cicada can be applied to voting and auction schemes where the scoring function $\Sigma = f \circ t$ has a linear tally function t. The framework is instantiated with a linear HTLP (Section 3.3), vector packing scheme (Section 5.1), and matching NIZK to ensure correctness of submissions by proving both the well-formedness of the puzzle and the solution's membership in $\mathcal{X}$.

Theorem 1. *Given a linear scoring function* Σ*, a secure NIZKPoK* NIZK*, a secure* HTLP*, and a packing scheme* (PSetup, Pack, Unpack)*, the Cicada protocol* Π_Σ *(Figure 2) is a secure time-locked voting/auction protocol.*

Intuitively, submission privacy follows from the security of the HTLP and zero-knowledge of the NIZK used: the submission can't be opened before time T and none of the proofs leak any information about it. Non-malleability is enforced by requiring the NIZK to be a proof of knowledge and including the user's identity i in the instance to prove, e.g., including it in the hash input of the Fiat-Shamir transform. This prevents a malicious actor from replaying a different user's ballot correctness proof. We delegate the full proof to Appendix E.

As we will see next, this captures many common schemes such as cumulative voting and sealed-bid auctions. We note that Cicada introduces a crucial design choice via the packing parameter $\ell \in [m]$, which defines a storage-computation trade-off that we detail in Section 7.2.

The Cicada Framework

Let $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ be an linear voting/auction scheme where $\mathcal{X} = [0, w]^m$, HTLP a linear HTLP, $T \in \mathbb{N}$ a time parameter representing the election/auction length, and (PSetup, Pack, Unpack) a packing scheme. Let NIZK be a NIZKPoK for submission correctness (language depends on Σ, HTLP; see Section 6.2) and PoE a proof of exponentiation(see Section 6.1).

$\mathsf{Setup}(1^\lambda, T, \ell) \xrightarrow{R} (\mathsf{pp}, \mathcal{Z})$. Set up the public parameters $\mathsf{pp}_{\mathsf{NIZK}} \xleftarrow{R} \mathsf{NIZK.Setup}(1^\lambda)$, $\mathsf{pp}_{\mathsf{tlp}} \xleftarrow{R} \mathsf{HTLP.Setup}(1^\lambda, T)$, and $\mathsf{pp}_{\mathsf{pack}} \leftarrow \mathsf{PSetup}(\ell, w)$. Let $\mathcal{Z} = \{Z_j\}_{j \in [m/\ell]}$ where $Z_j \xleftarrow{R} \mathsf{HTLP.Gen}(0)$. Output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{tlp}}, \mathsf{pp}_{\mathsf{pack}}, \mathsf{pp}_{\mathsf{NIZK}})$ and $\mathcal{Z}$.

$\mathsf{Seal}(\mathsf{pp}, i, \mathbf{v}_i) \xrightarrow{R} (\mathcal{Z}_i, \pi_i)$. Parse $\mathbf{v}_i := \mathbf{v}_{i,1} || \ldots || \mathbf{v}_{i,m/\ell}$. Compute $Z_{i,j} \leftarrow \mathsf{HTLP.Gen}(\mathsf{Pack}(\mathbf{v}_{i,j}))\ \forall j \in [m/\ell]$ and $\pi_i \leftarrow \mathsf{NIZK.Prove}((i, \mathcal{Z}_i), \mathbf{v}_i)$. Output $(\mathcal{Z}_i := \{Z_{i,j}\}_{j \in [m/\ell]}, \pi_i)$

$\mathsf{Aggr}(\mathsf{pp}, \mathcal{Z}, i, \mathcal{Z}_i, \pi_i) \to \mathcal{Z}'$. If $\mathsf{NIZK.Verify}((i, \mathcal{Z}_i), \pi_i) = 1$, update $\mathcal{Z}$ to $\mathcal{Z} \boxplus \mathcal{Z}_i$.

$\mathsf{Open}(\mathsf{pp}, \mathcal{Z}) \to (\mathcal{S}, \pi_{\mathsf{open}})$. Parse $\mathcal{Z} := \{Z_j\}_{j \in [m/\ell]}$ and solve for the encoded tally $\mathcal{S} = \{s_j\}_{j \in [m/\ell]}$ where $s_j \leftarrow \mathsf{HTLP.Solve}(Z_j)$. Prove the correctness of the solution(s) as $\pi_{\mathsf{open}} \leftarrow \mathsf{PoE.Prove}(\mathcal{S}, \mathcal{Z}, 2^T)$ and output $(\mathcal{S}, \pi_{\mathsf{open}})$.

$\mathsf{Finalize}(\mathsf{pp}, \mathcal{Z}, \mathcal{S}, \pi_{\mathsf{open}}) \to \{y, \bot\}$. If $\mathsf{PoE.Verify}(\mathcal{S}, \mathcal{Z}, 2^T, \pi_{\mathsf{open}}) \neq 1$, return $\bot$. Otherwise, parse $S := \{s_j\}_{j \in [m/\ell]}$ and let $\mathbf{v} := \mathbf{v}_1 || \ldots || \mathbf{v}_{m/\ell}$, where $\mathbf{v}_j \leftarrow \mathsf{Unpack}(s_j)\ \forall j \in [m/\ell]$. Output y such that $y = \Sigma(\mathbf{v})$.

Figure 2: The Cicada framework for non-interactive private auctions and elections.

Additive voting. Many common voting schemes are "additive", meaning each ballot (a length-m vector) is simply added to the tally, and a finalization function f is applied to the tally after the voting phase has ended to determine the winner. Additive voting schemes include first-past-the-post (FPTP), approval, range, and cumulative voting. Simple ranked-choice voting schemes, e.g., Borda count [Eme13], are also additive, differing only in what qualifies as a "proper" ballot (restrictions on vector entry domain, vector norm, etc.; see Table 2). Thus, we can use Cicada to instantiate private voting protocols for all these schemes.

Sealed-bid auctions. The Cicada framework can also be used to implement a sealed-bid auction with a number of HTLPs which is independent of the number of participants n. Assuming bids are bounded by M, we use an HTLP with solution space $\mathcal{X}$ such that $|\mathcal{X}| > M^n$. Each user i submits $Z_i \leftarrow \mathsf{HTLP.Gen}(bid_i)$ and π_i, where π_i proves $0 \leq bid_i \leq M$. A packing of the bids is computed at aggregation time, with Aggr updating Z to $Z \boxplus (M^{i-1} \cdot Z_i)$. After the bidding phase, the final "tally" is opened to s^* and the bids are recovered as $\mathsf{Bids} := \{s^* \bmod M^{i-1}\}_{i \in [n]}$. Any payment and allocation function can now be computed over the bids; in the simplest case, the winner is $\arg\max_i(\mathsf{Bids})$ and their payment is $\max_i(\mathsf{Bids})$. Notice that the full set of bids is revealed after the auction concludes. This cannot be avoided when using Cicada with linear HTLPs, since $\max_i$ is a nonlinear function, i.e., it cannot be computed it homomorphically.

Locking up collateral is necessary for every (private) auction scheme. We treat the problem of collateral lock-up as an important but orthogonal problem and refer to [TAF+23] for an extensive discussion.

6 Ballot/bid correctness proofs

6.1 Proof of Solution

During the finalization phase of our protocol, any party can solve the final HTLP off-chain and submit a solution to the contract. To enforce the correctness of this solution we require the solver to include a proof of the following relation:

$$\mathcal{R}_{\mathsf{PoS}} = \{((h, y, u, v, w \in \mathbb{G}, s \in \mathbb{Z}); \bot) : w = u^{2^T} \wedge\ v = wy^s \in \mathbb{G}\} \tag{3}$$

This can be realized as the conjunction of two proofs of exponentiation [Pie18, Wes19] for $w = u^{2^T}$ and $y^s = v/w$. In more detail, a Proof of Exponentiation (PoE) [Pie18, Wes19] is a proof for the following relation:

$$\mathcal{R}_{\mathsf{PoE}} = \{((u, w \in \mathbb{G}, x \in \mathbb{Z}); \bot) : w = u^x \in \mathbb{G}\}$$

Note that there is no witness in the $\mathcal{R}_{\mathsf{PoE}}$ relation, i.e., the verifier knows the exponent x. The primary goal of the PoE proof system for the verifier is to outsource a possibly large exponentiation in a group $\mathbb{G}$ of unknown order.

Wesolowski's proof of exponentiation protocol (PoE)

Public parameters: $\mathbb{G} \xleftarrow{R} GGen(\lambda)$.
Public inputs: $u, w \in \mathbb{G}, x \in \mathbb{Z}$.
Claim: $u^x = w$.

1. $\mathcal{V}$ sends $l \xleftarrow{R} \mathsf{Primes}(\lambda)$ to $\mathcal{P}$.
2. $\mathcal{P}$ computes $q = \lfloor \frac{x}{l} \rfloor \in \mathbb{Z} \wedge r \in [l]$, where $x = ql + r$. $\mathcal{P}$ sends $Q = u^q \in \mathbb{G}$ to $\mathcal{V}$.
3. $\mathcal{V}$ computes $r = x \bmod l$.

$\mathcal{V}$ accepts iff $w = Q^l u^r$.

Observe that the verifier sends a prime number as a challenge. When we make this protocol non-interactive via the Fiat-Shamir transform, we use a standard $\mathsf{HashToPrime}(\cdot)$ function to generate the correct challenge for the prover. In our implementation, we use the Baillie-PSW primality test [PSW80] to show that a randomly hashed challenge is indeed prime.

6.2 Proofs of well-formedness

To prove that HTLP ballots are well-formed during the submission phase, we will use several different proofs of knowledge about TLP solutions. We assume HTLPs of the form $(u, v) = (g^r, h^r y^s) \in \mathbb{G}_1 \times \mathbb{G}_2$, where $\mathbb{G}_1, \mathbb{G}_2$ are groups of unknown order. This captures all known constructions of HTLPs: in the case of the Paillier HTLP (Construction 3), $\mathbb{G}_1 = \mathbb{J}_N$, $\mathbb{G}_2 = \mathbb{Z}^*_{N^2}$, $h = (g^{2^T})^N$, and $y = 1 + N$. For the exponential ElGamal HTLP (Construction 5), $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{Z}^*_N$, $h = g^{2^T}$, and $y \in \mathbb{G}_1$. And for the class group HTLP [TCLM21], $\mathbb{G}_1, \mathbb{G}_2$ are cyclic subgroups of the respective class groups $Cl(\Delta_K), Cl(q^2\Delta_K)$, respectively, $h = \psi_q(G^{2^T})$ where G is a generator of $\mathbb{G}_1$ and $\psi_q : Cl(\Delta_K) \to Cl(q^2\Delta_K)$ is an injective map, and $y \in \mathbb{G}_2$ is the generator of a subgroup in which the discrete logarithm problem is easy (see [TCLM21] for details). Most of our protocols make use of the fact that for such HTLPs, v has the same structure as a Pedersen commitment [Ped92].

Since we are operating in groups of unknown order, to circumvent the impossibility result of [BCK10] and achieve negligible soundness error for Schnorr-style sigma protocols, we assume access to some public element(s) of $\mathbb{G}_1, \mathbb{G}_2$ whose representations are unknown. We prove security assuming $\mathbb{G}_1, \mathbb{G}_2$ are generic groups output by some randomized algorithm $GGen(\lambda)$. For more on instantiating Schnorr-style protocols in groups of unknown order while maintaining negligible soundness error, see [BBF19].

Well-formedness and knowledge of solution. To prove knowledge of a puzzle solution in zero-knowledge, our starting point is the folklore Schnorr-style protocol for knowledge of a Pedersen-committed value. Our protocol zk-PoKS is shown below.

zkPoK of TLP solution (zk-PoKS)

Public parameters: $\mathbb{G}_1, \mathbb{G}_2 \xleftarrow{R} GGen(\lambda)$, $b > 2^{2\lambda}|\mathbb{G}_i|\ \forall i \in \{1, 2\}$, and $g \in \mathbb{G}_1, h, y \in \mathbb{G}_2$.
Public input: HTLP $Z = (u, v)$.
Private input: $s, r \in \mathbb{Z}$ such that $Z = (g^r, h^r y^s)$.

1. $\mathcal{P}$ samples $\alpha, \beta \xleftarrow{R} [-b, b]$ and sends $A := h^\alpha y^\beta, B := g^\alpha$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a challenge $e \xleftarrow{R} [2^\lambda]$.
3. $\mathcal{P}$ computes $w = re + \alpha$ and $x = se + \beta$, which it sends to $\mathcal{V}$.

$\mathcal{V}$ accepts iff the following hold:

$$v^e A = h^w y^x$$
$$u^e B = g^w$$

Equality of solutions. Again, our starting point is the folklore protocol of equality of Pedersen-committed values: given two HTLPs with second terms v_1, v_2, if the solutions are equal the quotient is $v_1/v_2 = h^{r_1 - r_2}$. To prove the equality of the solutions, it therefore suffices to show knowledge of the discrete logarithm of v_1/v_2 with respect to h using Schnorr's classic sigma protocol [Sch89] with the previously described adjustments. Because of its simplicity we do not explicitly write out the protocol, which we will refer to as zk-PoSEq.

Binary solution. In an FPTP (or majority) vote for $m = 2$ candidates, users only need to prove that their ballot $(g^r, h^r y^s)$ encodes 0 or 1. More formally, users prove the statement $(u = g^r \wedge v = h^r) \vee (u = g^r \wedge vy^{-1} = h^r)$. This can be proved using the OR-composition [CDS94] of two discrete logarithm equality proofs [CP92] with respect to bases g and h and discrete logarithm r. A similar proof strategy could be applied if the user has multiple binary choices, e.g., approval and range voting. The OR-composition of multiple discrete logarithm equality proofs yields a secure ballot correctness proof for those voting schemes.

Positive solution. We use Groth's trick [Gro05], based on the classical Legendre three-square theorem from number theory, to show that a puzzle solution s is positive by showing that $4s + 1$ can be written as the sum of three squares. Our protocol deals only with the second component of the TLP, making use of the proof of solution equality (zk-PoSEq) described above and a proof that a TLP solution is the square of another (zk-PoKSqS, described next).

Proof of positive solution (zk-PoPS)

Public parameters: $\mathbb{G}_2 \xleftarrow{R} GGen(\lambda)$, a secure HTLP, and $h, y \in \mathbb{G}_2$.
Public input: $v \in \mathbb{G}_2$ such that $(\cdot, v) \in \mathsf{Im}(\mathsf{HTLP.Gen})$.
Private input: $s, r \in \mathbb{Z}$ such that $v = h^r y^s$ and $s > 0$.

1. Find three integers $s_1, s_2, s_3 \in \mathbb{Z}$ such that $4s + 1 = s_1^2 + s_2^2 + s_3^2$ and, for each $j = 1, 2, 3$, compute two HTLPs:

$$Z_j \leftarrow \mathsf{HTLP.Gen}(s_j)$$
$$Z'_j \leftarrow \mathsf{HTLP.Gen}(s_j^2)$$

2. Use zk-PoKSqS to compute a proof σ_j of square solution for each pair (Z_j, Z'_j) for $j = 1, 2, 3$.
3. Use zk-PoSEq to compute a proof σ_{eq} of solution equality for $4 \cdot Z \boxplus 1$ and $Z'_1 \boxplus Z'_2 \boxplus Z'_3$.

The full proof consists of $(\sigma_1, \sigma_2, \sigma_3, \sigma_{\mathsf{eq}})$, all computed with the same challenge $e \in [2^\lambda]$.

Square solution. To prove that a puzzle solution is the square of another, we use a conjunction of two zk-PoKS variants which proves knowledge of the same solution with respect to different bases. In particular, we consider only the second terms $v_1 = h^{r_1} y^s$ and $v_2 = h^{r_2} y^{s^2}$. We use the fact that v_2 can be rewritten as $h^{r_2 - r_1 s} v_1^s$ and prove that its opening w.r.t. base v_1 equals the opening of v_1.

Proof of square solution (zk-PoKSqS)

Public parameters: $\mathbb{G}_2 \xleftarrow{R} GGen(\lambda)$, $b > 2^{2\lambda}|\mathbb{G}_2|$, and $h, y \in \mathbb{G}_2$.
Public input: $v_1, v_2 \in \mathbb{G}_2$.
Private input: $s, r_1, r_2 \in \mathbb{Z}$ such that $v_1 = h^{r_1} y^s$ and $v_2 = h^{r_2} y^{s^2} = h^{r_2 - r_1 s} v_1^s$.

1. $\mathcal{P}$ samples $\alpha_1, \alpha_2, \beta \xleftarrow{R} [-b, b]$ and sends $A_1 := h^{\alpha_1} y^{\beta}, A_2 := h^{\alpha_2} v_1^{\beta}$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a challenge $e \xleftarrow{R} [2^{\lambda}]$.
3. $\mathcal{P}$ computes $w_1 = r_1 e + \alpha_1, w_2 = (r_2 - r_1 s)e + \alpha_2$, and $x = se + \beta$, which it sends to $\mathcal{V}$.

$\mathcal{V}$ accepts iff the following hold:

$$v_1^e A_1 = h^{w_1} y^x$$
$$v_2^e A_2 = h^{w_2} v_1^x$$

Quadratic voting [LW18]. Each voter i submits two linear HTLPs: Z_i^{tally} containing s_i and Z_i^{norm} containing s_i^2, where s_i is an encoding of the ballot $\mathbf{b}_i$. Z_i^{tally} will be accumulated into the running tally as usual, and Z_i^{norm} will be used to enforce the norm bound. A well-formed sealed ballot is therefore of the form $Z_i = (Z_i^{\mathsf{tally}}, Z_i^{\mathsf{norm}})$ such that:

Check #1. The vector entries enclosed in Z_i^{norm} are the squares of those enclosed in Z_i^{tally}.

Check #2. Z_i^{norm} has ℓ_1 norm strictly equal to w.[1]

The first check is much simpler and more efficient when using RNS packing. Recall that with this packing, a solution s encodes the ballot $(b_1, \ldots, b_m)$ as $s \mod p_j \equiv b_j \ \forall j \in [m]$, and that this encoding is fully SIMD homomorphic. It follows that $s^2 \mod p_j \equiv b_j^2$ for all $j \in [m]$.[2] With the RNS packing it therefore suffices to prove a square relationship *once* for the puzzles encoding s and s^2 (e.g., using zk-PoKSqS) rather than m times for all the vector entries. This is in contrast to the PNS packing used by all previous private voting schemes in the literature, where the absence of a multiplicative homomorphism would require proving the square relationship for every vector entry *individually*.

Regardless of the vector encoding, the second check is more involved: the user needs to open a sum of vector entries (the residues) without revealing the entries (residues) themselves. One approach is for the user to commit to each vector entry in Z_i^{norm}, i.e., $a_{ij} = s_i^2 \mod p_j$, with a Pedersen commitment, and use a variant proof of knowledge of exponent modulo p_j (PoKEMon [BBF19]) to show the commitments contain the appropriate values a_{ij}. Then, it can open the sum of the commitments. PoKEMon proofs are batchable, so the contract can verify them efficiently and check that the sum of the commitments opens to w.

7 Performance evaluation

We evaluate three instantiations of our Cicada framework (Section 5): binary voting, cumulative voting, and a sealed-bid auction. We chose these schemes as they are the most often deployed in today's blockchain ecosystems [Ope23, Opt, XWY+21]. We use the PNS packing system because, for these schemes, it results in more efficient NIZKs than RNS.

[1] We make this stricter requirement to simplify the norm check. Note that voters should be incentivized to submit such votes, since it maximizes their voting power.

[2] Assuming $s_j^2 < p_j$ for all j, which in our case will hold regardless, we set each $p_j < nw$ to avoid overflow when adding ballots and $s_j^2 \le w < nw$.

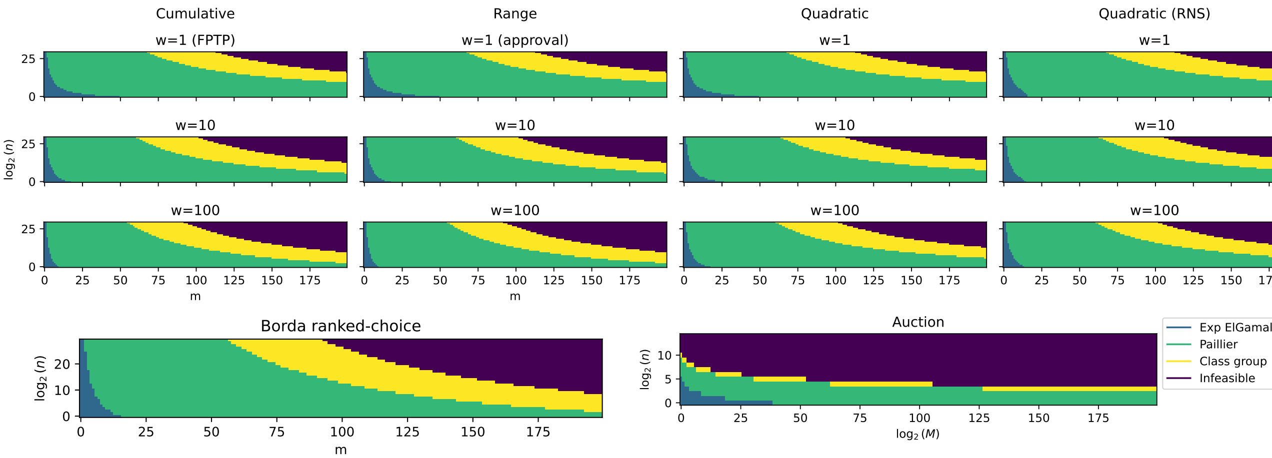

Figure 3: Most efficient HTLP construction for voting and auction using Cicada with maximal packing (using PNS except where indicated).

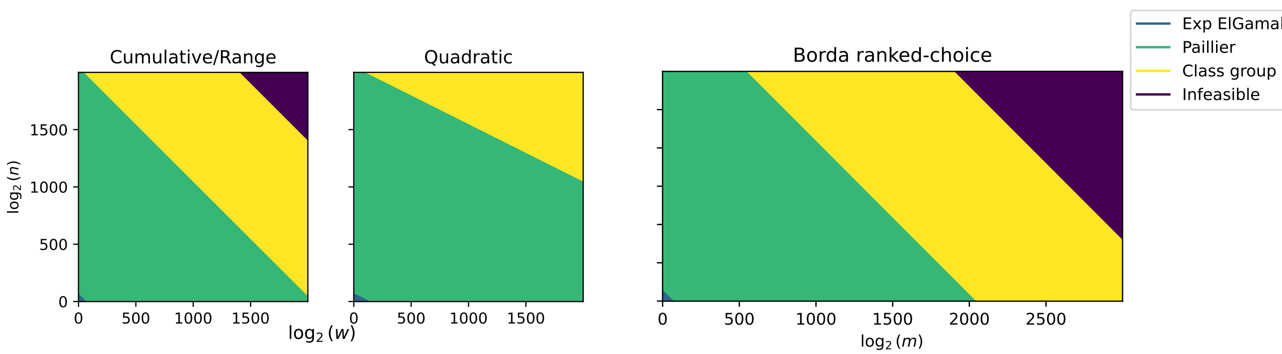

Figure 4: Most efficient HTLP construction for voting schemes using Cicada without packing. For the sealed-bid auction, the bid bit-length directly determines the HTLP construction to use (exponential ElGamal up to 80 bits, then Paillier up to 2048, and class group up to 3400).

7.1 Choice of HTLP construction

In this section, we evaluate the practicality and optimality of various HTLP constructions based on the parameters M, n, m, w of the auction or vote. Assuming the classic PNS packing, we require $(nw+1)^m \leq |\mathbb{G}|$ for voting and $M^n \leq |\mathbb{G}|$ for auctions, where $\mathbb{G}$ is the group in which the HTLP is instantiated. We show the optimal HTLP construction for auctions and voting for various parameter settings in Figure 3 (with packing) and Figure 4 (without packing). We use the security parameter $\lambda = 80$ (see discussion in Section 7.2), which corresponds to a 1024-bit modulus N for exponential ElGamal and Paillier HTLPs and 3400-bit discriminants for class group HTLPs. For the exponential ElGamal HTLP, we fixed the maximum ballot at 2^{80}, which corresponds to $\approx 2^{40}$ brute-forcing work using Pollard's rho algorithm [Pol78].

Exponential ElGamal HTLP (Construction 5) This is the most efficient HTLP construction: for a given security parameter, it has the smallest required cryptographic groups and most efficient group operations. However, since the puzzle solution is encoded in the exponent, solving the puzzle requires brute-forcing a discrete logarithm. This limits the use of this construction to a small set of parameter settings: assuming the largest discrete logarithm an off-chain solver can be expected to brute-force has τ bits, we require $(nw+1)^m \leq 2^{2\tau}$.

Paillier HTLP (Construction 3) This is a slightly less efficient construction since the size of the HTLPs for a given security parameter is doubled due to working over mod N^2 instead of mod N. This increases both the required storage and the complexity of the group operation. On the other hand, due to its larger solution space, the Paillier HTLP supports much broader parameter settings for a given security parameter.

Class group HTLP Class group offer the sole HTLP construction without a trusted setup [TCLM21]. This comes at the cost of the largest groups for a given security parameter. Class groups are not widely supported by major cryptographic libraries, and their costly group operation makes blockchain deployment difficult. We are unaware of any class group implementations for Ethereum smart contracts.

Impractical parameter settings Accommodating very large settings of n, w, m, M requires larger groups, leading to group operations and storage requirements which are intolerably inefficient for certain applications.

7.2 Implementation

We implemented our transparent on-chain coordinator as an Ethereum smart contract in Solidity.[3] For efficiency, we use the exponential ElGamal HTLP with a 1024-bit modulus N. To enable 1024-bit modular arithmetic in $\mathbb{Z}_N^*$, we developed a Solidity library which may be of independent interest. This size of N corresponds to approximately $\lambda = 80$ bits of security. Although this security level is no longer deemed cryptographically safe, the secrecy of the HTLP solutions is only guaranteed up to time T regardless, so this security level will suffice for our use case as long as the best-known factoring attack takes at least T time. A 2012 estimate for factoring 1024-bit integers is about a year [BHL12], which is significantly longer than the typical submission period of a decentralized auction or election.

The main factors influencing gas cost (see Section 7.3) are submission size, correctness proof size, and verification complexity. These factors mainly depend on the packing parameter $\ell \in [m]$, which determines a storage-computation trade-off with the following extremes:

One aggregate HTLP for all. If $\ell = m$, the contract maintains a single aggregate HTLP Z. This greatly reduces the on-chain space requirements of the resulting voting or auction scheme at the expense of typically more complex and larger submission correctness proofs.

One aggregate HTLP per candidate. If $\ell = 1$, the contract must maintain m aggregate HTLPs $\{Z_j\}_{j\in[m]}$. This increases the on-chain storage, but the submissions of correctness proofs become smaller and cheaper to verify.

In Section 7.3, we empirically explore this trade-off space by measuring the gas costs of various deployments of our framework with a range of parameter settings M, n, m, w, ℓ.

First, we briefly describe the proof systems used for each scheme we implement; detailed descriptions are given in Section 6.

Binary voting. In a binary vote (i.e., approval voting with $m = 1$), such as a simple yes/no referendum, users prove that the submitted ballot $Z = (u, v)$ is an exponential ElGamal HTLP with solution 0 or 1: $(u = g^r \wedge v = h^r) \vee (u = g^r \wedge vy^{-1} = h^r)$. This is achieved via OR-composition [CDS94] of two sigma protocols for discrete logarithm equality [CP92].

Cumulative voting. In cumulative voting, each user distributes w votes among m candidates. To accommodate a larger number of candidates, our implementation keeps m tally HTLPs Z_j, one for each candidate (in other words, $\ell = 1$). Each voter i submits m ballots $Z_{ij} = (g^{r_{ij}}, h^{r_{ij}} y^{s_{ij}})$ for all $j \in [m]$. Besides proving (using the protocol zk-PoKS) that each HTLP is well-formed (the same r_{ij} is used in both terms), the voter must prove that $0 \leq s_{ij}\ \forall\ j \in [m]$ and $\sum_{j=1}^{m} s_{ij} = w$. The first condition is shown with a proof of positive solution (zk-PoPS) via Legendre's three-square decomposition theorem [Gro05]. As a building block, we use a proof of square solution (zk-PoKSqS) to show that a puzzle solution is a square. The second condition is proven by providing the randomness $R_i = \prod_j r_{ij}$ which opens $\prod_j Z_{ij}$ to w.

[3]Open-sourced at https://github.com/a16z/cicada.

	Cumulative vote ($\ell = 1$)					
m	2	3	4	5	6	
Aggr	$3,391,514$	$5,081,542$	$6,781,389$	$8,489,786$	$10,208,185$	
Finalize	$269,505$	$397,789$	$521,895$	$644,814$	$770,934$	
	Sealed-bid auction ($\ell = 1$)					Sealed-bid auction ($\ell = b$)
b	8	10	12	14	16	any
Aggr	$3,586,022$	$4,488,050$	$5,394,047$	$6,304,164$	$7,218,905$	3,055,107
Finalize	$1,005,208$	$1,253,119$	$1,497,760$	$1,749,489$	$2,003,282$	$147,634$

Table 3: Gas costs for Cicada cumulative voting and sealed-bid auctions with various numbers of candidates m, bid bit-lengths b (max. bid $M = 2^{b-1}$), and packing parameters ℓ.

Sealed-bid auction. To illustrate two extremes of the packing spectrum, we implement two flavors of sealed-bid auctions. The first uses a single aggregate HTLP as described in Section 5 (this can be viewed as $\ell = b$, where $b = \lceil \log_2(M) \rceil$ is the bit-length of a bid): Bidder i submits a single HTLP $Z_i = (g^{r_i}, h^{r_i} y^{s_i})$, proving well-formedness with zk-PoKS and two zk-PoPS to show $0 \leq s \leq M$. The coordinator aggregates the ith bidder's bid by adding $M^{i-1} \cdot Z_i$ to its tally.

The second approach applies b aggregate HTLPs (i.e., $\ell = 1$): Each bidder i submits b HTLPs $\{Z_{ij}\}_{j \in [b]}$ and uses the same proof system as in binary voting to prove their well-formedness, i.e., the user inserted for each bit of the bid 0 or 1. The coordinator adds $2^i \cdot Z_{ij}$ to each corresponding aggregate HTLP Z_j.

7.3 Empirical Performance Evaluation

Submission costs. The on-chain cost of submitting a bid/ballot is the cost of running the Aggr function by the contract, i.e., the verification of the well-formedness proofs plus adding the users' submissions to the tally HTLPs (if and only if they verify). We report our measurements without packing (i.e., $\ell = 1$) in Table 3. Submitting a binary vote ballot costs $418,358$ gas (≈ 11.02 USD).[4] For cumulative voting, the submission cost scales linearly in m: with $m = 2$ candidates, submitting a ballot costs $3,391,514$ gas (≈ 94.49 USD), and each additional candidate adds $\approx 1,699,847$ gas (≈ 44.79 USD).

An auction with a single HTLP for each bit of the bid (the $\ell = 1$ case) requires a submission cost of $3,586,022$ gas (≈ 94.49 USD) for an 8-bit bid. Every additional bit in the submitted bid burns $\approx 451,014$ gas (≈ 11.89 USD).

On the other hand, if one applies packing, i.e., $\ell = b$, then the cost of submitting a sealed bid is constant at $3,055,107$ gas (≈ 80.50 USD). As seen in Table 3, with bid-space $M = 2^7$ it is already more economical to have a single aggregate HTLP and use a packing scheme, despite more complex bid-correctness proofs.

Finalization costs. Our voting and auction schemes end with solving the tally HTLP(s) off-chain, i.e., computing $(g^r)^{2^T} (= h^r)$. With exponential ElGamal, solving the puzzle also requires a brute-force discrete logarithm computation by the off-chain solver. The correctness of this computation is proven to the contract with Wesolowski's PoE [Wes19](recalled in Section 6). The Finalize cost comes from verifying the PoE(s) on-chain, which burns $101,066$ gas (≈ 2.66 USD) per proof. Without packing, the untrusted solver must provide a Wesolowski proof per tally HTLP, so the Finalize gas cost is linear in the number of tally HTLPs, as evidenced by Table 3. A portion of the Wesolowki verification cost comes from checking that the challenge is a prime number. In our implementation, the prover provides a Baillie-PSW [PSW80] primality certificate, whose verification cost is $44,972$ gas (≈ 1.18 USD).

Verification costs We implemented the sigma protocols described in Section 6 in Solidity and report their verification costs in Table 4. Recall that with Groth's trick [Gro05] in the proof of positivity (zk-PoPS), we must to decompose the integer solution into the sum of only three squares. Therefore, the gas cost of verifying zk-PoPS is equal to the cost of verifying three proofs of knowledge of square solutions (zk-PoKSqS) and one proof of knowledge of equal solution (zk-PoSEq).

In the short-term, deploying on Layer 2 (L2) already brings these costs down by 1–2 orders of magnitude. For

[4]We can estimate gas costs for approval voting using the cost of binary voting, as the former uses a disjunction of m copies of the same NIZK and thus scales linearly.

Sigma protocol	Verification gas cost
Proof of Exponentiation (PoE [Wes19])	$101,066$
PoK of solution (zk-PoKS)	$266,096$
Proof of solution equality (zk-PoSEq)	$336,155$
Proof of square solution(zk-PoKSqS)	$336,168$
Proof of positive solution (zk-PoPS)	$1,351,958$

Table 4: EVM gas costs of verification for the proof systems described in Section 6.

example, when deploying our implementation on the Optimism L2 rollup, casting a binary vote would cost less than US$0.30. Further optimizations (e.g., Karatsuba multiplication [KO62], batched Wesolowski proof verification [Rot21], or verification via efficient zkSNARKs [Gro16, GWC19]) can bring the costs down even more.

8 Extensions

This section introduces extensions to the Cicada framework that may be useful in future applications.

8.1 Everlasting ballot privacy for HTLP-based protocols

The basic Cicada framework does not guarantee long-term ballot privacy. Submissions are public after the Open stage. This is because users publish their HTLPs on-chain: once public, the votes contained in the HTLPs are only guaranteed to be hidden for the time it takes to compute T sequential steps, after which point it is plausible that someone has computed the solution. In many applications, it is desirable that individual ballots remain hidden *even after voting has ended* since the lack of everlasting privacy may facilitate coercion and vote-buying. As mentioned in Section 4, this can be achieved modularly by first decoupling the ballots from their voters via a privacy-enhancing overlay. Alternatively, we describe how the Seal procedure can be modified to prevent the opening of individual ballots, achieving everlasting privacy.

Observe that all known *efficient HTLP constructions* are of the form $(u, v) = (g^r, h'^r X)$,[5] where the solution is encoded in X and recovering it requires recomputing $h^r = (g^r)^{2^T}$ via repeated squaring of the first component. Our insight is that the puzzle information-theoretically hides the solution X without the first component. Importantly, publishing g^r is not necessary in any of our HTLP-based voting protocols *except as a means to verifiably compute the first component of the final HTLP*, i.e., $g^R = g^{\sum_{i\in[n]} r_i}$. The observation that g^R can be computed *without* revealing the individual values g^{r_i} enables us to construct the first practical and private voting protocols that guarantee *everlasting* ballot privacy with a single on-chain round.

For simplicity, consider a protocol in which both the ballot of user i and the tally consists of a single HTLP, respectively $Z_i = (g^{r_i}, h^{r_i} X_i)$ and $Z = (g^R, h^R X)$. Observe that for everlasting ballot privacy, updates to Z must inherently be batched: a singleton update $\mathsf{Aggr}(\mathsf{pp}, Z, Z_i, \pi) \to (g^{R+r_i}, h^{R+r_i} Y)$ (for some Y) would reveal $g^{r_i} = g^{R+r_i}/g^R$, which is the opening information to Z_i, as the quotient of the first component of Z after and before the update. Hence, the ballot X_i of user i would be recoverable in T sequential steps, i.e., after computing $h^{r_i} = (g^{r_i})^{2^T}$.

Batching ballot submissions off-chain in groups of k allows parties to achieve everlasting privacy as long as at least one party is honest. The parties aggregate their submissions off-chain as $(g^R, h^R X) = (\prod_i g^{r_i}, \prod_i h^{r_i} X_i)$ and compute a proof π_{batch} of well-formedness in a distributed-prover zero-knowledge proof protocol [DPP+21]. We use the observation that the individual second components v_i are hiding to optimize the batching by computing $h^R X$ in the clear, See Figure 5 for the details of the protocol.

This idea opens up a new design space for the MPC protocol used for batching, such as doing the randomness generation in a preprocessing phase instead, allowing dynamic additions to the anonymity set, optimizing the batch proof generation, and dealing with parties who fail to submit. We leave the full exploration of this large design space and related questions to future work.

[5] In the exponential ElGamal case, $h' = h$, while in the Paillier construction, $h' = h^N$ (see Appendix A.1). We will drop the tickmark on h' in the remainder of this section to avoid notational clutter.

Off-chain batching

Public parameters: A semiprime N and $h, y \in \mathbb{Z}_N^*$, a voting scheme $\Sigma : \mathcal{X}^n \to \mathcal{Y}$.

Let $P_1, \dots, P_k$ be a group of $k < n$ parties with addresses $\mathsf{addr}_1, \dots, \mathsf{addr}_k$ wishing to batch their ballots $(u_i, v_i) := (g^{r_i}, h^{r_i} X_i)$.

1. Each party broadcasts v_i. Now, every party can compute $v := \prod_i v_i = h^R X$, which encodes the sum of their submissions.
2. The parties use an $k-1$ malicious-secure MPC protocol [DKL+13, Kel20] on inputs u_i to compute $u := \prod_i u_i = g^R$.
3. They also compute two distributed-prover zero-knowledge proofs [DPP+21] in the MPC: (i) a discrete logarithm equality proof π_R that $\mathsf{dlog}_g(u) = \mathsf{dlog}_h(v)$ with distributed witness R, and (ii) a submission correctness proof π_s that the aggregated solution s encoded in X is consistent with the sum of k valid submissions, i.e., $s \in k \cdot \mathcal{X}$. Let $\pi_{\mathsf{batch}} = (\pi_R, \pi_s)$.
4. Finally, each party signs the final aggregated submission $Z_{\mathsf{batch}} = (u, v)$.

Output: $(Z_{\mathsf{batch}}, \pi_{\mathsf{batch}}, \{\mathsf{addr}_1, \dots, \mathsf{addr}_k\}, \{\sigma_1, \dots, \sigma_k\})$.

On-chain batched ballot submission

Public parameters: Cicada public parameters pp.

1. The designated party P_1 submits $Z_{\mathsf{batch}}, \pi_{\mathsf{batch}}, \{\mathsf{addr}_1, \dots, \mathsf{addr}_k\}, \{\sigma_1, \dots, \sigma_k\}$ to the tallying contract, which verifies the proofs and signatures, and adds (u, v) to the tally HTLP Z as in the basic protocol.
2. If P_1 doesn't submit by time $T - \tau$, any other party in the batch group can submit instead.

Figure 5: The on- and off-chain ballot batching protocols that $k < n$ parties can use to achieve everlasting ballot privacy.

8.2 Succinct ballot-correctness proofs

Real-world elections often have hundreds of candidates, e.g., Optimism's retroactive public good funding [Opt]. However, the state-of-the-art ballot correctness proofs [BBCG+23, Gro05] for all voting schemes (e.g., majority, approval voting, etc.) are linear in the number of candidates, rendering these schemes impractical in the blockchain setting. To counter these issues, we design constant-size ballot correctness proofs with constant verification time at the expense of an added preprocessing phase. The high-level idea is as follows. All correct ballots (e.g., $\{\mathsf{Pack}(s) : s \in \{0,1\}^m\}$ in the case of approval voting) are inserted into an accumulator or polynomial commitment (PC) [KZG10] during a transparent preprocessing phase. When users submit their votes $Z \xleftarrow{R} \mathsf{HTLP.Gen}(s)$, they prove in zero-knowledge that Z encodes a correct ballot, i.e., the users show that the solution s of Z had been previously inserted into the accumulator or PC with a succinct (blinded) membership proof [ZBK+22]. We detail our succinct ballot-correctness proof using the KZG commitment in Appendix C.

8.3 Coercion-resistance

Lastly, we briefly outline how one could add coercion resistance [JCJ05] to our framework. In the e-voting literature, there are two main pathways to obtaining coercion resistance: receipt-freeness or allowing unlikable revotes. Receipt-freeness seems challenging to achieve in the blockchain context, and we leave it to future work. Therefore, we follow the revoting paradigm akin to Lueks et al. [LQAT20]. One can allow indistinguishable revotes as follows. We could store our ballots in a zero-knowledge set (e.g., Semaphore is a readily available implementation of this concept for Ethereum [Eth19]). Additionally, a Merkle tree of nullifiers would be also stored on-chain containing the ballots that are revoked due to revoting. Whenever users want to revote, they could prove in zero-knowledge that they revoke a previous ballot that they inserted in the zero-knowledge set while they reveal the accompanying nullifier and insert it into the nullifier tree. We leave it to future work to flash out the technical details and implementation of this important added feature.

9 Related work

The cryptographic literature on both voting schemes and sealed-bid auctions is enormous, dating to the 1990s. However, most of these schemes are unsuitable for a fully decentralized and trust-minimized setting due to their inefficiency or reliance on trusted parties, i.e., tally authorities, servers running the public bulletin board, auctioneers, etc. Below, we review auction and voting protocols that use a blockchain as the public bulletin board.

Voting. The study of voting schemes for blockchain applications dates to at least 2017, when McCorry et al. [MSH17] proposed a "boardroom" voting protocol for DAO governance. The main disadvantage of their protocol is that the entire protocol can be aborted due to a single party. Groth [Gro05] and Boneh et al. [BBCG+23] develop techniques to create ballot correctness proofs for various voting schemes. These protocols all have proofs with size linear in the number of candidates. We break this barrier with the application of polynomial commitments and assuming a transparent, lightweight pre-processing phase. The application of HTLPs to voting was suggested when they were proposed by Malavolta and Thyagarajan [MT19]. However, they left the details of making such a protocol practical, secure, and efficient to future work. We aim to fill this gap with our techniques for various election types and our EVM implementation.

Auctions. Auctions are a natural fit for blockchains and were suggested as early as 2018 [GY18], albeit with a *trusted auctioneer*. Bag et al. introduced SEAL, a privacy-preserving sealed-bid auction scheme without auctioneers [BHSR19]. However, their protocol employs two rounds of communication, since they apply the Hao-Zielinski Anonymous Veto network protocol [HZ06]. Tyagi et al. proposed Riggs [TAF+23], a fair non-interactive auction scheme using timed commitments [FKPS21, §6]. This is perhaps the closest work to ours in implementing auctions (though not voting) in a fully decentralized setting using time-based cryptography, though their design does not utilize homomorphism to combine puzzles. As a result, gas costs are high and to achieve practicality Riggs relies on an optimistic second round in which users voluntarily open their puzzles. Chvojka et al. suggest a TLP-based protocol for both e-voting and auctions [CJSS21]. Their protocol has a per-auction trusted setup. In Appendix D, we propose a distributed setup protocol to reduce the trust assumption, which may be of independent interest.

Time-based cryptography. Time-based cryptography, which uses inherently sequential functions to delay the revelation of information, also has a lengthy history dating to Rivest, Shamir, and Wagner's proposal of time-lock encryption in 1996 [RSW96]. Numerous variants have emerged since then, including timed commitments [BN00], proofs-of-sequential-work [MMV13], VDFs [BBBF18], and homomorphic time-lock puzzles [MT19], which we employ here. For a recent survey, we refer the reader to Medley et al. [MLQ23]. The only practical work we know of taking advantage of HTLPs is Bicorn [CATB23], which builds a distributed randomness beacon with a single aggregate HTLP for an arbitrary number of entropy contributors.

10 Conclusion and Future Directions

In this work, we introduced Cicada, a framework for creating non-interactive private auction and voting schemes from HTLPs. Cicada is compatible with many popular schemes, including majority, approval, range, cumulative, ranked-choice, and quadratic voting as well as single-item sealed-bid auctions. We include a performance evaluation which shows that these schemes can be deployed todayon Ethereum, although with gas costs up to hundreds of dollars. When deployed on a Layer 2 chain, these costs decrease by 1–2 orders of magnitude to a few dollars, resulting in a promising paradigm for efficiently achieving voting and auctions with strong properties.

Acknowledgements. We thank Foteini Baldimtsi, Jeremy Clark, Brett Falk, and Elaine Shi for insightful discussions. This work was supported by a16z crypto research. Joseph Bonneau was additionally supported by DARPA Agreement HR00112020022 and NSF grant CNS-2239975. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA, a16z, or any other supporting organization.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security*, 2008. *[page 2.]*

[AHK17] Shahzad Asif, Md Selim Hossain, and Yinan Kong. High-throughput multi-key elliptic curve cryptosystem based on residue number system. *IET Computers & Digital Techniques*, 11(5), 2017. *[page 8.]*

[AM23] Aydin Abadi and Steven J. Murdoch. Decentralised Repeated Modular Squaring Service Revisited: Attack and Mitigation. Cryptology ePrint Archive, Paper 2023/1347, 2023. *[page 3.]*

[AOZZ15] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO*, 2015. *[page 2.]*

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, 2018. *[page 19.]*

[BBCG+23] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Arithmetic sketching. In *CRYPTO*, 2023. *[page 18.]*

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *CRYPTO*, 2019. *[pages 6, 10, 12, 26, 28, and 29.]*

[BCK10] Endre Bangerter, Jan Camenisch, and Stephan Krenn. Efficiency limitations for σ-protocols for group homomorphisms. In *TCC*, 2010. *[page 10.]*

[BCM05] Endre Bangerter, Jan Camenisch, and Ueli Maurer. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In *PKC*, 2005. *[page 6.]*

[BDJ+06] Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Crypto*, 2006. *[page 2.]*

[BDM06] Jean-Claude Bajard, Sylvain Duquesne, and Nicolas Méloni. *Combining Montgomery Ladder for Elliptic Curves Defined over $\mathbb{F}_p$ and RNS Representation*. PhD thesis, LIR, 2006. *[page 8.]*

[BF01] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. *Journal of the ACM (JACM)*, 48(4):702–722, 2001. *[page 6.]*

[BGG18] Sean Bowe, Ariel Gabizon, and Matthew D Green. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. In *Financial Crypto*, 2018. *[page 27.]*

[BHL12] Daniel J Bernstein, Nadia Heninger, and Tanja Lange. Facthacks: Rsa factorization in the real world. `https://www.hyperelliptic.org/tanja/vortraege/facthacks-29C3.pdf`, 12 2012. *[page 14.]*

[BHSR19] Samiran Bag, Feng Hao, Siamak F Shahandashti, and Indranil Ghosh Ray. Seal: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 15, 2019. *[page 18.]*

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *CRYPTO*, 2000. *[page 19.]*

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *STOC*, 1994. *[page 2.]*

[CATB23] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. *Financial Crypto*, 2023. *[page 19.]*

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994. *[pages 11 and 14.]*

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based e-voting scheme. In *PQC*, 2016. *[page 2.]*

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Eurocrypt*, 1997. *[page 5.]*

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable rsa modulus generation with a dishonest majority. In *IEEE Security and Privacy*, 2021. *[page 6.]*

[CJSS21] Peter Chvojka, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles. In *ESORICS*, 2021. *[pages 2, 18, 27, and 28.]*

[COPZ22] Melissa Chase, Michele Orrù, Trevor Perrin, and Greg Zaverucha. Proofs of discrete logarithm equality across groups. *Cryptology ePrint Archive*, 2022. *[page 27.]*

[CP92] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO*, 1992. *[pages 6, 11, and 14.]*

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, 1998. *[page 28.]*

[DK02] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In *Eurocrypt*, 2002. *[page 6.]*

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18*, pages 1–18. Springer, 2013. *[page 17.]*

[DM10] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed rsa key generation. In *TCC*, 2010. *[page 6.]*

[DPLNS17] Rafaël Del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In *ACM CCS*, 2017. *[page 2.]*

[DPP+21] Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any np statement jointly? efficient distributed-prover zero-knowledge protocols. *Cryptology ePrint Archive*, 2021. *[pages 16 and 17.]*

[EL03] Edith Elkind and Helger Lipmaa. Interleaving cryptography and mechanism design: The case of online auctions. Cryptology ePrint Archive, Paper 2003/021, 2003. *[page 1.]*

[Eme13] Peter Emerson. The original Borda count and partial voting. *Social Choice and Welfare*, 40:353–358, 2013. *[pages 4 and 9.]*

[Eth19] Ethereum Foundation. Semaphore: a zero-knowledge set implementation for ethereum, May 2019. *[pages 7 and 18.]*

[Exp] Privacy Scaling Explorations. Perpetual powers of tau. *[page 27.]*

[FG14] Jon Fraenkel and Bernard Grofman. The Borda Count and its real-world alternatives: Comparing scoring rules in Nauru and Slovenia. *Australian Journal of Political Science*, 49(2), 2014. *[page 4.]*

[FKPS21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In *TCC*, 2021. *[pages 6 and 18.]*

[FMW22] Robin Fritsch, Marino Müller, and Roger Wattenhofer. Analyzing voting power in decentralized governance: Who controls DAOs? *arXiv preprint arXiv:2204.01176*, 2022. *[page 1.]*

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Auscrypt*, 1993. *[page 2.]*

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO*, 1986. *[page 29.]*

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009. *[page 2.]*

[GG22] Aayush Gupta and Kobi Gurkan. Plume: An ecdsa nullifier scheme for unique pseudonymity within zero knowledge proofs. Cryptology ePrint Archive, Paper 2022/1255, 2022. *[page 1.]*

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, pages 467–482, 2005. *[pages 6, 8, 11, 14, 15, 18, and 30.]*

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Eurocrypt*, 2016. *[page 16.]*

[GTN11] Mahadevan Gomathisankaran, Akhilesh Tyagi, and Kamesh Namuduri. Horns: A homomorphic encryption scheme for cloud computing using residue number system. In *CISS*, 2011. *[page 8.]*

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. *[page 16.]*

[GY18] Hisham S. Galal and Amr M. Youssef. Verifiable sealed-bid auction on the ethereum blockchain. Cryptology ePrint Archive, Paper 2018/704, 2018. *[pages 1, 2, and 18.]*

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Eurocrypt*, 2000. *[page 8.]*

[HZ06] Feng Hao and Piotr Zieliński. A 2-round anonymous veto protocol. In *Security Protocols Workshop*, 2006. *[page 18.]*

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES*, 2005. *[page 18.]*

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020. *[page 17.]*

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. *[page 26.]*

[KO62] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk*, 145(2):293–294, 1962. *[page 16.]*

[KPT+22] Igor Anatolyevich Kalmykov, Vladimir Petrovich Pashintsev, Kamil Talyatovich Tyncherov, Aleksandr Anatolyevich Olenev, and Nikita Konstantinovich Chistousov. Error-correction coding using polynomial residue number system. *Applied Sciences*, 12(7):3365, 2022. *[page 8.]*

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Asiacrypt*, 2010. *[pages 18 and 27.]*

[LQAT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security*, 2020. *[page 18.]*

[LW18] Steven P Lalley and E Glen Weyl. Quadratic voting: How mechanism design can radicalize democracy. In *AEA Papers and Proceedings*, volume 108, 2018. *[pages 4 and 12.]*

[MLQ23] Liam Medley, Angelique Faye Loe, and Elizabeth A. Quaglia. SoK: Delay-based Cryptography. In *Computer Security Foundations*, 2023. *[page 19.]*

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *ITCS*, 2013. *[page 19.]*

[MSH17] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *Financial Crypto*, 2017. *[page 18.]*

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *CRYPTO*, 2019. *[pages 2, 5, 18, 19, 24, 25, and 27.]*

[Nef01] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM CCS*, 2001. *[page 7.]*

[NRBB24] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In *ACNS*, 2024. *[page 28.]*

[Ope23] OpeanSea. How to sell an NFT, June 2023. *[pages 1 and 12.]*

[Opt] Optimism. Optimism RetroPGF 2: Badgeholder manual. Accessed 2023-09-18. *[pages 12 and 18.]*

[Opt23] Optimism. What is the Optimism Collective?, February 2023. *[page 1.]*

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999. *[pages 5 and 24.]*

[PE] Privacy and Scaling Explorations. Rate-limiting nullifier. `https://rate-limiting-nullifier.github.io/rln-docs/`. *[page 1.]*

[PE23] Privacy and Scaling Explorations. Maci: Minimal anti-collusion infrastructure. `https://maci.pse.dev/`, 2023. *[pages 1 and 2.]*

[Ped92] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992. *[page 10.]*

[PFPB19] Louiza Papachristodoulou, Apostolos P Fournaris, Kostas Papagiannopoulos, and Lejla Batina. Practical evaluation of protected residue number system scalar multiplication. In *CHES*, 2019. *[page 8.]*

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In *ITCS*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. *[page 9.]*

[Pol78] John M Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978. *[page 13.]*

[Pre04] Drazen Prelec. A bayesian truth serum for subjective data. *science*, 306(5695):462–466, 2004. *[page 25.]*

[PRF23] Mallesh Pai, Max Resnick, and Elijah Fox. Censorship resistance in on-chain auctions. *arXiv preprint arXiv:2301.13321*, 2023. *[page 1.]*

[PSS19] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution version 1.4. *Tornado cash privacy solution version*, 1, 2019. *[page 7.]*

[PSW80] Carl Pomerance, John L Selfridge, and Samuel S Wagstaff. The pseudoprimes to 25 10ˆ9. *Mathematics of Computation*, 35(151):1003–1026, 1980. *[pages 10 and 15.]*

[Rot21] Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In *TCC*, 2021. *[page 16.]*

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical Report 486, Massachusetts Institute of Technology. Laboratory for Computer Science, 1996. *[pages 2, 4, 5, and 19.]*

[San99] Tomas Sander. Efficient accumulators without trapdoor extended abstract. In *Information and Communication Security*, 1999. *[page 6.]*

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989. *[pages 6, 11, and 28.]*

[SY03] Koutarou Suzuki and Makoto Yokoo. Secure Generalized Vickrey Auction Using Homomorphic Encryption. In *Financial Cryptography*, 2003. *[page 1.]*

[TAF+23] Nirvan Tyagi, Arasu Arun, Cody Freitag, Riad Wahby, Joseph Bonneau, and David Mazières. Riggs: Decentralized sealed-bid auctions. In *ACM CCS*, 2023. *[pages 9 and 18.]*

[TC14] Thian Fatt Tay and Chip-Hong Chang. A new algorithm for single residue digit error correction in redundant residue number system. In *ISCAS*, 2014. *[page 8.]*

[TCLM21] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabian Laguillaumie, and Giulio Malavolta. Efficient cca timed commitments in class groups. In *ACM CCS*, 2021. *[pages 6, 10, and 14.]*

[TGB+21] Sri Aravinda Krishnan Thyagarajan, Tiantian Gong, Adithya Bhat, Aniket Kate, and Dominique Schröder. Opensquare: Decentralized repeated modular squaring service. In *ACM CCS*, 2021. *[page 3.]*

[Tha23] Justin Thaler. Proofs, arguments, and zero-knowledge, July 2023. *[page 6.]*

[VNL+20] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020. *[page 8.]*

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Eurocrypt*, 2019. *[pages 9, 15, and 16.]*

[XWY+21] Pengcheng Xia, Haoyu Wang, Zhou Yu, Xinyu Liu, Xiapu Luo, and Guoai Xu. Ethereum name service: the good, the bad, and the ugly. *arXiv preprint arXiv:2104.05185*, 2021. *[pages 1 and 12.]*

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *ACM CCS*, 2022. *[pages 18 and 27.]*

A Extended Preliminaries

In this section, we detail the preliminaries that we could not include in the main body of the paper due to space constraints.

A.1 HTLP Constructions

Malavolta and Thyagarajan [MT19] give two HTLP constructions with linear and multiplicative homomorphisms, respectively. They require N to be a *strong* semiprime, i.e., $N = p \cdot q$ such that $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are also prime. The linearly-homomorphic HTLP is based on Paillier encryption [Pai99], while the multiplicative homomorphism is achieved by working over the subgroup $\mathbb{J}_N \subseteq \mathbb{Z}_N^*$ of elements with Jacobi symbol $+1$. We recall their constructions in Figure 6.

Correctness of the linear HTLP holds because for all $s \in \mathbb{Z}_N$ and $Z = (u, v) \leftarrow \mathsf{HTLP.Gen}(\mathsf{pp}, s)$,

$$\mathsf{HTLP.Open}(\mathsf{pp}, Z) = \frac{(v/(h^R)^N \mod N^2) - 1}{N} = \frac{((1+N)^s) - 1}{N} = s \tag{4}$$

since $(1+N)^x = 1+Nx \mod N^2$. Correctness of the homomorphism follows since for all linear functions $f(x_1, x_2) = b + a_1x_1 + a_2x_2$ and all $Z_i = (u_i, v_i) \in \mathsf{Im}(\mathsf{HTLP.Gen}(\mathsf{pp}, s_i; r_i))$ for $i \in \{1, 2\}$,[6]

$$\begin{aligned}
\mathsf{HTLP.Eval}(\mathsf{pp}, f, Z_1, Z_2) &= (u_1^{a_1} \cdot u_2^{a_2}, (1+N)^b \cdot v_1^{a_1} \cdot v_2^{a_2}) \\
&= (g^{r_1a_1} \cdot g^{r_2a_2}, \quad (1+N)^b \cdot h^{r_1Na_1} \cdot (1+N)^{s_1a_1} \cdot h^{r_2Na_2} \cdot (1+N)^{s_2a_2}) \\
&= (g^{r_1a_1+r_2a_2}, \quad h^{(r_1a_1+r_2a_2)\cdot N} \cdot (1+N)^{b+s_1a_1+s_2a_2}) \\
&= \mathsf{HTLP.Gen}(\mathsf{pp}, f(s_1, s_2); r_1a_1 + r_2a_2)
\end{aligned}$$

which opens to $f(s_1, s_2)$ by eq. (4).

The multiplicative HTLP operates over the solution space $\mathbb{J}_N$ (instead of $\mathbb{Z}_N$). It is easy to see that $\mathsf{HTLP.Open}(\mathsf{pp}, \mathsf{HTLP.Gen}(\mathsf{pp}, s)) = s$ for all $s \in \mathbb{Z}_N^*$. Furthermore, for all $f(x_1, x_2) = ax_1x_2$ and all $Z_i = (u_i, v_i) \in \mathsf{Im}(\mathsf{HTLP.Gen}(\mathsf{pp}, s_i; r_i))$ for $i \in \{1, 2\}$,

$$\begin{aligned}
\mathsf{HTLP.Eval}(\mathsf{pp}, f, Z_1, Z_2) &= (u_1 \cdot u_2 \mod N, a \cdot v_1 \cdot v_2 \mod N) \\
&= (g^{r_1} g^{r_2} \mod N, \quad h^{r_1} h^{r_2} \cdot as_1s_2 \mod N) \\
&= (g^{r_1+r_2} \mod N, \quad h^{r_1+r_2} \cdot as_1s_2 \mod N) \\
&= \mathsf{HTLP.Gen}(\mathsf{pp}, f(s_1, s_2); r_1 + r_2)
\end{aligned}$$

Thus correctness holds.

Lifting the multiplicative HTLP to put s in the exponent yields a more efficient linear HTLP for a small solution space $\mathcal{S} \subset \mathbb{Z}_N$, where $\mathcal{S} = \{s : s \in \mathbb{J}_N \wedge s \ll N\}$ (Figure 7, changes shown in). This can be viewed as a construction based on exponential ElGamal encryption over $\mathbb{Z}_N^*$.

[6] For space and clarity we drop the moduli and assume that we are working in the appropriate ring in each coordinate (namely $\mathbb{Z}_N$ and $\mathbb{Z}_{N^2}$, respectively).

Construction 3 (Linear HTLP [MT19].).

$\mathsf{HTLP.Setup}(1^\lambda, T) \xrightarrow{R} \mathsf{pp}$. *Sample a strong semiprime* N *and a generator* $g \xleftarrow{R} \mathbb{Z}_N^*$*, then compute* $h = g^{2^T} \mod N \in \mathbb{Z}_N^*$*. (This can be computed efficiently using the factorization of* N*). Output* $\mathsf{pp} := (N, g, h)$*.*

$\mathsf{HTLP.Gen}(\mathsf{pp}, s; r) \to Z$. *Given a value* $s \in \mathbb{Z}_N$*, use randomness* $r \in \mathbb{Z}_{N^2}$ *to compute and output*

$$Z := (g^r \mod N,\ h^{r \cdot N} \cdot (1+N)^s \mod N^2) \in \mathbb{J}_N \times \mathbb{Z}_{N^2}^*$$

$\mathsf{HTLP.Open}(\mathsf{pp}, Z, r) \to s$. *Parse* $Z := (u, v)$ *and compute* $w := u^{2^T} \mod N = h^r$ *via repeated squaring. Output* $s := \frac{(v/w^N \mod N^2) - 1}{N}$*.*

$\mathsf{HTLP.Eval}(\mathsf{pp}, f, Z_1, Z_2) \to Z$. *To evaluate a linear function* $f(x_1, x_2) = b + a_1 x_1 + a_2 x_2$ *homomorphically on puzzles* $Z_1 := (u_1, v_1)$ *and* $Z_2 := (u_2, v_2)$*, return*

$$Z = (u_1^{a_1} \cdot u_2^{a_2} \mod N, v_1^{a_1} \cdot v_2^{a_2} \cdot (1+N)^b \mod N^2).$$

Construction 4 (Multiplicative HTLP [MT19].).

$\mathsf{HTLP.Setup}(1^\lambda, T) \xrightarrow{R} \mathsf{pp}$. *Same as construction 3.*

$\mathsf{HTLP.Gen}(\mathsf{pp}, s; r) \to Z$. *Given a value* $s \in \mathbb{J}_N$*, use randomness* $r \in \mathbb{Z}_{N^2}$ *to compute and output*

$$Z := (g^r \mod N,\ h^r \cdot s \mod N) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$$

$\mathsf{HTLP.Open}(\mathsf{pp}, Z, r) \to s$. *Parse* $Z := (u, v)$ *and compute* $w := u^{2^T} \mod N = h^r$ *via repeated squaring. Output* $s := v/w$*.*

$\mathsf{HTLP.Eval}(\mathsf{pp}, f, Z_1, Z_2) \to Z$. *To evaluate a multiplicative function* $f(x_1, x_2) = a x_1 x_2$ *homomorphically on puzzles* $Z_1 := (u_1, v_1)$ *and* $Z_2 := (u_2, v_2)$*, return*

$$Z = (u_1 \cdot u_2 \mod N, a \cdot v_1 \cdot v_2 \mod N)$$

Figure 6: The HTLP constructions of [MT19].

B Additional voting and scoring protocols

B.1 Bayesian truth serum

Bayesian truth serum [Pre04] is a method for eliciting truthful subjective answers where objective truth does not exist or is not knowable. The core of the idea is to reward answers that are "surprisingly common" by leveraging respondents' own predictions of what will be common. Thus, for a question with many (mutually exclusive) potential answers, the score of user i responding $\mathbf{x}_i := (x_{i1}, \ldots, x_{im})$ and $\mathbf{y}_i := (y_{i1}, \ldots, y_{im})$ is calculated as

$$\mathsf{score}_i := \sum_{j \in [m]} x_{ij} \log \frac{\overline{x}_j}{\overline{y}_j} + \alpha \sum_{j \in [m]} \overline{x}_j \log \frac{y_{ij}}{\overline{x}_j} \tag{5}$$

where $\alpha > 0$ is a constant. The variable $x_{ij} \in \{0, 1\}$ denotes user i's decision (choose or don't choose) for option $j \in [m]$, $\overline{x}_j$ is the empirical frequency of choice j over all the users' answers, y_{ij} is user i's estimate of $\overline{x}_j$ (i.e., their estimate of the probability of answer j among all users), and $\overline{y}_j$ is the empirical (geometric) average of y_{ij} over all the users' answers. Since each user can only choose a single answer, x_{ij} will be 0 for all but one value of j, which we denote j^*. Thus, we can think of the equation above as equivalent to

$$x_{ij^*} \log \frac{\overline{x}_{j^*}}{\overline{y}_{j^*}} + \alpha \sum_{j \in [m]} \overline{x}_j \log \frac{y_{ij}}{\overline{x}_j}.$$

The first term is referred to as the *information score* and the second as the *prediction score*. The information score is highest when the user's choice k^* is "surprisingly common", i.e., when the empirical frequency of answer

Construction 5 (Efficient linear HTLP.)**.**

$\mathsf{HTLP.Setup}(1^\lambda, T) \xrightarrow{R} \mathsf{pp}$. *Output* $\mathsf{pp} := (N, g, h, y)$*, where* $y \xleftarrow{R} \mathbb{Z}_N^*$ *and the remaining parameters are the same as in constructions 3 and 4.*

$\mathsf{HTLP.Gen}(\mathsf{pp}, s; r) \to Z$. *Given a value* $s \in$*, use randomness* $r \in \mathbb{Z}_N$ *to compute and output*

$$Z := (g^r \mod N,\ h^r \cdot \mod N) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$$

$\mathsf{HTLP.Open}(\mathsf{pp}, Z, r) \to s$. *Parse* $Z := (u, v)$ *and compute* $w := u^{2^T} \mod N = h^r$ *via repeated squaring.*

$\mathsf{HTLP.Eval}(\mathsf{pp}, f, Z_1, Z_2) \to Z$. *To evaluate a homomorphically on puzzles* $Z_1 := (u_1, v_1)$ *and* $Z_2 := (u_2, v_2)$*, return*

Figure 7: Efficient linear HTLP for small solution space.

j^* ($\overline{x}_{j^*}$) is higher than the crowd's estimate of the empirical frequency of j^* ($\overline{y}_{j^*}$). Therefore participants are incentivized to submit their truthful responses, even (and especially) if they believe them to be uncommon.

The prediction score is the Kullback-Leibler divergence [KL51] between the user's estimate of the average answer and the true average answer, weighted by α. This is maximized when the two values are equal (i.e., the divergence is 0), and so incentivizes truthful reporting of y_{ij}, the user's estimate of $\overline{x}_j$.

We show how Bayesian truth serum can be implemented in the Cicada framework. First, rewrite Equation (5) as

$$\mathsf{score}_i := \sum_{j\in[m]} x_{ij}(\log \overline{x}_j - \overline{y}'_j) + \alpha \sum_{j\in[m]} \overline{x}_j(y'_{ij} - \log \overline{x}_j) \tag{6}$$

where $y'_{ij} = \log y_{ij}$ and $\overline{y}'_j = \log \overline{y}_j$. The smart contract will use two (lists of) HTLPs $\mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{x}}}, \mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{y}}'}$ to keep track of two running "tallies":

$$\overline{\mathbf{x}} = (\overline{x}_1, \ldots, \overline{x}_m) = \sum_i \mathbf{x}_i$$
$$\overline{\mathbf{y}}' = (\overline{y}'_1, \ldots, \overline{y}'_m) = \sum_i \frac{1}{n}\mathbf{y}'_i$$

Each user's ballot consists of the vectors $\mathbf{x}_i, \mathbf{y}'_i$, where $\mathbf{x}_i \in [0,1]^m$ has ℓ_1 norm 1 and $\mathbf{y}'_i = \log \mathbf{y} \in \mathbb{N}^m$ with $\sum_{j\in[m]} y_{ij} = n$. Assuming no packing for simplicity, the ballot is encoded as three lists of HTLPs: a list of linear HTLPs $\mathcal{Z}^{\mathsf{ans}}_i := \{Z^{\mathsf{ans}}_{ij}\}_{j\in[m]}$ for the entries of $\mathbf{x}_i$, and two lists of (respectively) linear and multiplicative HTLPs $\mathcal{Z}^+_i := \{Z^+_{ij}\}_{j\in[m]}$ and $\mathcal{Z}^\times_i := \{Z^\times_{ij}\}_{j\in[m]}$, both encoding the entries of $\mathbf{y}'_i$. The smart contract coordinator must ensure that the following hold:

Check #1a. All Z^{ans}_{ij} encode $x_{ij} \in [0,1]$.

Check #1b. $\sum_{j\in[m]} x_{ij} = 1$.

Check #2a. All Z^+_{ij} encode $y'_{ij} > 0$.

Check #2b. $\sum_{j\in[m]} 2^{y'_{ij}} = n$ (assuming log base 2).

Check #3. $Z^\times_{ij}$ contains the same value as Z^+_{ij} for all $j \in [m]$.

Most of these checks can be achieved using the protocols in Section 6: Check #1a with the binary solution protocol, #1b and #2b by providing randomness which opens the homomorphic sum to the correct value, and #2a with zk-PoPS. Check #2b additionally requires a zero-knowledge proof of exponentiation, e.g., [BBF19]. Because the puzzles to check in #3 use different constructions, we can't apply zk-PoSEq directly; instead, one can combine two zk-PoKS proofs with a standard PoK for discrete logarithm.

The aggregation algorithm $\mathsf{Aggr}((\mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{x}}}, \mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{y}}'}), i, \mathcal{Z}_i, \pi_i)$ updates the tally to $(\mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{x}}} \boxplus \mathcal{Z}^{\mathsf{ans}}_i, \mathcal{Z}^{\mathsf{tally}}_{\overline{\mathbf{y}}'} \boxplus \frac{1}{n} \cdot \mathcal{Z}^+_i)$. During the opening phase, anyone can solve for the final tallies $\overline{\mathbf{x}}_{\mathsf{final}}, \overline{\mathbf{y}}'_{\mathsf{final}}$ and the individual user submissions $\{(\mathbf{x}_i, \mathbf{y}'_i)\}_{i\in[n]}$. If correct, they are used in $\mathsf{Finalize}$ to compute the final set of scores as follows:

Preprocessing ballots for succinct ballot-correctness proofs

Public parameters: The common reference string $\mathsf{crs} := \{g_1^{\tau^j}\}_{j=1}^d \in \mathbb{G}_1^d$. A semiprime N and $h, y \in \mathbb{Z}_N^*$. A voting scheme $\Sigma : \mathcal{X}^n \to \mathcal{Y}$, e.g., approval voting.

1. Let $\mathcal{X}$ be the set of correct ballots and $|\mathcal{X}| = d$
2. Let $f(x) \in \mathbb{F}_p^{\leq d}[x]$ be a univariate polynomial s.t. $\forall s_i \in \mathcal{X} : f(i) := \mathsf{Pack}(s_i)$. The polynomial $f(x)$ could be computed using Lagrangian interpolation.
3. Let com be the KZG commitment to the polynomial f.

Output: com.

Figure 8: Preprocessing ballots to enable succinct ballot-correctness proofs.

1. Let $\overline{\mathbf{x}}' := \log \overline{\mathbf{x}}$. Compute $\mathbf{I}' := \overline{\mathbf{x}}' - \overline{\mathbf{y}}'$ and $\mathbf{P}' := \overline{\mathbf{x}} \cdot \overline{\mathbf{x}}'$.
2. For each user $i \in [n]$:
   (a) Compute i's information score $I_i := \sum_{j\in[m]} I_{ij}$, where $\mathbf{I}_i = (I_{i1}, \ldots, I_{im}) := \mathbf{x}_i \cdot \mathbf{I}'$.
   (b) Compute i's prediction score $P_i := \sum_{j\in[m]} P_{ij}$, where $\mathbf{P}_i = (P_{i1}, \ldots, P_{im}) := \overline{\mathbf{x}} \cdot \mathbf{y}_i' - \mathbf{P}'$.
   (c) User i's score is $I_i - P_i$.

C Succinct ballot-correctness proofs from polynomial commitments

In this section, we assume that a common reference string for the KZG polynomial commitment (PC) scheme [KZG10] is already available to users, namely $\mathsf{crs} := \{g_1^{\tau^j}\}_{j=1}^d$, where $g_1 \in \mathbb{G}_1$ is a generator in a bilinear pairing-friendly cyclic group $\mathbb{G}_1$ over $\mathbb{F}_p$ for some prime p, $\tau \xleftarrow{R} \mathbb{F}_p$ hidden to everyone. The crs is typically established during a sequential, secure multi-party computation (MPC), e.g., [BGG18].

Let us assume that users have established during a preprocessing phase (Figure 8) a short commitment com that encodes all the possible ballots in a particular voting scheme, e.g., $\mathcal{X} = [0,1]^m$ for approval voting. The size of classical proofs of well-formedness, e.g., OR-composition of sigma-protocols, scale linearly in the number of candidates m. The following proof strategy yields a constant-size proof of correctness for moderately-sized $\mathcal{X}$, i.e., $|\mathcal{X}| \leq d$[7].

First, given a ballot $Z = (g^r, h^r y^s) \in \tilde{\mathbb{G}}_1 \times \tilde{\mathbb{G}}_2$, the user creates an elliptic curve point $Z_1 = h_1^r y_1^s \in \mathbb{G}_1$ for random generators $h_1, y_1 \xleftarrow{R} \mathbb{G}_1$ in a pairing-friendly group. Using the discrete logarithm across different groups techniques developed in [COPZ22], the user can show that Z and Z_1 have the same discrete logarithms r and s with for their bases $h, y \in \mathbb{G}, h_1, y_1 \in \mathbb{G}_1$, respectively. Now that Z_1 and the polynomial commitment are in the same pairing-friendly group $\mathbb{G}_1$, the user can create a blinded KZG opening proof [ZBK+22] to prove ballot correctness. Specifically, the proof π shows that the value s in Z_1 matches an evaluation of the polynomial f committed by com at some (hidden) point j, i.e., $f(j) = s$. Note that the verifier only sees constant-size commitments of f, j, and s. Since the blinded KZG proof π is also constant-size, this strategy yields the first succinct ballot-correctness proofs for many common voting schemes, e.g., approval and range voting.

D A trusted setup protocol for the CJSS scheme

Chvojka, Jager, Slamanig, and Striecks [CJSS21] describe how to combine a public-key encryption scheme with a TLP to obtain a private voting or auction protocols which, unlike the HTLP-based approach suggested by [MT19], is "solve one, get many for free". The high-level idea of the protocol is to encrypt each user's bid with a common public key whose corresponding secret key is inserted into a TLP (see Figure 9). Therefore, none of the bids can be decrypted until the corresponding encryption secret key is obtained by solving the TLP. One drawback of this

[7] The largest KZG CRS we know of [Exp] is for $d = 2^{28}$, so in the case of $\mathcal{X} = [0,1]^m$ this strategy requires $m \leq 28$.

scheme, however, is that it requires an additional trusted setup procedure to create a TLP containing the secret key corresponding to the encryption public key used. Furthermore, unlike the HTLP approach, the setup cannot be reused and must be re-run for every protocol invocation.

The CJSS Framework

Let Π_{E} be a CCA-secure public-key encryption scheme, TLP a time-lock puzzle scheme, and $\Sigma : \mathcal{X}^n \to \mathcal{Y}$ a base voting/auction protocol.

$\mathsf{Setup}(1^\lambda, T) \xrightarrow{R} (\mathsf{pp}, \mathcal{Z})$. Sample a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_{\mathsf{E}}.\mathsf{Gen}(1^\lambda)$ and TLP parameters $\mathsf{pp}_{\mathsf{tlp}} \leftarrow \mathsf{TLP.Setup}(1^\lambda, T)$. Compute $Z_{\mathsf{sk}} \xleftarrow{R} \mathsf{TLP.Gen}(\mathsf{pp}_{\mathsf{tlp}}, \mathsf{sk})$ and return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{tlp}}, \mathsf{pk})$ and $\mathcal{Z} := (Z_{\mathsf{sk}}, \perp)$.

$\mathsf{Seal}(\mathsf{pp}, i, s) \xrightarrow{R} (\mathsf{ct}_i, \pi_i)$. Parse pk from pp and compute an encrypted bid/ballot as $\mathsf{ct}_i \leftarrow \Pi_{\mathsf{E}}.\mathsf{Enc}(\mathsf{pk}, s_i)$ along with a proof π_i that ct_i is a valid encryption under pk.

$\mathsf{Aggr}(\mathsf{pp}, \mathcal{Z}, \mathsf{ct}_i, \pi_i) \to \mathcal{Z}'$. Verify π_i. If the check passes, parse $\mathcal{Z} := (Z_{\mathsf{sk}}, \mathcal{C})$ and update to $\mathcal{Z}' := (Z_{\mathsf{sk}}, \mathcal{C} \cup \{\mathsf{ct}_i\})$.

$\mathsf{Open}(\mathsf{pp}, \mathcal{Z}, \perp) \to \mathsf{sk}$. Let $\mathcal{Z} := (Z_{\mathsf{sk}}, \mathcal{C})$ and publish $\mathsf{sk} \leftarrow \mathsf{HTLP.Solve}(\mathsf{pp}_{\mathsf{tlp}}, Z_{\mathsf{sk}})$.

$\mathsf{Finalize}(\mathsf{pp}, \mathsf{sk}) \to y$. Use the secret key sk to decrypt each ciphertext $\mathsf{ct}_i \in \mathcal{C}$ to $s_i \leftarrow \Pi_{\mathsf{E}}.\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_i)$. Compute the final result in the clear as $\Sigma(s_1, \ldots, s_n)$.

Figure 9: The "solve one, get many for free" paradigm (CJSS) [CJSS21].

We observe that, for encryption schemes with discrete-log key-pairs such as Cramer-Shoup [CS98], there is a natural decentralized setup protocol secure against all-but-one corruptions. Using the blockchain as a broadcast channel (similar to [NRBB24]), a simple sequential MPC protocol to set up the parameters works as follows. Suppose there is some smart contract that stores the public key $\mathsf{pk} = g^{\mathsf{sk}} \mod N$ and a TLP Z_{sk} containing sk (initially, one can set $\mathsf{sk} = 0$). Each contributor i can update pk by adding s_i homomorphically in the exponent and contributing an HTLP $Z_i = (g^{r_i} \mod N, h^{r_i \cdot N} \cdot (1+N)^{s_i})$. The contribution must be accompanied by a proof of well-formedness. For the previous state $\mathsf{pk}, Z_{\mathsf{sk}}$, contributor i proves that its contribution pk_i, Z_i passes the following checks:

Check #1. It knows the discrete logarithm of pk_i with respect to the base g. This can be achieved with a proof of knowledge of the exponent [Sch89].

Check #2. It knows the representation of the HTLP contribution Z_i with respect to the bases $g, h^N, (1+N)$ (i.e., the discrete logarithms r_i, r_i, s_i). This can be proven by a "knowledge of representation" proof system in groups of unknown order (e.g., the PoKE family of proofs [BBF19]; see Section 3.4).

Check #3. Finally, the discrete logarithms a, b, c from check #2 are such that $a = b$ and $c = \mathsf{dlog}_g(\mathsf{pk}_i)$.

The state is updated with the ith contribution iff all the checks pass. After the update, $Z_{\mathsf{sk}} := Z_{\mathsf{sk}} \cdot Z_i$ and $\mathsf{pk} := \mathsf{pk} \cdot \mathsf{pk}_i = g^{s+s_i}$. A single honest contributor suffices to guarantee a uniformly distributed keypair.

E Security Proofs

We use $\mathcal{D}_1 \approx_\lambda \mathcal{D}_2$ to denote that two distributions $\mathcal{D}_1, \mathcal{D}_2$ have statistical distance bounded by $\mathsf{negl}(\lambda)$.

E.1 The Cicada framework

Proof of Theorem 1. For simplicity, we give a proof for the simple case of $\mathcal{X} = [0, 1]$, i.e., submissions consist of a single bit, but our argument generalizes to larger domains $\mathcal{X}$. Let $n \in \mathbb{N}$ be the number of users.

The correctness of the Cicada framework (cf. Definition 3) follows by construction and from the correctness of the underlying building blocks (i.e., soundness in the case of NIZKs).

Next, we prove submission privacy. Let $\mathsf{ExpSPriv}^{\mathcal{A}}_{\Pi_\Sigma}(\lambda, T, i)$ be the original submission privacy game for the Cicada scheme Π_Σ with T-bounded adversary $\mathcal{A}$, cf. Definition 4. We define a series of hybrids to show that $\Pr[\mathsf{ExpSPriv}^{\mathcal{A}}_{\Pi_\Sigma}(\lambda, T, i) = 1] \leq \mathsf{negl}(\lambda)$ for all $\lambda, T \in \mathbb{N}$ and $i \in [n]$.

$\underline{\mathcal{H}_0}$: This is the original game $\mathsf{ExpSPriv}^{\mathcal{A}}_{\Pi_\Sigma}(\lambda, T, i)$, where $Z_i \leftarrow \mathsf{HTLP.Gen}(b)$ and $\pi_i \leftarrow \mathsf{NIZK.Prove}(i, Z_i, b)$.

$\underline{\mathcal{H}_1}$: Replace π with $\tilde{\pi} \leftarrow \mathsf{NIZK.Sim}(i, Z_i)$. $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_0$ by the zero-knowledge property of NIZK.

$\underline{\mathcal{H}_2}$: Replace Z_i with $Y_i \leftarrow \mathsf{HTLP.Gen}(1-b)$ and $\tilde{\pi}$ with $\tilde{\sigma} \leftarrow \mathsf{NIZK.Sim}(i, Y_i)$. $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable because the distributions $\{Z_i, \mathsf{Sim}(i, Z_i)\}$ and $\{Y_i, \mathsf{Sim}(i, Y_i)\}$ are indistinguishable since $\{Z_i\}, \{Y_i\}$ are indistinguishable by the security of HTLP.

$\underline{\mathcal{H}_3}$: Replace $\tilde{\sigma}$ with $\sigma \leftarrow \mathsf{NIZK.Prove}(i, Y_i, 1-b)$. $\mathcal{H}_3$ is indistinguishable from $\mathcal{H}_2$ by the zero-knowledge property of NIZK.

This series of hybrids implies $\Pr[b' = b] \approx_\lambda \Pr[b' = 1 - b]$, where b' is the output of $\mathcal{A}$ in $\mathcal{H}_0$ or $\mathcal{H}_3$, respectively. Therefore $\Pr[\mathsf{ExpSPriv}^{\mathcal{A}}_{\Pi_\Sigma}(\lambda, T, i) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Finally, we show that if NIZK is a PoK and HTLP is secure, then Cicada is non-malleable, cf. Definition 5. Suppose towards a contradiction that Cicada is malleable. We will use this and the fact that NIZK is a PoK to construct an adversary $\mathcal{B}$ which has non-negligible advantage in the HTLP security game. Again, we work in the simple case $\mathcal{X} = [0, 1]$, i.e., $m, \ell, w = 1$, but the argument generalizes to other parameter settings.

Since by our assumption Cicada is malleable, there exists $\mathcal{A}$ which outputs $(i, \cdot, \mathcal{Z}_i, \pi_i) \notin \mathcal{Q}$ such that $\mathsf{NIZK.Verify}((i, \mathcal{Z}_i), \pi) = 1$ with non-negligible probability. Given a puzzle Z_b containing some unknown bit b, $\mathcal{B}$ works as follows. First, it computes $(\mathsf{pp}, Z) \xleftarrow{R} \mathsf{Setup}(1^\lambda, T, 1)$ and sends them to the non-malleability adversary $\mathcal{A}$. $\mathcal{B}$ responds to $\mathcal{A}$'s oracle queries (j, b_j) with honestly computed (Z_j, π_j), keeping track of queries and responses in the set $\mathcal{Q}$. When $\mathcal{A}$ outputs (i, Z_i, π_i), $\mathcal{B}$ looks for $(i, b_i, Z_i, \pi_i) \in \mathcal{Q}$ and outputs b_i. Since $\mathcal{A}$ has non-negligible advantage, it follows that $\mathsf{NIZK.Verify}((i, Z_i), \pi_i) = 1$. This implies that either $\Pr[b_i = b] = \frac{1}{2} + \mathsf{negl}(\lambda)$ or NIZK is not knowledge sound. Both possibilities contradict our assumptions, namely that the HTLP is secure and the NIZK is knowledge sound. Thus, Cicada must be non-malleable. □

E.2 Sigma Protocols

We prove special-soundness and honest-verifier zero-knowledge (HVZK) of our sigma protocols (Section 6). Any such protocol can be made into a non-interactive zero-knowledge proof of knowledge (NIZKPoK) via the Fiat-Shamir transform [FS86].

Theorem 2 (zk-PoKS)**.** *The protocol* zk-PoKS *in Section 6.2 is a special sound and HVZK proof system in the generic group model.*

Proof. For special soundness, we show that given two distinct accepting transcripts with the same first message, i.e., (A, B, e, w, x) and (A, B, e', w', x') where $e \neq e'$, we can extract the witnesses r, s. The proof follows the blueprint of the proof of Theorem 10 in [BBF19]. Since the transcripts are accepting, we have

$$\begin{aligned} h^w y^x &= v^e A & \qquad h^{w'} y^{x'} &= v^{e'} A \\ &= h^{re+\alpha} y^{se+\beta} & &= h^{re'+\alpha} y^{se'+\beta} \end{aligned}$$

Combining the two equations we get

$$\begin{aligned} h^{r\Delta e} y^{s\Delta e} &= h^{\Delta w} y^{\Delta x} \\ \iff \qquad v^{\Delta e} &= h^{\Delta w} y^{\Delta x} \end{aligned} \tag{7}$$

where $\Delta e = e - e'$ and $\Delta y, \Delta x$ are defined similarly. Then with overwhelming probability, $r\Delta e = \Delta w$ and $s\Delta e = \Delta x$ (cf. Lemma 4 of [BBF19]), so $\Delta e \in \mathbb{Z}$ divides $\Delta w \in \mathbb{Z}$ and $\Delta x \in \mathbb{Z}$ and we can extract $r, s \in \mathbb{Z}$ as $r = \Delta w / \Delta e$ and $s = \Delta x / \Delta e$.

We will now show that these values are correct, i.e., $v = h^{\Delta w/\Delta e} y^{\Delta x/\Delta e}$. Assume towards a contradiction that this does not hold and $\mu = h^{\Delta w/\Delta e} y^{\Delta w/\Delta e} \neq v$. Since $\mu^{\Delta e} = v^{\Delta e}$ by Equation (7), this must mean that $(\mu/v)^{\Delta e} = 1$ and therefore $\mu/v \in \mathbb{G}_2$ is an element of order $\Delta e > 1$. Since Δe is easy to compute and μ/v is a non-identity element of $\mathbb{G}_2$, this contradicts the assumption that $\mathbb{G}_2$ is a generic group (specifically, it contradicts non-trivial order hardness [BBF19, Corollary 2]). We thus conclude that our extractor successfully recovers the witnesses r and s.

We still need to verify that the r^* we can extract from u will be consistent with the one extracted from v, i.e., $r^* = r$. Again we know

$$\begin{aligned} g^w &= u^e B & \qquad g^{w'} &= u^{e'} B \\ &= g^{r^* e + \alpha^*} & &= g^{r^* e' + \alpha^*} \end{aligned}$$

so by a similar argument $r^* = \Delta w / \Delta e$, which equals r. Thus the protocol satisfies special soundness.

To prove HVZK, we give a simulator which produces an accepting transcript $(\tilde{A}, \tilde{B}, \tilde{e}, \tilde{w}, \tilde{x})$ that is perfectly indistinguishable from an honest transcript (A, B, e, w, x). The simulator is quite simple: it samples $\tilde{e} \xleftarrow{R} [2^\lambda]$ identically to an honest verifier, then samples $\tilde{w}, \tilde{x} \xleftarrow{R} \mathbb{Z}$ and sets $\tilde{A} := h^{\tilde{w}} y^{\tilde{x}} v^{-\tilde{e}}$ and $\tilde{B} := g^{\tilde{w}} u^{-\tilde{e}}$. It follows by inspection that the transcript is an accepting one. Furthermore, notice that $\tilde{A}$ and $\tilde{B}$ are uniformly distributed in $\mathbb{G}_2$ and $\mathbb{G}_1$, respectively, just like A, B in the honest transcript. Also, both $\tilde{x}$ and x are uniform in $\mathbb{Z}$. Thus the simulated transcript is perfectly indistinguishable from an honest one. □

Theorem 3 (zk-PoKSqS)**.** *The protocol* zk-PoKSqS *in Section 6.2 is a special sound and HVZK proof system in the generic group model.*

Proof. For special soundness, we show that given two distinct accepting transcripts with the same first message, i.e., $(A_1, A_2, e, w_1, w_2, x)$ and $(A_1, A_2, e', w_1', w_2', x')$ where $e \neq e'$, we can extract the witnesses r_1, r_2, s. Notice that v_2 is not guaranteed to encode the square of s_1, so $v_2 = h^{r_2 - r_1 s_2 / s_1} v_1^{s_2/s_1}$. Let $\sigma_2 = s_2/s_1$ and $\rho_2 := r_2 - r_1 s_2/s_1 = r_2 - r_2\sigma_2$.

Using the same extractor as in the proof of Theorem 2, we can extract correct integers $r_1 = \Delta w_1/\Delta e$, $s_1 = \Delta x/\Delta e$, $\rho_2 = \Delta w_2/\Delta e$, and $\sigma_2 = \Delta x/\Delta e$. Notice $s_1 = \sigma_2$, which implies $\sigma_2 = s_1^2$. Finally we use $r_1, s_1, \rho_2 \in \mathbb{Z}$ to recover $r_2 := \rho_2 + r_1 s_1 \in \mathbb{Z}$. Thus the protocol is special sound.

To prove HVZK, we give a simulator which produces an accepting transcript $(\tilde{A}_1, \tilde{A}_2, \tilde{e}, \tilde{w}_1, \tilde{w}_2, \tilde{x})$ that is perfectly indistinguishable from an honest transcript $(A_1, A_2, e, w_1, w_2, x)$. The simulator is quite simple: it samples $\tilde{e} \xleftarrow{R} [2^\lambda]$ identically to an honest verifier, then samples $\tilde{w}_1, \tilde{w}_2, \tilde{x} \xleftarrow{R} \mathbb{Z}$ and sets $\tilde{A}_1 := h^{\tilde{w}_1} y^{\tilde{x}} v_1^{-\tilde{e}}$ and $\tilde{A}_2 := h^{\tilde{w}_2} v_1^{\tilde{x}} v_2^{-\tilde{e}}$. It follows by inspection that the transcript is an accepting one. Furthermore, notice that $\tilde{A}_1, \tilde{A}_2$ are uniformly distributed in $\mathbb{G}_2$, respectively, just like A_1, A_2 in the honest transcript. Also, both $\tilde{w}_1, \tilde{w}_2, \tilde{x}$ are uniform in $\mathbb{Z}$ just like w_1, w_2, x. Thus the simulated transcript is perfectly indistinguishable from an honest one. □

Theorem 4 (zk-PoPS)**.** *The protocol* zk-PoPS *in Section 6.2 is sound and HVZK.*

Proof. Soundness follows directly from the (knowledge) soundness of zk-PoKSqS and zk-PoSEq as well as Legendre's three-square theorem [Gro05].

For HVZK, note that an honest zk-PoPS transcript has the form $(\{A_{1,j}, A_{2,j}\}_{j\in[3]}, R, e, \{w_{1,j}, w_{2,j}, x_j\}_{j\in[3]})$, where (R, e, x) is an honest zk-PoSEq transcript and $(A_{1,j}, A_{2,j}, e, w_{1,j}, w_{2,j}, x_j)$ for $j = 1, 2, 3$ are honest zk-PoKSqS transcripts. Given the instance v, our zk-PoPS simulator first computes some random HTLPs $(\tilde{u}_j, \tilde{v}_j), (\tilde{u}_j', \tilde{v}_j') \xleftarrow{R}$ HTLP.Gen(0) for $j = 1, 2, 3$. These simulated underlying instances are indistinguishable from the honest instances an honest prover would use. This follows from the security of HTLP.

Next, our simulator samples $\tilde{e} \xleftarrow{R} [2^\lambda]$ identically to an honest verifier and uses the simulators of the proof systems, always with the same challenge $\tilde{e}$, to produce a simulated transcript:

$$(\tilde{A}_{1,j}, \tilde{A}_{2,j}, \tilde{e}, \tilde{w}_{1,j}, \tilde{w}_{2,j}, \tilde{x}_j) \leftarrow \mathsf{Sim}_{\mathsf{zk-PoKSqS}}(\tilde{v}_j, \tilde{v}_j'; \tilde{e}) \ \forall j = 1, 2, 3$$
$$(\tilde{R}, \tilde{e}, \tilde{x}) \leftarrow \mathsf{Sim}_{\mathsf{zk-PoSEq}}\left(\frac{4 \cdot v \boxplus 1}{\tilde{v}_1' \boxplus \tilde{v}_2' \boxplus \tilde{v}_3'}; \tilde{e}\right)$$

By HVZK of zk-PoKSqS and zk-PoSEq, these transcripts are accepting and indistinguishable from an honestly generated transcript. □