Towards Practical Zero-Knowledge Proof for PSPACE

Ashwin Karthikeyan
University of Toronto
ashwin@cs.toronto.edu

Hengyu Liu
University of Illinois Urbana–Champaign
hengyu2@illinois.edu

Kuldeep S. Meel
University of Toronto, Georgia Institute of Technology
meel@cs.toronto.edu

Ning Luo*
University of Illinois Urbana–Champaign
nl27@illinois.edu

Abstract

Efficient zero-knowledge proofs (ZKPs) have been restricted to NP statements so far, whereas they exist for all statements in PSPACE. This work presents the first practical zero-knowledge (ZK) protocols for PSPACE-complete statements by enabling ZK proofs of QBF (Quantified Boolean Formula) evaluation. The core idea is to validate quantified resolution proofs (Q-Res) in ZK. We develop an efficient polynomial encoding of Q-Res proofs, enabling proof validation through low-overhead arithmetic checks. We also design a ZK protocol to prove knowledge of a winning strategy related to the QBF, which is often equally important in practice. We implement our protocols and evaluate them on QBFEVAL. The results show that our protocols can verify 72% of QBF evaluations via Q-Res proof and 82% of instances' winning strategies within 100 seconds, for instances where such proofs or strategies can be obtained.

1 Introduction

Zero-knowledge proofs (ZKPs) enable one party (the prover) to prove to another party (the verifier) the validity of a statement without revealing any information beyond the statement [1]. Such power makes ZKP a fundamental tool for achieving privacy and verifiability in many computational settings, particularly since recent advances in efficient ZKP constructions have significantly improved their practicality. We have witnessed ZKP's deployment in many real-world applications, including authentication, scalable blockchain protocols, and privacy-preserving machine learning [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Most efforts to make ZKPs efficient have focused on NP statements, leaving statements of practical interest in PSPACE under-investigated.

We consider a simple two-player game between a prover and a verifier on a 2×2 grid where the coordinate of the upper left cell is $(0, 0)$, the upper right cell is $(1, 0)$, and the bottom right cell is $(1, 1)$. The prover starts at position $(0, 0)$ and aims to reach the goal at $(1, 1)$ in exactly two steps. The verifier may block either cell $(0, 1)$ or $(1, 0)$, but only after observing the prover's first move. The prover moves first, choosing either right or down, and then, after the verifier selects a cell to block, makes a second move. The prover wins if she reaches $(1, 1)$ without entering the blocked cell after the verifier blocked it. The goal of the prover is to convince the verifier that she knows a strategy to reach $(1, 1)$ *w*ithout revealing the strategy itself.

*Ning Luo is the corresponding author - nl27@illinois.edu

The interaction of the game can be encoded as a quantified Boolean formula (QBF):

$$\exists x_0\ \forall y\ \exists x_1.\ \texttt{Reachable}(x_1, y, x_0)$$

Here, $y \in \{0, 1\}$ denotes the verifier's blocking choice: $y = 1$ corresponds to blocking cell $(0, 1)$, and $y = 0$ corresponds to blocking $(1, 0)$. The variables x_0 and x_1 represent the prover's first and second moves, respectively, with 0 indicating a horizontal move and 1 indicating a vertical move. The predicate $\texttt{Reachable}(x_1, y, x_0)$ evaluates to true if the prover successfully reaches $(1, 1)$ without entering the blocked cell. For example, $\texttt{Reachable}(x_1 = 1, y = 0, x_0 = 0) = 1$, since the prover first moves right to $(1, 0)$, then the verifier blocks $(1, 0)$, and then the prover moves down to reach $(1, 1)$. The goal is reached successfully as the prover reached $(1, 1)$ and did not enter the blocked cell $(1, 0)$ after the verifier blocked it.

The QBF formula evaluates to true, as the prover has a strategy $(x_0 = 0, x_1 = 1)$ that guarantees reaching the goal regardless of the verifier's blocking choice. This can be easily verified since the verifier's blocking action occurs only after the prover moves to $(1, 0)$, and therefore, the blocked cell has no effect on the chosen path. However, in more complex winning configurations, determining whether the prover has a winning strategy is nontrivial. On the other hand, most such position-based games can be encoded as QBF instances, where the existence of a winning strategy corresponds to the truth of the QBF.

QBF evaluation is PSPACE-complete and extends propositional logic by allowing both existential ($\exists$) and universal ($\forall$) quantification over Boolean variables. The inclusion of quantified variables enables the modeling of uncertainty, such as adversarial or environmental actions. This level of expressiveness exceeds that of NP and naturally arises in many practical domains, including software verification, circuit synthesis, and AI robustness.

Establishing ZKPs for QBFs has both theoretical value and practical significance. Due to the expressiveness of QBFs, ZKPs of QBFs can enable privacy-preserving proofs for a broad class of applications currently beyond the scope of existing ZKP frameworks. Below, we highlight the following use cases that are representative of this gap, and we will also illustrate this in our evaluation (see Section 6):

Partial Equivalence Checking (PEC) [17]: A hardware designer wants to convince the customer that their opaque and partially specified implementation of a combinational circuit can still be completed into a full design that is functionally equivalent to the customer's specified target for future integration.

Conformant Planning (C-PLAN) [18, 19]: An AI service provider convinces its client to purchase a plan comprising a sequence of actions that transitions from initial states to a goal-satisfying state, regardless of the initial state or non-deterministic behavior of the environment.

Black Box Checking (BBC) [20]: A white-hat hacker proves the existence of software bugs that are independent of the behaviors of unknown structures and modules.

These applications ask the verifier to trust not only a negative certificate of infeasibility but also a constructive guarantee of capability. For a winning strategy, for example, an executable conformant plan for C-PLAN or a bug-exhibiting trace for BBC, will turn the satisfiability of a QBF into an operational certificate, whereas UNSAT proofs cannot certify how to act or what is the strategy to win. This work makes such capability attestable in zero knowledge, we prove the existence of strategies without disclosing them.

In this work, we propose efficient ZKPs for QBF evaluation, which make ZKPs for other PSPACE statements tractable and thereby address the aforementioned challenges. In Section 6, we further elaborate our protocols' capabilities on PEC, C-PLAN, and BBC through evaluations

on real-world benchmarks.

Key challenges. The theoretical feasibility of constructing ZKPs for PSPACE languages follows from the classical result that IP = PSPACE [21, 22], where the evaluation of a QBF is encoded as an interactive proof using the sumcheck protocol. A generic transformation then converts this interactive proof into a zero-knowledge protocol [23]. However, this construction requires evaluating high-degree multivariate polynomials over large fields, with degrees scaling linearly with the number of variables. As a result, the approach incurs substantial computational overhead and is not suitable for real-world QBF instances. Designing efficient and scalable ZKPs for PSPACE-complete problems remains an open and largely unexplored direction.

Not only the truth value of a QBF but also the *winning strategy* is of interest in practice. A winning strategy is a concrete example that describes how to assign existential (universal) quantified variables in response to universal (existential) quantified ones. In the grid game example, a winning strategy corresponds to a sequence of moves that ensures success regardless of the verifier's responses. More generally, the winning strategy of a QBF encodes a functional dependency of existential (universal) quantified variables on the universal (existential) quantified variables, which can be represented via Skolem functions (Skolemization) [24]. Existing ZKP protocols that only prove knowledge of a QBF's evaluation do not take into consideration the knowledge of such a concrete winning strategy. On the other hand, the knowledge of such a winning strategy is not only the object of verification in real-world applications, but also can improve the performance of the ZKP for QBF evaluation as the extended witness.

This work. We design and implement a novel, efficient ZKP for the evaluation of public QBFs. Our protocol can be used directly to efficiently prove knowledge of any statements in PSPACE once the statement has been reduced to prove the evaluation of QBF. We also introduce a ZKP protocol for proving knowledge of a winning strategy for a given QBF. In fact, we have demonstrated that with the aid of winning strategies, the resulting ZKP can be highly efficient for some instances, allowing us to reduce the proving time by 200X*.

Our first protocol leverages quantified resolution proofs (Q-Res) as an additional input from the prover, making ZKP for QBF evaluation and, therefore, PSPACE practical. Q-Res of a QBF consists of deriving clauses according to two rules: ∃-resolution (∃-RES) and ∀-reduction (∀-RED). The proof ends up with a derived empty clause, demonstrating that the formula is false. Although a Q-Res of a QBF can theoretically be exponentially large, for many QBFs encoding real-world problems, the resulting Q-Res proofs are of reasonable size. Modern QBF solvers can generate such proofs.

To further make ZK for PSPACE practical, we propose a ZKP that enables the prover to prove her knowledge of a winning strategy rather than merely the evaluation of the QBF. If the prover is willing to disclose the size of this winning strategy, we find that the ZKP for PSPACE can be efficiently reduced to ZKP for validity (when the original QBF is true) or unsatisfiability (when the original QBF is false), with only a small overhead in verifying the correct dependency in the winning strategies in ZKP. Furthermore, ZKP for unsatisfiability and validity has been studied, with highly optimized protocols readily available. By leveraging these well-established techniques, our approach strikes a balance between efficiency and practicality, making it suitable for large-scale applications.

We improve the implementation of our protocols by analyzing the Q-Res proof structure in the context of ZKP. We propose a hierarchical encoding scheme that groups clauses by their size,

*It is not necessary for the winning strategy to improve the proving time as the size of the winning strategy can be significantly larger than the size of Q-Res proof.

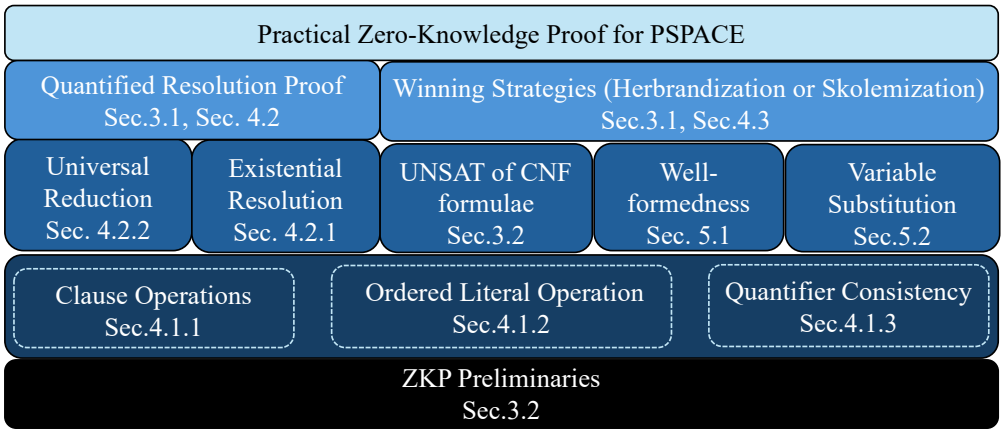

Figure 1: **Paper roadmap.** We propose two practical approaches for proving QBF evaluation in ZK: one based on Q-Res proofs and the other based on winning strategies. We implement and evaluate both approaches, with results presented in Section 6.

improving runtime performance by approximately half.

Our Contribution. We present the first practical ZKP for the evaluation of a QBF. Our work expands ZKPs for PSPACE-complete problems and their applicability to real-world scenarios. Our paper makes the following contribution:

- We introduce a quantifier-encoding scheme that enables the verification of both Q-Resolution and Q-Cube-Resolution proofs, and we design a protocol to check that a private CNF (sub)formula encodes a private winning strategy by verifying its derivation from a private And-Inverter Graph (AIG).
- We develop a novel and efficient ZKP protocol for proving the evaluation of QBF by synergizing the advances in QBF reasoning and ZKP. Our approach provides an efficient way to encode the QBF and Q-Res using polynomials and perform validity checking of Q-Res proofs.
- We enable the prover in ZKP to prove not only the evaluation of the QBF, but also the knowledge of the winning strategies. With the winning strategy, ZKP for QBF evaluation can be reduced to ZKP for UNSAT by revealing the size of the winning strategy.
- We implement our protocols and evaluate them on QBFEVAL, a well-established benchmark suite for QBF solvers, and with instances derived from real-world problems. For QBFEVAL'07, Out of 392 false QBFs for which we obtain Q-Res proofs or winning strategies, our protocols can verify QBF evaluations for about 82% of instances in 100 seconds (see Figure 2). Instances from PEC, C-Plan, and BBC are verified within 300, 1,200, and 200 seconds, respectively.
- We present a highly optimized implementation of our protocols. To enhance efficiency, we also introduce a batching scheme that groups clauses into buckets of similar width, thereby minimizing the padding overhead. This optimization reduces the runtime by approximately 50%.

Our implementation can be found at https://github.com/PP-FM/zkqbf-suite.

1.1 Related Work

Proof systems for QBF. Proof systems for complexity classes beyond NP remain an active area of research, addressing both foundational questions in proof complexity and the development of practical solvers. Early work by Jussila, Sinz, and Biere [25, 26] showed that extended resolution

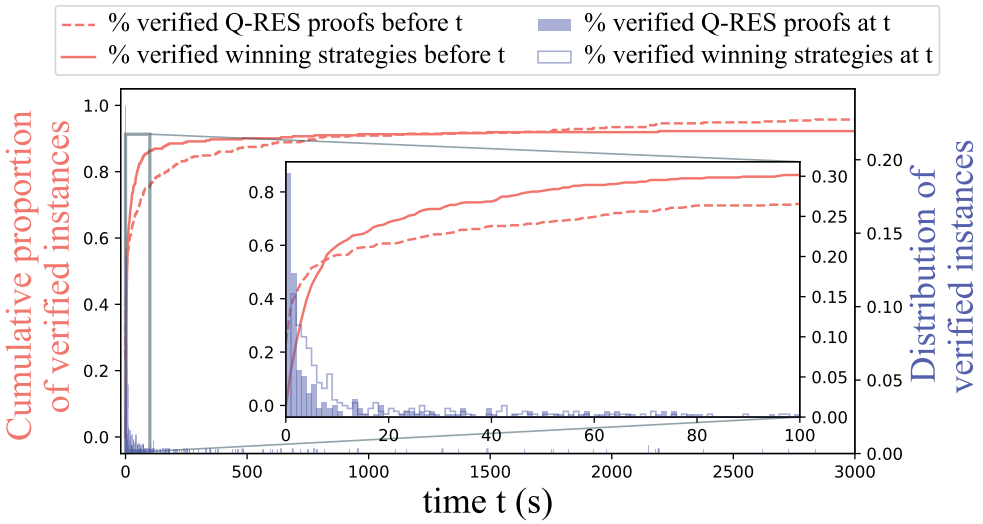

Figure 2: **Our protocols' evaluation against QBFEVAL.** We present the cumulative fraction of QBF instances successfully verified by our protocols via Q-RES proofs and winning strategies within a given time threshold (left Y-axis), as well as the fraction verified around each time point (right Y-axis). Our protocols can verify 72% of QBFs' evaluations via Q-Res proof and 82% of instances' winning strategies within 100 seconds, for instances where such proofs or strategies can be obtained. See Section 6 for details.

proofs can be systematically derived from Binary Decision Diagram (BDD) operations. This insight enabled proof-producing SAT and QBF solvers based on symbolic representations. The primary proof systems for quantified Boolean formulas (QBFs) include Q- and QU-resolution, long-distance Q-resolution, ∀Exp+Res, IR-calc (instantiation and resolution calculus), IRM-calc (IR with merging), and merge resolution [27, 28, 29, 30]. Among these, IR-calc and IRM-calc provide a unified framework that captures both CDCL-style and expansion-based reasoning, and they support efficient extraction of universal strategies [27]. The complexity of strategy extraction and its connection to proof size has also been studied extensively [30, 31]. Frege systems and their circuit-augmented variants [32] are known to simulate nearly all existing clausal and expansion-based QBF proof systems, offering a standard format for expressing complex proofs. Beyond classical logics, interactive proofs provide an additional dimension of expressiveness and efficiency. The IP = PSPACE theorem enables the verification of PSPACE-complete languages using interactive protocols with polynomial-time verifiers [33, 34].

We also acknowledge concurrent work by Kolesar et al. [35] that targets ZK Protocols for PSPACE-Complete problems through proving the equivalence of two regular expressions.

ZKPs beyond NP. A classical result established in the 1980s shows that any language in the class IP admits a zero-knowledge proof, via a generic transformation from interactive proofs to ZKP [23]. This foundational result ensures that all PSPACE-complete problems, including QBF, are in ZK. However, the transformation is not optimized for practical use and does not yield efficient or scalable ZKP protocols. Recent work has advanced practical ZKPs for specific logical properties such as unsatisfiability and validity [36,37,38]. ZKUNSAT [36] and ZKSMT [37] implement zero-knowledge protocols for certifying unsatisfiability of propositional and SMT formulas, respectively. However, both frameworks lack support for quantifiers and therefore cannot address problems in PSPACE. A different direction is taken by zkPi [38], which encodes formal proofs from interactive theorem provers, primarily Lean. zkPi accommodates a broad range of logical constructs, including algebraic data types, lambda calculus, and inductive reasoning. In contrast, our work targets scalable and

efficient zero-knowledge protocols tailored explicitly for large QBF instances, and does not rely on proof traces from external theorem provers. Recent developments in ZK extend to settings beyond NP using multi-prover interactive proofs (MIP*) and protocols for NEXP [39,40,41]. These systems rely on multiple provers and are thus orthogonal to the single-prover setting relevant to PSPACE. Moreover, the complexity-theoretic result QIP = PSPACE [42] establishes the equivalence between quantum interactive proofs and PSPACE, highlighting that PSPACE languages admit interactive proofs with quantum verifiers.

2 A motivating example.

Continuing with the grid game example, the prover can establish the truth of the original QBF by demonstrating that its negation is false. For clarity, we illustrate this using a simplified predicate, denoted as $\neg$`Reachable`. The goal of the prover is to falsify the following QBF:

Formula 1.

$$\forall x_0\ \exists y\ \forall x_1.\quad (x_0 \vee y \vee x_1) \wedge (x_0 \vee \neg y \vee \neg x_1) \\ \underbrace{\wedge(\neg x_0 \vee y \vee \neg x_1) \wedge (\neg x_0 \vee \neg y \vee x_1)}_{\neg \mathit{Reachable}} \tag{1}$$

$\forall$-Reduction ($\forall$-Red): Consider the clause $(x_0 \vee y \vee x_1)$ under the quantifier prefix $\forall x_0\ \exists y\ \forall x_1$ in the example. Since the existential variable y appears before x_1 in the quantifier prefix, it cannot depend on x_1. Therefore, we can safely remove x_1 from the first clause, yielding $C_0 = (x_0 \vee y)$. This rule of deduction is called $\forall$-reduction. Using the same rule, we can also have $C_1 = (\neg x_0 \vee y)$ from the third clause, and $C_2 = (\neg x_0 \vee \neg y)$ from the fourth clause.
$\exists$-Resolution ($\exists$-Res): Given the clauses C_1 and C_2, we can resolve on the existential variable y to obtain $C_3 = (\neg x_0)$. The derived clause C_3 is valid under the same quantifier prefix if the original QBF is valid and C_3 does not contain both a literal and its negation (C_3 is non-tautological). This rule of deduction is called the $\exists$-resolution.

We can apply $\forall-$Red again to $(\neg x_0)$ in the end, as x_0 is the only universal variable in the clause. We obtain the empty clause $\perp$, which falsifies the original QBF. Such a process leading to the empty clauses by applying $\forall$-Red and $\exists$-Res is called a Q-resolution (Q-Res) proof.

$$\dfrac{\dfrac{(\neg x_0 \vee y \vee \neg x_1)}{(\neg x_0 \vee y)}\ \forall\textbf{-Red on } x_1 \qquad \dfrac{(\neg x_0 \vee \neg y \vee x_1)}{(\neg x_0 \vee \neg y)}\ \forall\textbf{-Red on } x_1}{(\neg x_0)}\ \textbf{Res on } y \\ \dfrac{(\neg x_0)}{\perp}\ \forall\textbf{-Red on } x_0$$

Figure 3: **The Q-Res proof for falsifying Formula 1.** Clauses are deducted via $\forall$-Red and $\exists$-Res. $\forall$-reduction removes universal literals from a clause when permitted by the quantifier prefix. $\exists$-Res combines two clauses by eliminating the complementary literals. The deducted empty clause indicates that the evaluation of Formula 1 is false.

To enable practical ZKP for QBF evaluation via Q-Res, the following functionalities should be efficiently supported to arithmetize the verification of the Q-Res proof.

- Verification of the quantifier type (universal or existential quantified) of each variable, and consistency of quantifiers of variables across proof steps;
- Validation of the quantifier-induced order of literals to ensure the soundness of universal reduction;

- Efficient checking of both existential resolution ($\exists$-RES) and universal reduction ($\forall$-RED) steps.

To achieve this, we first maintain two sets $\mathcal{L}_\forall$ and $\mathcal{L}_\exists$ for existential and universal quantified literals, respectively. Each literal is represented by a binary string where the first bit encodes the sign (0 for negation) and the remaining bits denote the variable order in binary. For example, $\neg x_1$ is encoded as $\texttt{010}$ and x_1 as $\texttt{110}$ given the quantifier prefix $\forall x_0 \exists y \forall x_1$. These binary strings admit dual interpretations: 1) as field elements in $\mathbb{F}_{2^K}$ (for K-bit encodings, and 2) integer values through binary-to-decimal conversion.

Clauses are encoded via polynomials in $\mathbb{F}_{2^k}[X]$ whose roots correspond to literal encodings. The encodings of literals are interpreted as elements of $\mathbb{F}_{2^k}$. For instance, the clause $C = x_0 \vee \neg y \vee \neg x_1$ is encoded as:

$$\gamma(C)(X) = (X - \texttt{100}_\mathbb{F})(X - \texttt{001}_\mathbb{F})(X - \texttt{010}_\mathbb{F})$$

Such clause encoding allows efficient verification of resolution and reduction steps via arithmetic over $\mathbb{F}_{2^k}$.

Consider verifying the correctness of the clause C_r, which is claimed to be the result of applying a universal reduction ($\forall$-RED) step to the original clause $x_0 \vee y \vee \neg x_1$. Let $P_{C_r}(X)$ be the polynomial encoding C_r; We can first construct the extended witness containing a residual set $W_{\mathsf{res}} = \{\texttt{100}\}$, a removal set $W_{\mathsf{rem}} = \{\texttt{010}\}$, and an existential pivot encoding $w_\exists = \texttt{001}$. Verification proceeds by checking in ZK:

1. Membership of $w_\exists \in \mathcal{L}_\exists$ and inclusion of $W_{\mathsf{rem}} \subseteq \mathcal{L}_\forall$;
2. Equivalence that $(X - w_\exists) \prod_{e \in W_{\mathsf{res}}} (X - e) \equiv \gamma(C_r)(X)$;
3. For each $\ell \in W_{\mathsf{rem}}$, $\xi(\ell) > \xi(w_\exists)$.

Here, $\xi(\cdot)$ is an operator that maps a literal encoding to its order in the quantifier prefix, defined as the integer represented by the last two bits of the encoding. We also need to verify the dual interpretation of literal encodings, clause containment of literals, and quantifier consistency. Our approach is detailed in Section 4.1.

Proof via winning strategies. In fact, to show the QBF is unsatisfiable, we can further define the functions f_{x_1} and f_{x_0} for the universal-quantified variables x_1 and x_0 as $f_{x_0} = \mathsf{false} \quad f_{x_1}(y) = y$. We then substitute the x_1 and x_0 with this function and obtain:

$$\psi_H = (y \vee y) \wedge (\neg y \vee \neg y) \wedge (y \vee \neg y) \wedge (\neg y \vee y) \tag{2}$$

Simplifying the formula yields $y \wedge \neg y$, which is unsatisfiable. This unsatisfiability of Ψ_H directly implies that the original Formula 1 is false. The reason is that, using the universal strategy $f_{x_0} = \mathsf{false}$ and $f_{x_1}(y) = y$, the universal player can always force the formula to evaluate to false, regardless of the existential player's choice of y. Moreover, since x_1 appears after y in the quantifier prefix, f_{x_1} is allowed to depend on y while f_{x_0} can not. By such substitutions, we reduce the falsification of the QBF to checking the unsatisfiability of a purely propositional formula, which can be handled using existing ZKP for unsatisfiability proofs for private formulae [36]. Meanwhile, we also need to check the following constraints in ZK:

1. *Dependency correctness:* The strategy must respect the quantifier prefix. Here, f_{x_0} must be a constant (as x_0 is the first variable), and f_{x_1} can depend only on existential variables that appear before x_1 (in this case, y).

2. *Substitution correctness:* Ψ_H must be a result from substituting the universal variables x_0 and x_1 with their respective strategy functions in the ψ.

In Section 5, we describe how efficient verification of dependency correctness and substitution correctness can be achieved in ZK.

3 Preliminaries

3.1 Quantified Boolean Formulae

A *quantified Boolean formula (QBF)* is a propositional formula extended with quantifiers over Boolean variables. A QBF in *prenex conjunctive normal form (PCNF)* has the form:

$$\Psi = \underbrace{Q_1\mathcal{X}_1\ Q_2\mathcal{X}_2; \ldots\ Q_n\mathcal{X}_n.}_{\text{prefix}}\ \underbrace{\psi(x_0, \ldots, x_n)}_{matrix}$$

where each $Q_i \in \{\forall, \exists\}$ is a quantifier and $\mathcal{X}_i$ is a set of Boolean variables. ψ is a Boolean formula in conjunctive normal form (CNF) over variables drawn from $\mathcal{X} = \bigcup \mathcal{X}_i$. Throughout this work, we assume that all quantified Boolean formulae (QBFs) are given in PCNF. QBF truth evaluation is known to be PSPACE-complete.

Semantics. Let Φ be as above. The truth of Φ is defined inductively:

- If Φ has no quantifiers, then its value is determined by the truth of the propositional formula F.

- $\exists x.\Psi$ is true if there exists $b \in \{0, 1\}$ such that $\Psi[x \mapsto b]$ is true.

- $\forall x.\Psi$ is true if for all $b \in \{0, 1\}$, $\Psi[x \mapsto b]$ is true.

We use $\mathcal{X}_\forall$ and $\mathcal{L}_\forall$ to denote the sets of universally quantified variables and their corresponding literals. Literals are variables or their negations. That is, $\mathcal{X}_\forall = x_i \mid x_i \in \mathcal{X}_i, , Q_i = \forall$ and $\mathcal{L}_\forall = x_i, \neg x_i \mid x_i \in \mathcal{X}_\forall$. Similarly, we use $\mathcal{X}_\exists$ and $\mathcal{L}_\exists$ to denote the sets of existentially quantified variables and literals.

The quantifier prefix $Q_1\mathcal{X}_1 \cdots Q_n\mathcal{X}_n$ induces a partial order over the literal set $\mathcal{L}$ based on the ordering of quantifier blocks. Specifically, for $x_i \in \mathcal{X}_i$ and $x_j \in \mathcal{X}_j$, we write $x_i \prec x_j$ (and similarly $\neg x_i \prec x_j$, $x_i \prec \neg x_j$, etc.) if the block $\mathcal{X}_i$ appears before $\mathcal{X}_j$ in the prefix.

A clause, defined as a disjunction of literals, is naturally represented as a *set* of literals. In this representation, the notation $\ell \in C$ denotes that the literal ℓ occurs as a disjunct in clause C. The union $C_1 \cup C_2$ corresponds to the clause containing all literals that appear in either C_1 or C_2.

The Q-Resolution proof system (Q-Res). A *Q-resolution proof* operates on the clauses in the CNF matrix of a QBF and derives the *empty clause* using the following rules:

Universal Reduction ($\forall$-Red*).* Let $\Psi = Q.\psi$ be a QBF in PCNF and let C be a clause in ψ. Let y be the innermost existential literal in C. Any universal literal $x_i \in C$ with $y \prec x_i$ can be removed from C without affecting the truth value of Ψ. We say that $C \vdash_{Q,\forall-RED} C_r$ if C_r is obtained by applying $\forall$-Red on C according to Q.

Existential-Resolution ($\exists$-Res*).* Let C_1, C_2 be clauses in ψ. If there exists an existential variable $y \in \mathcal{X}_\exists$ such that $y \in C_1$ and $\neg y \in C_2$, the Q-resolvent of C_a and C_b on pivot y is derived as follows by computing the resolvent $C = (C_a \cup C_b) \setminus \{y, \neg y\}$. If C is a tautology, discard it; otherwise, C is the Q-resolvent. We say that $C_a, C_b, y \vdash_{Q,\mathsf{xres}} C_r$ if C_r is obtained by applying $\exists$-Res on C_a, C_b pivoting on y according to Q.

A Q-resolution proof is a finite sequence of clauses $C_1, C_2, \ldots, C_m$ in which each clause C_i is either an initial clause from the matrix F or is derived from earlier clauses C_a and C_b by first

applying universal reduction (∀-Red) to each and then performing existential resolution (∃-Res) on the results. If the final clause in the sequence is the empty clause $\perp$ (i.e., $C_m = \perp$), the sequence constitutes a *Q-resolution refutation* of the QBF Φ.

Theorem 1 ([43]). *A closed QBF in PCNF is unsatisfiable if and only if there exists a sequence of Q-resolution steps leading to the empty clause.*

To prove a QBF to be true, one can employ a dual proof system. This proof system can be implemented in zero-knowledge using techniques almost identical to those used for Q-Res. We provide details in Appendix Appendix A.

Winning strategies. A *winning strategy* for the universal player, i.e., a refutation strategy, is a set of Boolean functions

$$\mathcal{H} = \{g_x \mid x \in \mathcal{X}_\forall\}$$

such that for some $x \in \mathcal{X}_\forall$, the function $g_x : \text{Pred}_\exists(x) \to \{0,1\}$ maps the valuations of all variables that precede x in the prefix to a Boolean value. Here, $\text{Pred}_\exists(x) = \{x_j \mid x_j \prec x\}$. The strategy $\mathcal{H}$ is *winning* for the universal player if substituting each universal variable $x \in \mathcal{L}_\forall$ with its corresponding *Herbrand function* g_x yields a propositional formula Ψ_H that is unsatisfiable. We refer to Ψ_H as the *Herbrandization* of the QBF Ψ.

Dually, a winning strategy for the existential player is a set of Boolean functions $\mathcal{S} = \{f_x \mid x \in \mathcal{X}_\exists\}$ such that for every $x \in \mathcal{X}_\exists$, the function $f_x : \text{Pred}_\forall(x) \to \{0,1\}$ maps the valuations of all *universally* quantified variables that precede x in the quantifier prefix to a Boolean value. The strategy $\mathcal{S}$ is *winning* if substituting each existential variable $x \in \mathcal{X}_\exists$ with its corresponding *Skolem function* f_x in ψ yields a propositional formula Ψ_S over only the universal variables $\mathcal{X}_\forall$ that is a tautology. We refer to Ψ_S as the *Skolemization* of the QBF Ψ.

Theorem 2 ([44, 45]). *A QBF Ψ is* true *(or satisfiable, when there are free variables) if and only if there exists an existential winning strategy (i.e., Skolem functions). It is* false *if and only if there exists a universal winning strategy (i.e., Herbrand functions).*

Example. We define the Herbrand functions f_{x_1} and f_{x_0} for x_1 and x_0 as: $f_{x_0} = \mathsf{false} \quad f_{x_1}(y) = y$. We substitute the universal variables with their corresponding Herbrand functions and obtain an unsatisfiable formula. Therefore, the strategy above is a valid Herbrand strategy, and the QBF is *false*.

3.2 Efficient Zero-Knowledge Proof

ZKP [46,47] allows a prover to convince a verifier that it possesses an input w such that $P(w) = 1$ for some public predicate P, while revealing no additional information about w. There have been many lines of work in designing practically efficient ZK protocols under different settings and assumptions (eg. [48, 49, 50, 51]).

The goal of this work is to build practical ZKPs for QBF evaluations and winning strategies based on existing protocols rather than proposing the construction of a general-purpose ZKP. To this end, we extract the ZKP functionalities required for our protocol in Figure 4. In particular, we use a special type of ZK protocol commonly referred to as "commit-and-prove" ZK [52], which allows a witness to be committed and later proven over multiple predicates while ensuring consistency of the committed values. Figure 4 lists the ZK functionality we need for building our protocols.

Set argument. Our protocol reduces statements over literals and clauses to statements about set membership and subset relations. These set-based primitives serve as the foundation for verifying structural properties of formulae. In practice, a variety of ZKPs exist that efficiently implement such set operations, such as [53]. We list the functionality of our interests in Figure 5.

Functionality $\mathcal{F}_{\mathsf{ZK}}$

Witness: On receiving $(\mathsf{Witness}, x)$ from the prover, where $x \in \mathbb{F}$, store x and send $[x]$ to each party.

Instance: On receiving $(\mathsf{Instance}, x)$ from both parties, where $x \in \mathbb{F}$, store x and send $[x]$ to each party. If the inputs sent by the two parties do not match, the functionality aborts.

Circuit relation: On receiving $(\mathsf{Relation}, C, [x_1], \ldots, [x_{n-1}])$ from both parties, where $x_i \in \mathbb{F}$ and $C \in \mathbb{F}^n \to \mathbb{F}^m$, compute $y_1, \ldots, y_m := C(x_1, \ldots, x_{n-1})$ and send $\{[y_1], \ldots, [y_m]\}$ to both parties.

Productions-of-multi-variate-polynomial equality: On receiving $(\mathsf{PoPEqCheck}, X \{[P_i(X)]\}_{i\in[n]}, \{[Q_i(X)]\}_{i\in[m]})$ from both parties, where $[P_i(X)]$ and $[Q_i(X)]$ are multi-variate polynomials over X with their coefficient committed: if $\Pi_i P_i(X) \neq \Pi_i Q_i(X)$, the functionality aborts.

Conversion: On receiving $(\mathsf{Conv}, [x], i, \xi)$ from $\mathcal{P}$ and $(\mathsf{Conv}, [x], \xi)$ from $\mathcal{V}$ where $i \in \mathbb{N}$, store i and send $[i]$ to both parties. If $\xi : \mathbb{F} \to \mathbb{N}$ from $\mathcal{P}$ and $\mathcal{V}$ are different or $\xi(x) \neq i$, the functionality aborts.

Comparison: On receiving $(\diamond, [i], [j], \lambda)$ from both parties, where $\diamond \in \{>, \geq, <, \leq\}$. If i or $j \notin N$ or $i \diamond j$ does not hold when $\diamond$ is interpreted as the standard integer comparison, the functionality aborts.

Figure 4: Functionality for zero-knowledge proofs of circuit satisfiability and integer comparison.

Functionality $\mathcal{F}_{\mathsf{ZKSet}}$

Set initialization: On receiving $(\mathsf{Init}, N, [s_1], \ldots, [s_N])$ from $\mathcal{P}$ and $\mathcal{V}$, where $s_i \in \mathbb{F}$. Store the $S = \{s_i\}$ and set $f := \mathsf{honest}$ and send $[S]$ to each party.

Set subset: On receiving $(\mathsf{Subset}, S', [S])$ from $\mathcal{P}$, and $(\mathsf{Subset}, [\mathsf{S}])$ from $\mathcal{V}$, If S' is not a subset of S, set $f := \mathsf{cheating}$ and send $[S']$ to each party.

Set membership: On receiving $(\mathsf{Mem}, \{[s_1], \cdots, [s_w], [S]\})$ from $\mathcal{P}$ and $\mathcal{V}$, where $s_i \in \mathbb{F}$, set $f := \mathsf{cheating}$ if $s_i \notin S$ for some i.

Set check: Upon receiving (check) from $\mathcal{V}$ do: If $\mathcal{P}$ sends $(\mathsf{cheating})$ then send $\mathsf{cheating}$ to $\mathcal{V}$. If $\mathcal{P}$ sends $(\mathsf{continue})$ then send f to $\mathcal{V}$.

Figure 5: Functionality for set operations in ZK.

Append-only array. We leverage $\mathcal{F}_{\mathsf{FlexZKArray}}$ to store all clauses from both the input formula and the derived proof. $\mathcal{F}_{\mathsf{ZKFlexZKArray}}$ is a specialized array structure that avoids the overhead of generic RAM access in ZKP by supporting only two operations: *append* and *read* [36]. The protocol assumes that the prover precomputes and appends all clause entries in advance, allowing the verifier to verify only the read operations in ZK. In addition, the access functionality is intentionally weakened: the verifier cannot track repeated appends or enforce strict ordering of reads. The formal definition is provided in Appendix Figure 16.

ZKUNSAT [36]. ZKUNSAT proves the unsatisfiability of a private proposition Boolean formula in CNF by leveraging its resolution proof. A resolution proof is a sequence of derived clauses, ending in the empty clause $\perp$, where each clause is either part of the initial formula or derived using the resolution rule. The resolution rule allows deriving a new clause from two clauses that contain a unique pair of complementary literals. Specifically, we say that $C_a, C_b, \ell_p \vdash_{\mathsf{res}} C_r$

$$C_a = \ell_p \vee \ell_1 \vee \cdots \vee \ell_m \quad C_b = \neg\ell_p \vee \ell'_1 \vee \cdots \vee \ell'_k$$
$$C_r = \ell_1 \vee \cdots \vee \ell_m \vee \ell'_1 \vee \cdots \vee \ell'_k,$$

and C_r is not a tautological clause (i.e., it does not contain both a literal and its negation).

Notice that the derivation $C_a, C_b, \ell_p \vdash_{\mathsf{xres}} C_r$ is the same as the $\exists$-RESstep in Q-RES, with the exception that it does not enforce the pivot variable to be existentially quantified. We leverage the efficient technique from [36] to verify resolution steps in Q-RES. Below, we outline how [36] enables ZKP of correct resolution steps and the conditions under which soundness is guaranteed.

In ZKUNSAT, each literal ℓ is encoded as a field element $\epsilon(\ell) \in \mathbb{F}$ and committed accordingly. A clause C is represented as a polynomial $\mathcal{P}_C$ whose roots are the encoded literals in C. To verify the correctness of resolution steps, ZKUNSAT checks a set of algebraic relations over the polynomials $\mathcal{P}_C$, $\mathcal{P}'_C$, and $\mathcal{P}_{C_r}$ corresponding to the two parent clauses and their resolvent. These checks reduce to verifying polynomial identities, which can be implemented using $\mathcal{F}_{\mathsf{ZK}}$.`PoPEqCheck`. To ensure the soundness, the encoding function $\epsilon(\cdot)$ should be injective and designed such that for every literal ℓ, and $\epsilon(\ell) + \epsilon(\neg\ell) = \mathtt{cst}_{\mathbb{F}}$[†] holds.

for some fixed constant $\mathtt{cst}_{\mathbb{F}} \in \mathbb{F}$. Under this encoding, satisfaction of the polynomial relations guarantees the soundness of each resolution step in the proof. ZKUNSAT also leverages the $\mathcal{F}_{\mathsf{FlexZKArray}}$ functionality to store all clauses from both the input formula and the derived proof.

4 ZKP for Q-Res Proof Validation

This section presents our protocol for verifying the correctness of a QBF's Q-resolution proof. We begin by describing the data structure used to encode QBFs and Q-RES proofs arithmetically. We then detail how this structure enables the efficient verification of $\forall$-RED and $\exists$-RES steps within a zero-knowledge proof (ZKP) framework.

4.1 Encoding QBF

4.1.1 Ordered literal encoding

The variables in QBF are ordered. Given a QBF with $\Psi = Q_1\mathcal{X}_1 \ldots Q_k\mathcal{X}_k.\ \psi(x_1, \ldots, x_n)$ over variables $\mathcal{X} = \cup\mathcal{X}_i$. We assume that the quantifier prefix induces a total order over $\mathcal{X}$.[‡] We define the encoding of a literal ℓ as the concatenation of three *binary strings*:

$$\epsilon(\ell) = \mathsf{order}(\ell)_{\mathbf{B}} \| \ \mathsf{sign}(\ell) \tag{3}$$

where $\mathsf{sign}(\ell) \in \{0, 1\}$ is a 1-bit flag indicating the sign of the literal: $\mathsf{sign}(\ell) = 0$ if $\ell = \neg x$, and $\mathsf{sign}(\ell) = 1$ if $\ell = x$ for some $x \in \mathcal{X}$. The function $\mathsf{order}(\ell) = \mathsf{order}_x$ if $\ell = x$ or $\neg x$, and order_x is the order of x in $\mathcal{X}$. We use $\mathsf{order}(\ell)_{\mathbf{B}}$ to denote the binary presentation of the order as an integer.

We adopt this encoding to enable efficient extraction of the information required for verifying both $\forall$-RED and $\exists$-RES steps. The length of encoding can be bounded by $1 + \log \mathcal{X}$. In addition, the encoding is injective when the string length is fixed, enabling each literal to be uniquely mapped to an element in $\mathbb{F}$.

Each encoding $\epsilon(\ell) \in \{0, 1\}^k$ is interpreted both as a unique element in a finite field and as a natural number via the interpretation functions $\mathsf{Itp}\mathbb{F}$ and $\mathsf{Itp}\mathbb{N}$, respectively:

$$\mathsf{Itp}_{\mathbb{F}} : \{0, 1\}^k \to \mathbb{F} \text{ s.t. } \mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell)) + \mathsf{Itp}_{\mathbb{F}}(\epsilon(\neg\ell)) = 1_{\mathbb{F}} \tag{4}$$

$$\mathsf{Itp}_{\mathbb{N}} : \{0, 1\}^k \to \mathbb{N} \text{ s.t. } \mathsf{Itp}_{\mathbb{N}}(\epsilon(\ell)) = \mathsf{order}(\ell) \tag{5}$$

The finite field $\mathbb{F}$ is chosen such that $|\mathbb{F}| \geq |\mathcal{L}|$, and the function $\mathsf{Itp}_{\mathbb{F}}$ should also be injective.

The composed function $\mathsf{Itp}_{\mathbb{F}} \circ \epsilon$ provides an injective encoding that satisfies the algebraic constraints needed to verify resolution steps in zero knowledge using the approach introduced in [36].

[†]This condition is necessary to enable efficient checking of complementary literals.

[‡]We assign each quantifier block $\mathcal{X}_i$ a total order over its variables and extend this to a total order over the entire set $\mathcal{X}$, such that the global ordering is consistent with the quantifier prefix.

Functionality $\mathcal{F}_{\mathsf{OrdLiteral}}$

Parameter: On receiving $(\mathsf{Init}, \mathsf{order}, k)$ from $\mathcal{P}$ and $\mathcal{V}$ where $\mathsf{order} : \mathcal{L} \to \mathbb{N}_{2^k}$, abort if they are not identical; otherwise store (order, k).

Input: On receiving (Input, ℓ) from $\mathcal{P}$ and (Input) from verifier where $\ell \in \mathcal{L}$, if ℓ from two parties differ, the functionality aborts; Otherwise store ℓ and send $[\ell]$ to both parties.

Order: On receiving $(\mathsf{Order}, ([\ell], i))$ from $\mathcal{P}$ and $(\mathsf{Order}, [\ell])$ from verifier. If $\mathsf{order}(\ell) \neq i$, the functionality aborts. Otherwise, the functionality sends $[i]$ to both parties.

Figure 6: Functionality for ZK operations on ordered literals.

Protocol $\Pi_{\mathsf{OrdLiteral}}$

Parameter: Given the function order, both $\mathcal{P}$ and $\mathcal{V}$ compute and agree on the functions $\mathsf{Itp}_{\mathbb{F}}$, and $\mathsf{Itp}_{\mathbb{N}}$ that satisfy Equation 4 and 5.

Input: $\mathcal{P}$ computes the encoding $\epsilon(\ell)$ according to Equation 3. Then $\mathcal{P}$ and $\mathcal{V}$ authenticate $\mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell))$ using $\mathcal{F}_{\mathsf{ZK}}$ and obtain $[\mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell))]$. The two parties then output $[\ell] = [\mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell))]$.

Order: Set $\xi = \mathsf{Itp}_{\mathbb{N}} \cdot \mathsf{Itp}_{\mathbb{F}}^{-1}$. $\mathcal{P}$ sends $(\mathsf{NInterpret}, [\ell], i, \xi)$ to $\mathcal{F}_{\mathsf{ZK}}$, while $\mathcal{V}$ sends $(\mathsf{NInterpret}, [\ell], \xi)$. They set the output as the returned value $[i]$.

Figure 7: Protocols for ZK operations on ordered literals.

Meanwhile, $\mathsf{Itp}_{\mathbb{N}} \circ \epsilon$ allows for recovering the relative quantifier order of literals, which is necessary to verify the correctness of universal reduction ($\forall$-Red).

The remaining task is to verify the correctness of conversions between elements in the finite field $\mathbb{F}$ and their corresponding bounded integer representations in $\mathbb{N}$ in ZK. We can leverage the existence of interpreters of any given $\mathbb{F}$, as we showed in Figure 7. In our implementation, we achieve this by adopting the approach introduced in [54, 55, 56], which provides an efficient approach for checking consistency between $\mathbb{F}$ and bounded integers in $\mathbb{N}$ in ZK when $\mathbb{F}$ is instantiated as $\mathbb{F}_{2^k}$.

ZK operations on ordered literals. In Figure 6, we specify the zero-knowledge operations over literals under a total order induced by the quantifier prefix of the input QBF, formalized in the functionality $\mathcal{F}_{\mathsf{OrdLiteral}}$.

During initialization, both parties provide an ordering function $\mathsf{order} : \mathcal{L} \to \mathbb{N}_{2^k}$ and bit-length parameter k. The functionality ensures that the parties agree on an ordering.

To commit a literal, prover inputs its encoded value, and both prover and verifier receive a committed version $[\ell]$ via $\mathcal{F}_{\mathsf{ZK}}$. To retrieve the quantifier-induced order of a literal, prover and verifier invoke the `Order` interface, which checks that the provided rank i matches $\mathsf{order}(\ell)$. The functionality then returns $[i]$ as a committed integer. These operations ensure that ordering checks over literals, necessary for verifying quantifier-respecting dependencies in Herbrand and Skolem strategies, can be performed in ZK.

4.1.2 Clause encoding

We adopt clause encoding from [36] for efficient verification of $\exists$-Res and $\forall$-Red in ZK, which we elaborate in 3. The clause is encoded as a polynomial over $\mathbb{F}$ rooted at the encodings of literals in the clauses, when the encodings are interpreted as elements in $\mathbb{F}$. Formally, for $C = \ell_1 \vee \cdots \vee \ell_d$

$$\gamma(C)(X) = (X - \mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell_0))) \cdots (X - \mathsf{Itp}_{\mathbb{F}}(\epsilon(\ell_d))).$$

ZK operations on polynomial-encoded clauses. We define a functionality $\mathcal{F}_{\mathsf{Clause}}$ that supports

Functionality $\mathcal{F}_{\mathsf{Clause}}$

Input: On receiving $(\mathsf{Input}, \ell_0, \cdots, \ell_{k-1}, w)$ from $\mathcal{P}$ and (Input, w) from verifier where $\ell_i \in \mathcal{L}$, the functionality check that $k \leq w$ and abort if it does not hold. Otherwise store $C = \ell_0 \vee \cdots \vee \ell_{k-1}$, and send $[C]$ to each party.

Literal retrieval: On receiving $(\mathsf{Retrieval}, \{\ell_1, \cdots, \ell_k\}, [C])$ from $\mathcal{P}$ and $(\mathsf{Retrieval}, [C])$ from $\mathcal{V}$, check if $C = \{\ell_1 \vee \cdots \vee \ell_w\}$; if not the functionality aborts. Otherwise, send $[\ell_1], \cdots, [\ell_k]$ to each party.

Equal: Upon receiving $(\mathsf{Equal}, [C], \{[C_i]\})$ from both parties, the functionality first checks whether any two clauses C_i and C_j (for $i \neq j$) have overlapping literals. If such overlap exists, the functionality aborts. Otherwise, it verifies whether $C = \bigvee_i C_i$; if this condition is not satisfied, the functionality aborts.

Res: On receiving $(\mathsf{res}, [C_0], [C_1], [\ell_p], [C_r])$ from both parties, check if $\{C_0, C_1\}, \ell_p \vdash_{\mathtt{res}} C_r$; if not the functionality aborts.

IsFalse: On receiving $(\mathsf{IsFalse}, [C])$ from both parties, check if $C = \bot$; if not, the functionality aborts.

Figure 8: Functionality for ZK operations on clauses.

authenticated operations over clauses in ZK, described in Figure 8. The protocols for implementing $\mathcal{F}_{\mathsf{Clause}}$ are explained in Appendix Figure 9. The prover initializes a clause by inputting a list of literal encodings, whose size is bounded by a public parameter w, and both parties receive a shared commitment $[C]$. Given this commitment, the parties can retrieve literals in the clauses using `Retrieval`, which ensures clause consistency by checking the inclusion of declared literals.

The functionality also provides a `Equal` operation that verifies in ZK whether a clause is logically equivalent to the disjunction of a set of committed subclauses, requiring no duplicate literal inclusion across clauses.

For resolution steps, $\mathcal{F}_{\mathsf{Clause}}$ offers a `Res` operation that checks whether a resolvent $[C_r]$ is validly derived from two input clauses under the non-tautological resolution rule. Finally, the `IsFalse` interface allows checking whether a clause is the empty clause $\bot$, signaling refutation in QBF proofs. All operations are designed to be sound and privacy-preserving within the zero-knowledge framework.

4.1.3 Quantifier encoding

We leverage two *public* sets $\mathcal{L}_\forall = \{x, \neg x | x \in \mathcal{X}_\forall\}$ and $\mathcal{L}_\exists = \{x, \neg x | x \in \mathcal{X}_\exists\}$ for keeping quantifier information with the order of the variables encoded by the literal encoding. Notice these two sets are public information as the QBF is public. The verifier can directly check if these two sets are well-formed: for any $\ell \in \mathcal{L}_\forall$ (or $\mathcal{L}_\exists$), $\neg\ell$ is also in $\mathcal{L}_\forall$ ($\mathcal{L}_\exists$). We list the functionality of $\mathcal{F}_{\mathsf{Quantifier}}$ in Figure 10. This functionality can be directly realized using $\mathcal{F}_{\mathsf{ZKSet}}$.

4.2 ZKP of QBF Evaluation via Q-Res Proofs

A Q-resolution (Q-Res) proof consists of a sequence of inference steps, each applying the Q-Res rule. Every Q-Res step takes two supporting clauses as input and performs an existential resolution ($\exists$-Res) followed by a universal reduction ($\forall$-Red) on the resulting resolvent. To ensure correctness, the verifier must check that both supporting clauses are either part of the input QBF's matrix or have been derived in earlier Q-Res steps. The latter condition can be handled using append-only data structures in ZK, following the approach in [36]. In this section, we first focus on how to verify the correctness of $\forall$-Red(Section 4.2.2) and $\exists$-Res(Section 4.2.1) steps in zero knowledge using the literal encoding described in Section 4.1.

Protocol Π_{Clause}

Parameters: A set $\mathcal{L}$ of all possible literals and a finite field $\mathbb{F}$. An integer w and a set of clauses $\mathbf{C}_w$ that contains all clauses no more than w literals of $\mathcal{L}$. $\epsilon : \mathcal{L} \to \{0,1\}^k$ and $\mathsf{ltp}_{\mathbb{F}}$ are injective.

Input [36]:

1. $\mathcal{P}$ holds a clauses $C = \ell_0 \vee \cdots \vee \ell_{k-1} \in \mathbf{C}_w$, defines $\gamma(C)(X) = (X - \mathsf{ltp}_{\mathbb{F}}(\epsilon(\ell_0))) \cdots (X - \mathsf{ltp}_{\mathbb{F}}(\epsilon(\ell_d)))$ and locally computes $c_0, \ldots, c_w$ such that $\gamma(C)(X) = \sum_{i\in[0,w]} c_i X^i$.
2. For each $i \in [0, w]$, two parties use $\mathcal{F}_{\mathsf{ZK}}$ to get $[c_i]$. Two parties output $[\gamma(C)] = \{[c_i]\}_{i\in[0,w]}$

Equal: Both parties send $(\mathsf{PoPEqCheck}, X, [\gamma(C)(X)], \{[\gamma(C_i)(X)]\})$ to $\mathcal{F}_{\mathsf{ZK}}$.

Literal retrieval:

1. $\mathcal{P}$ locally computes the encodings $\epsilon(\ell_i)$ and authenticate $\rho_i(X) = X - \epsilon(\ell_i)$ using $\mathcal{F}_{\mathsf{ZK}}$. AS a result, two parties get $[\rho_i(X)]$.
2. Both parties send $(\mathsf{PoPEqCheck}, X, \{[\rho_i(X)]\}, \{\gamma(C)\}(X))$ to $\mathcal{F}_{\mathsf{ZK}}$.

Res [36]: Details are provided in Appendix Figure 17.

IsFalse [36]: Both send $(\mathsf{PoPEqCheck}, X, [\gamma(C)(X)], [1])$.

Figure 9: Our protocol to instantiate $\mathcal{F}_{\mathsf{Clause}}$.

Functionality $\mathcal{F}_{\mathsf{Quantifier}}$

Initialization: Upon receiving $(\mathsf{init}, \mathcal{L}_\exists, \mathcal{L}_\forall)$ from both $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{L}_\exists$ and $\mathcal{L}_\forall$ are sets of literals, the functionality stores both lists. Abort if the inputs are not consistent across the two parties. Otherwise send $[\mathcal{L}_\forall]$ and $[\mathcal{L}_\exists]$ to each party. Set $f := \mathsf{honest}$ and ignore subsequent initialization calls.

Check: On receiving $(\mathsf{check}, [\ell], \square)$ from both $\mathcal{P}$ and $\mathcal{V}$, where $\square \in \{\forall, \exists\}$. Set $f := \mathsf{cheating}$ if $\square$ from parties are not consistent, or $\ell \notin \mathcal{L}_\square$.

Figure 10: Functionality for ZK operations on quantifier.

4.2.1 Existential resolution

Given two committed clauses C_a and C_b, $\{C_a, C_b\}, \ell_p \vdash_{\mathsf{Q,xres}} C_r$ if and only if the following conditions hold:

$$\ell_p \in \mathcal{L}_\exists \quad (6) \qquad \text{and} \qquad C_a, C_b, \ell_p \vdash_{\mathsf{res}} C_r \quad (7)$$

Here, ℓ_p denotes the pivot literal, and $\vdash_{\mathsf{res}}$ denotes the non-tautological resolution rule applied over complementary literals $\ell_p \in C_a$ and $\neg\ell_p \in C_b$. Constraint Equation (6) ensures that the pivot literal is existentially quantified, as required by the semantics of Q-RES. The verification of $\exists$-RES steps follows the same approach as in [36], with the additional requirement of checking that the quantifier type of the pivot literal is existential. This ensures that resolution is applied only over variables whose assignments are controlled by the existential quantifier, preserving the soundness of the proof under the QBF semantics.

4.2.2 Universal reduction

Given two committed clauses C and C_r, C_r is a valid universal reduction of C if and only if the following hold:

$$\exists w_\ell, \quad W_{\mathsf{res}} = \ell_1 \vee \cdots \vee \ell_k, \quad W_{\mathsf{rem}} = \ell'_1 \vee \cdots \vee \ell'_d \quad \text{s.t.}$$
$$\mathsf{order}(\ell_i) < \mathsf{order}(w_\ell), \quad \forall i \in [k] \tag{8}$$
$$\mathsf{order}(\ell'_i) > \mathsf{order}(w_\ell), \quad \forall i \in [d] \tag{9}$$
$$\ell'_i \in \mathcal{L}_\forall, \quad w_\ell \in \mathcal{L}_\exists \tag{10}$$
$$C = W_{\mathsf{res}} \cup W_{\mathsf{rem}} \cup \{w_\ell\}, \quad C_r = W_{\mathsf{res}} \cup \{w_\ell\} \tag{11}$$

Here, W_{rem} denotes the set of universally quantified literals that are removed during the $\forall$-Red step, while W_{res} contains the remaining literals preserved in the reduced clause. The literal w_ℓ is the existentially quantified literal in C with the highest order according to the quantifier prefix. Prover can prepare $w_\ell, \{w_\ell\}, W_{\mathsf{res}}$ and W_{rem} and commit them via $\mathcal{F}_{\mathsf{ZKLiteral}}$ and $\mathcal{F}_{\mathsf{ZKClause}}$.

In a universal reduction step, a set of universally quantified literals is removed from clause C to obtain a smaller clause C_r. This operation is sound only if the removed literals occur at the innermost positions in the quantifier prefix. Constraints 9 and 10 together ensure this condition holds: the removed literals must be universally quantified and must appear after all other literals in C with respect to the quantifier order. The soundness of Q-resolution (Q-Res) requires that all derived clauses be universally reduced. This means that any universal literals appearing after the innermost existential literal in a clause must be eliminated. Constraints 8 and 10 enforce this condition by ensuring that only the subset of universal literals with higher quantifier levels than the last existential literal is removed. Finally, constraint 11 guarantees that the decomposition of clause C into the retained literals (W_{res}), the removed literals (W_{rem}), and the retained existential literal (w_ℓ) is correct, and that the resulting clause C_r is computed accordingly. Together, these constraints ensure the correctness and soundness of each universal reduction step in the Q-Res proof.

Theorem 3. *Let Ψ be a QBF in PCNF, where all clauses are already universally reduced. Let its* Q-Res *proof consist of a sequence of derivation steps, each of the form $(C_a, C_b) \vdash C_r$, where C_r is derived from premises C_a and C_b. The proof is sound if the following conditions hold for every step:*

1. *The clauses C_a and C_b are either initial clauses from the matrix of Ψ or were derived in previous steps of the proof.*
2. *There exists a pivot literal ℓ_p and an intermediate clause $\bar{C}_r$ such that:*
   - *C_a, C_b, ℓ_p, and $\bar{C}_r$ satisfy the resolution constraints in Equations (6) and (7), and*
   - *$\bar{C}_r$ and C_r satisfy the universal reduction and structural consistency conditions in Equations (8) to (11).*

Under these conditions, the final clause $C_r = \bot$ constitutes a valid Q-Res *proof for refuting Ψ.*

This theorem follows directly from Theorem 1 and definitions of $\forall$-Redand $\exists$-Res.

4.3 Q-Res Validation in ZK

We present our protocol to verify the correctness of a Q-Res proof in ZK, detailed in Figure 11. First, both parties obtain clause commitments $[C_i]$ for all clauses in the input formula using $\mathcal{F}_{\mathsf{Clause}}$. We adopt the weakened random array access approach proposed in [36] to store both the clauses

from the QBF matrix and the derived clauses obtained through consecutive applications of $\exists$-Res and $\forall$-Red steps. In particular, prover extracts each step of the Q-Res proof locally, including all derived clauses, and both parties authenticate all clauses (input and derived) and store them in an array initialized using $\mathcal{F}_{\mathsf{FlexZKArray}}$.

Let the QBF be $Q_1x_1, Q_2x_2, \cdots, Q_kx_k.\psi$, where ψ is a CNF formula expressed as $C_1 \wedge \cdots \wedge C_f$. We assume that each clause C_i has already been universally reduced. The Q-Res proof is given as a sequence of R resolution steps, each identified by a pair of clause indices (k_i, l_i) indicating the premises used in the i-th step. The protocol requires public knowledge of the proof length R, the resolution proof width w (the maximum number of literals contained in any clause in the Q-Res proof), and the deduction degree d (the maximum number of literals removed in any $\forall$-Red step).

The protocol begins by having both parties initialize the necessary encodings. Using the quantifier prefix, they compute a total order over variables and generate structured representations of the initial clauses $C_1, \ldots, C_f$ via functionalities $\mathcal{F}_{\mathsf{OrdLiteral}}$ and $\mathcal{F}_{\mathsf{Clause}}$. Additionally, quantifier sets $[\mathcal{L}_\exists]$ and $[\mathcal{L}_\forall]$ are obtained using $\mathcal{F}_{\mathsf{Quantifier}}$, capturing which variables are existentially or universally quantified. The prover then constructs and commits to each derived clause C_{f+i} (for $i = 1$ to R) using $\mathcal{F}_{\mathsf{Clause}}$. All clauses, both initial and derived, are appended in a commitment array managed by $\mathcal{F}_{\mathsf{FlexZKArray}}$, which supports secure and private clause retrieval for future steps in the protocol.

The core of the protocol proceeds iteratively over each proof step $i \in [1, R]$. In each iteration, the prover identifies the pair (k_i, l_i) and the pivot literal ℓ_i used to resolve C_{k_i} and C_{l_i} into an intermediate clause $\bar{C}i$, which is then reduced by universal reduction to derive C_i. The necessary clauses $[C_{k_i}]$ and $[C_{l_i}]$. They verify that the pivot ℓ_{p_i} is universally quantified and that $\bar{C}i$ is a correct result of resolving Ck_i and C_{l_i} on ℓ_{p_i} using $\exists$-Res, via the $\mathcal{F}_{\mathsf{Clause}}$ and $\mathcal{F}_{\mathsf{Quantifier}}$ functionalities.

Next, the protocol verifies the correctness of the universal reduction step $\bar{C}_i \vdash_{Q,\forall-RED} C_i$. To do so, the prover prepares three components: $[W_{\mathsf{res}}]$ (literals retained), $[W_{\mathsf{rem}}]$ (literals removed), and $[w_\ell]$ (the pivot's witness variable). The prover and verifier use $\mathcal{F}_{\mathsf{OrdLiteral}}$ to check that each retained literal precedes $[w_\ell]$ in the quantifier order, and each removed literal follows w_ℓ. They also verify that $[w_\ell]$ is existential and that all removed literals are universal, using $\mathcal{F}_{\mathsf{Quantifier}}$. Finally, $\mathcal{V}$ verifies through $\mathcal{F}_{\mathsf{Clause}}$ that $[C_i]$ is correctly derived using $[W_{\mathsf{res}}]$, $[W_{\mathsf{rem}}]$ and $[w_\ell]$.

Completing all R proof steps, the verifier checks that the final clause C_{f+R} is the empty clause. The protocol ends with both parties sending check calls to ensure that $\mathcal{F}_{\mathsf{FlexZKArray}}$ and $\mathcal{F}_{\mathsf{ZKQuantifier}}$ do not throw cheating.

Theorem 4. *The protocol in Figure 11 is a zero-knowledge proof of knowledge for falsifying QBFs whose clauses are already universally reduced.*

Our focus is on enabling efficient use of ZKP for Q-Res proof validation, rather than proposing a new generic ZK protocol. The resulting protocol satisfies completeness and zero-knowledge in a direct manner. In the case of a corrupted verifier, a simulator can extract the Q-Res proof by extracting clause indices from $\mathcal{F}_{\mathsf{FlexZKArray}}$. Verifying soundness then reduces to ensuring that the extracted proof is valid. This is guaranteed by the consistency of $\mathcal{F}_{\mathsf{FlexZKArray}}$, which returns the same clause on repeated reads. As long as each round of interaction for checking $\exists$-Res and $\forall$-Red steps, that includes res, equal, and retrieval, executes without causing $\mathcal{F}_{\mathsf{Clause}}$ to abort, and the final check step does not trigger cheating from either $\mathcal{F}_{\mathsf{OrdLiteral}}$ or $\mathcal{F}_{\mathsf{Quantifier}}$, the proof is valid according to Theorem 3.

We present our ZK protocol for validation of Q-Res proofs for true QBFs in Appendix Appendix A.

5 ZKPs of Winning Strategies

In this section, we present our protocol for verifying the correctness of a winning strategy as a valid witness for a quantified Boolean formula (QBF). We begin by describing how to ensure that the winning strategies are well-formed. Next, we present our approach for verifying the correctness of substituting quantified variables with their corresponding Herbrand or Skolem functions. Finally, we verify the unsatisfiability of the resulting propositional formula; this step follows the technique from [36] and is therefore omitted.

5.1 Herbrand Function Well-Formedness.

A Herbrand function assigns values to a subset of universally quantified variables by expressing each as a Boolean function over variables that precede them in the quantifier prefix. That is, a set of propositional Boolean functions $\mathcal{H} = \{f_x : \text{Pred}_\forall(x) \to \{0,1\} \mid x \in \mathcal{X}_\forall\}$. We represent $\mathcal{H}$ as a list of tuples:

$$H = \left\{ \left(x_i,\ \ell_i^a,\ \ell_i^b\right) \;\middle|\; 0 < i \leq h \right\}$$

where each tuple corresponds to a logical constraint of the form $x_i \iff \ell_i^a \wedge \ell_i^b$. Here, the output variable x_i is either a universally quantified variable or an auxiliary variable used to construct the Herbrand function. Formally, we require $x_i \in \mathcal{X}_\forall \cup \mathcal{X}_{\mathsf{aux}}$, where $\mathcal{X}_{\mathsf{aux}}$ denotes a set of auxiliary variables disjoint from the original variable set $\mathcal{X}$. The input literals ℓ_i^a and ℓ_i^b are drawn from the set $\mathcal{L} \cup \mathcal{L}_{\mathsf{aux}}$, where $\mathcal{L}$ is the set of literals over $\mathcal{X}$, and $\mathcal{L}_{\mathsf{aux}} = \{x, \neg x \mid x \in \mathcal{X}_{\mathsf{aux}}\}$ is the set of literals over auxiliary variables. Notice H can be committed as a list of literal tuples via $\mathcal{F}_{\mathsf{ZKLiteral}}$.

We define the *strategy dependency relation* $\prec_H$ over $L \cup \mathcal{L}_{\mathsf{aux}}$ induced by a falsification strategy H as:

$$\prec_H := \left\{ (\ell, \ell') \;\middle|\; (x_i,\ \ell_i^a,\ \ell_i^b) \in H,\ \ell \in \{\ell_i^a, \neg\ell_i^a,\ \ell_i^b,\ \neg\ell_i^b\}, \ell' \in \{x_i,\ \neg x_i\} \right\}.$$

The relation $\prec_H$ captures direct support dependencies between literals in the Herbrand representation: $\ell' \prec_H \ell$ if ℓ appears in the support of the variable defining ℓ'. We extend this to its transitive closure when needed to reason about indirect dependencies.

The list H corresponds to a *well-formed* Herbrand function w.r.t. a given QBF if the following conditions hold:

(1) *Uniqueness:* Each output variable is universally quantified and defined exactly once. That is, $x_i \neq x_j$ for all $i \neq j$, and $x_i \in \mathcal{X}_\forall \cup \mathcal{X}_{\mathsf{aux}}$.

(2) *Acyclicity:* The dependency relation $\prec_H$ induced by H must form a partial order over $\mathcal{L} \cup \mathcal{L}_{\mathsf{aux}}$; In other words, the dependency induced by H must be acyclic.

(3) *Prefix-consistency:* The order $\prec_H$ is consistent with the quantifier-induced order $\prec$ over $\mathcal{L}$. That is, for any pair $\ell \prec_H \ell'$, it holds that $\ell \prec \ell'$ under the quantifier prefix of the QBF.

These conditions ensure that each variable is uniquely defined, that no circular dependencies are introduced across definitions, and that the Herbrand function respects the quantifier structure of the QBF. We describe how each condition can be efficiently verified:

(1) *Uniqueness:* We verify that $\{x_i\} \subseteq \mathcal{X}_\forall \cup \mathcal{X}_{\mathsf{aux}}$ and contains no duplicates. Notice the set $\mathcal{X}_\forall \cup \mathcal{X}_{\mathsf{aux}}$ is public and contains no duplicate elements. This check ensures that no output variable in H is assigned more than once.

(2) *Acyclicity:* Assuming H is indexed in topological order w.l.g.. We can check acyclicity by checking:

$$\ell_i^a,\, \ell_i^b \in \mathcal{L}_\exists \cup \{x_j, \neg x_j \mid j < i\}$$

for $(x_i, \ell_i^a, \ell_i^b)$. This condition ensures that dependencies only refer to previously defined variables, eliminating cycles. We implement this through $\mathcal{F}_{\mathsf{FlexZKArray}}$.

(3) *Prefix-consistency:* We define a function $\lambda : \{x_i\} \to \mathbb{N}$ that computes the dependency level of each output as $\lambda(x_i) = \max(\mathsf{order}(\ell_i^a), \mathsf{order}(\ell_i^b))$, and $\mathsf{order}(\ell)$ is defined according to the order over $\mathcal{L}$ induced by the quantifier prefix, as follows:

$$\mathsf{order}(\ell) = \begin{cases} \xi(\ell) & \text{if } \ell \in \mathcal{L}, \\ \lambda(x_j) & \text{if } \ell = x_j \text{ or } \neg x_j \text{ for auxiliary } x_j. \end{cases}$$

The prefix-consistency can be verified by checking in ZK:

$$\lambda(x_i) \leq \xi(x_i) \quad \text{holds for all } x_i.$$

Theorem 5 (Herbrand Function Well-Formedness)**.** *Let $\Psi = Q.\phi$ be a closed QBF with prefix $Q = Q_1\mathcal{X}_1 \ldots Q_k\mathcal{X}_n$, and let $H = \{(x_i, \ell_i^a, \ell_i^b)\}_{i=1}^h$ be a Herbrand strategy expressed as a sequence of definitions. Assume H is indexed in topological order. Then H is a well-formed Herbrand function for Ψ if it satisfies all of the validation checks described above.*

We put the proof in Appendix Appendix D.

5.2 Variable Substitution

We now describe how to verify the correctness of a CNF formula Ψ, which is expected to be obtained by substituting the universally quantified variables in a QBF with their corresponding Herbrand functions (see Section 3). Performing this substitution directly in a zero-knowledge (ZK) setting is inherently difficult due to the complexity of evaluating Boolean functions under substitution. To address this challenge, we avoid direct substitution by leveraging the structure of CNF and instead verify that the target formula $\psi \wedge \psi_H$ and $\psi_H = \mathrm{T}(H)$. Here ψ is the propositional matrix of the original QBF and is publicly known, while ψ_H is a private witness representing the Tseitin CNF encoding of the Herbrand function H via the transformation T.

Our goal now is to check the correctness of the CNF ψ_H given a committed list of Herbrand function tuples. Recall that each tuple $(x_i, \ell_i^a, \ell_i^b) \in H$ specifies a Boolean constraint of the form $x_i \iff \ell_i^a \wedge \ell_i^b$. Applying the Tseitin transformation, this constraint can be equivalently encoded in CNF as the conjunction of three clauses: $TC_i^1 = (\neg x_i \vee \ell_i^a)$, $TC_i^2 = (\neg x_i \vee \ell_i^a)$, and $TC_i^3 = (x_i \vee \neg\ell_i^a \vee \neg\ell_i^b)$. Prover constructs the witness CNF as $\psi_H = \bigwedge_i \left(TC_i^1 \wedge TC_i^2 \wedge TC_i^3\right)$, and commits to it using a clause-level commitment scheme. To prove that each clause in ψ_H correctly represents its corresponding gate in H, we can then leverage the $\mathcal{F}_{\mathsf{ZKLiteral}}$ functionality to demonstrate membership of the expected literals in each clause that can be achieved by $\mathcal{F}_{\mathsf{Clause}}$.

5.3 Refinement for Skolem functions

When proving that a QBF $\Psi = Q.\psi$ is true (i.e., satisfiable), the strategy involves concretizing each existentially quantified variable with a corresponding Boolean function, unlike in the Herbrandization case, which targets a subset of universally quantified variables. Nevertheless, we can adopt the same structural approach by encoding Skolem functions as a list $S = \{(x_i, \ell_i^a, \ell_i^b)\}$, analogous to the representation used for Herbrand functions. To ensure completeness and correctness of the strategy, we also prove that every existential variable is assigned by checking if $\mathcal{X}_\exists \subseteq \{x_i\}$ holds.

6 Implementation and Evaluation

We implement two protocols for verifying QBF evaluation in ZK: ZKQRES, which is based on validating Q-Res proofs, and ZKWS, which is based on validating winning strategies. In this section, we focus on presenting results for false QBFs, as they constitute the majority of instances for which we can obtain the certificates (Q-Res proofs and Herbrand functions). Results for true QBFs are discussed in Appendix Appendix E.3.

6.1 Setup

Implementation dependency. We implement our protocols using the EMP-toolkit [57] as the backend for interactive zero-knowledge proofs, and incorporate the open-source implementation of ZKUNSAT [36]. We solve QBFs and collect Q-Resolution proofs by DepQBF and QRPcheck [45] and Skolem/Herbrand functions from CAQE [58][§].

Testbed. We perform our evaluation of our protocols on AWS `i4i.16xlarge` instances, each equipped with 64 vCPUs, 512 GB RAM, and 50 Gbps inter-instance bandwidth between the prover and verifier, unless stated otherwise. High-memory instances are chosen due to the substantial memory demands of our largest benchmarks.

We run the QBF solvers on compute nodes equipped with two Intel Xeon Gold 6148 CPUs (40 cores, 2.4 GHz) and 202 GB of RAM. This configuration is also used to generate and preprocess certificates, i.e., Q-Res proofs or winning strategies.

Benchmarks from QBFEVAL. We use QBF formulae from the QBFEVALbenchmark suite [59], a widely adopted collection for evaluating solvers in QBF reasoning. We evaluate ZKQRES and ZKWS against the 2007 and 2023 editions[¶]. We present results for QBFEVAL'07 in this section and include detailed results of QBFEVAL'23 in the appendix[‖]. We evaluate ZKQRES on 351 and ZKWS on 322 instances from QBFEVAL'07. The remaining instances are excluded due to solver timeouts, proof preprocessing failures, or certificate sizes exceeding 25 MB[**]. We describe the details of the instance selection procedure in the Appendix Appendix E.1.

Benchmarks from real-world applications

We generate benchmark instances that encode the real-world applications introduced in Section 1. The formulae are selected using the same procedure employed to obtain instances from the QBFEVAL'07 benchmark set in our evaluation pipeline.

PEC. The PEC benchmarks encode partial equivalence checking problems as QBFs. We use the benchmark suite provided in [17].

C-PLAN. We generate QBF encodings for Blocks World planning problems using Q-Planner [60]. Specifically, we extract five instances in which the plan length is strictly less than 5.

BBC. We use benchmarks for BBC with QBFs provided by [61]. These formulae encode instances of black-box bounded model checking, a setting in which the internal structure of specific modules is unknown. Solving such QBF encodings is generally considered a challenging task.

[§] We use earlier versions of DepQBF and CAQE, as recent releases do not support proof or certificate generation.

[¶] The certified solver track was discontinued after 2016. To the best of our knowledge, QBFEVAL'07 is the latest available benchmark suite that includes the track requiring certificates

[‖] Since 2023 benchmarks are not designed to evaluate certificate generation, we are only able to obtain proofs within our timeout configuration (described in Appendix Appendix E.2) for 25 instances from QBFEVAL'23.

[**] We set a 25MB limit, as the running time for larger proofs is estimated to exceed one hour. These instances are therefore excluded due to our resource constraints.

6.2 Evaluation on QBFEVAL

ZKQRES. We evaluate the communication cost and prover running time of ZKQRES for verifying Q-Res proofs across benchmark instances. Out of 351 instances, ZKQRES successfully verifies 328 in ZK within 1.3 hours, while running out of memory for the rest of the instances. About 72% of instances are verified within 100 seconds. The cumulative proportion of verified instances and the distribution of verification time are shown in Figure 2.

To understand the factors affecting the efficiency of our protocols, the results in Figure 12 are sorted by the number of clauses, proof width, and their product, which approximates the overall proof size. The results indicate that the protocol's performance on Q-Res proofs scales relatively linearly with the proof size. Our evaluation shows that our protocol can verify instances with up to 2,000 Q-Res steps and proof width as large as 2,000. When the proof width is reduced to around 300, the system can handle proofs with up to 87,000 steps in a similar time frame.

ZKWS. We evaluate the communication cost and prover running time of ZKWS for verifying Herbrand function benchmarks. Out of 322 instances, ZKWS successfully verifies 283 of them within about half an hour, while it runs out of memory for the rest of the instances. The largest instance yields Herbrand functions of size up to 33k, with a proof width of approximately 315. About 82% instances are verified within 100 seconds. The cumulative proportion of verified instances and the distribution of verification time are also shown in Figure 2.

Figure 13 presents the results, sorted by the number of clauses, proof width, proof size (i.e., the product of the number of clauses and proof width), the size of the CNF produced by Herbrandization, and the number of assignments in each Herbrand function. As the figure indicates, both communication and time costs scale approximately proportionally with the proof size, consistent with ZKUNSAT performance results.

To understand the primary cost source and performance bottleneck, we break down the cost components involved in verifying Herbrand functions, as shown in Figure 14. The results indicate that the unsatisfiability check remains the primary bottleneck in the overall verification process.

Comparison between ZKQRES and ZKWS. We do not find an absolute advantage of one approach across all instances. For example, for some instances, ZKQRES takes only 32 seconds, while it takes 2K seconds for ZKWS. On the other hand, the other instance that takes 4K seconds via ZKQRES, while it takes only 22 seconds when using Herbrand functions with ZKWS. We list the results in Figure 2 by comparing the distribution of verification time using different approaches.

6.3 Evaluation on Real-World Application

We also evaluate our protocols on QBF instances encoding real-world applications. The detailed results are listed in Appendix Tables 2 and 3, and Figure 21.

PEC. Our protocol successfully verifies a set of real-world circuit equivalence instances within 160 seconds using Q-Res. In contrast, the same instances can be verified in just 8 seconds using Herbrandization, highlighting the practical efficiency gains offered by our approach.

C-PLAN. For planning problems generated from the Q-Planner examples, ZKQRES successfully verifies the QBF instances encoding a blocks world like problem with step-length constraints less than 5 within 160 seconds. In contrast, ZKWS can not verify instances with step-lengths greater than 3 due to the prohibitive size of the certificates.

BBC. The largest instance of the BBC problem that can be verified by ZKWS has proof width 316 and involves 48,802 clauses. ZKWS takes 29 seconds to verify the correctness of Herbrandization. The dominant cost, however, remains in the ZKUNSAT phase, which verifies the unsatisfiability of

Herbranded QBF in about 300 seconds.

6.4 ZKUNSAT Optimization

We utilize the ZKUNSAT framework to prove the unsatisfiability of Herbrandized or Skolemized QBFs. The performance of ZKUNSAT depends linearly on the clause width, defined as the maximum number of literals in any clause appearing in either the formula or the resolution proof. To ensure ZK, all clauses are padded to this maximum width, regardless of their actual size.

We find that such padding leads to significant waste when proving knowledge of Herbrand/Skolem functions. In both the formula and the proof derived from Herbrandization/Skolemization, only a small fraction of the clauses have large literal width, while the majority of the clauses are narrow. To mitigate this inefficiency, we introduce a clause partitioning strategy that reduces padding waste while preserving soundness. The key idea is to split the proof into different buckets, each corresponding to a subset of clauses with similar width. Such partitioning allows us to process narrower clauses without incurring the padding cost of the widest ones. The steps of the procedure are shown in Appendix Appendix C. While this reveals a part of the information, this structural leakage reduces padding overhead substantially. Shown in Figure 15, the performance improves by approximately 50% when the bucket size is set to 10K.

7 Information Leakage - Efficiency Trade-off:

In this section, we present the information leakage - efficiency trade-off that our protocol offers. We begin by summarizing the information leaked by our protocols.

ZKQRES (false QBFs). We reveal upper bounds on the Q-Resolution proof width, on the number of Q-Resolution steps, and on the maximum number of literals removed in any $\forall-\textsc{Red}$.

ZKWS (false QBFs). We reveal upper bounds on the number of $\wedge$-gates in the AIG for the Herbrand function and, in ZKUNSAT (for proving Herbrandization is unsatisfiable), upper bounds on resolution proof width and length.

Mitigations and cost. Leakage can be reduced by padding while preserving soundness: re-deriving clauses to mask true proof length; using higher-degree polynomials to obscure proof width and the maximum literals removed per step; and introducing unused variables in the Herbrand/Skolem function to hide its true size. These defenses incur overhead, reflecting an inherent efficiency trade-off.

Hardness. The hardness of finding QBF proofs given such upper bounds is unknown. However, when bounds are set to natural worst-case values implied by the input QBF and its variable count, search remains at least as hard as without any auxiliary information.

8 Acknowledgements:

We thank Arijit Shaw and Anwar Hithnawi for many useful discussions, and Martina Seidl for providing the QBFEVAL 2007 benchmarks. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number [RGPIN-2024-05956]. Part of the computational work for this article was performed on the Niagara supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto. We also acknowledge CloudLab [62] for providing the research infrastructure for our artifact evaluation process.

References

[1] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract)," in *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 174–187.

[2] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Berlin, Heidelberg, May 2016, pp. 305–326.

[3] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, Berlin, Heidelberg, May 2013, pp. 626–645.

[4] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 238–252.

[5] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO 2013, Part II*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8043. Springer, Berlin, Heidelberg, Aug. 2013, pp. 90–108.

[6] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 781–796.

[7] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," in *NDSS 2015*. The Internet Society, Feb. 2015.

[8] Z. Hu, P. Mohassel, and M. Rosulek, "Efficient zero-knowledge proofs of non-algebraic statements with sublinear amortized cost," in *CRYPTO 2015, Part II*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9216. Springer, Berlin, Heidelberg, Aug. 2015, pp. 150–169.

[9] P. Mohassel, M. Rosulek, and A. Scafuro, "Sublinear zero-knowledge arguments for RAM programs," in *EUROCRYPT 2017, Part I*, ser. LNCS, J.-S. Coron and J. B. Nielsen, Eds., vol. 10210. Springer, Cham, Apr. / May 2017, pp. 501–531.

[10] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 2087–2104.

[11] J. Bootle, A. Cerulli, J. Groth, S. K. Jakobsen, and M. Maller, "Arya: Nearly linear-time zero-knowledge proofs for correct program execution," in *ASIACRYPT 2018, Part I*, ser. LNCS, T. Peyrin and S. Galbraith, Eds., vol. 11272. Springer, Cham, Dec. 2018, pp. 595–626.

[12] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni, "Public-coin zero-knowledge arguments with (almost) minimal time and space overheads," in *TCC 2020, Part II*, ser. LNCS, R. Pass and K. Pietrzak, Eds., vol. 12551. Springer, Cham, Nov. 2020, pp. 168–197.

[13] D. Heath and V. Kolesnikov, "A 2.1 KHz zero-knowledge processor with BubbleRAM," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 2055–2074.

[14] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni, "Time- and space-efficient arguments from groups of unknown order," in *CRYPTO 2021, Part IV*, ser. LNCS, T. Malkin and C. Peikert, Eds., vol. 12828. Virtual Event: Springer, Cham, Aug. 2021, pp. 123–152.

[15] D. Heath, Y. Yang, D. Devecsery, and V. Kolesnikov, "Zero knowledge for everything and everyone: Fast zk processor with cached oram for ansi c programs," in *IEEE SP 2021*. IEEE, 2021, pp. 1538–1556.

[16] N. Franzese, J. Katz, S. Lu, R. Ostrovsky, X. Wang, and C. Weng, "Constant-overhead zero-knowledge for RAM programs," in *ACM CCS 2021*, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 178–191.

[17] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence checking of partial designs using dependency quantified boolean formulae," in *ICCD*. IEEE, 2013, pp. 396–403.

[18] U. Egly, M. Kronegger, F. Lonsing, and A. Pfandler, "Conformant planning as a case study of incremental qbf solving," *Annals of Mathematics and Artificial Intelligence*, vol. 80, pp. 21–45, 2017.

[19] J. Rintanen *et al.*, "Asymptotically optimal encodings of conformant planning in qbf," in *AAAI*, vol. 2007, 2007, pp. 1045–1050.

[20] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," in *PSTV 1999*. Springer, 1999, pp. 225–240.

[21] A. Shamir, "Ip= pspace," *Journal of the ACM (JACM)*, vol. 39, no. 4, pp. 869–877, 1992.

[22] R. Impagliazzo and M. Yung, "Direct minimum-knowledge computations," in *Eurocrypt 1987*. Springer, 1987, pp. 40–51.

[23] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge," in *Advances in Cryptology—CRYPTO'88: Proceedings 8*. Springer, 1990, pp. 37–56.

[24] M. Benedetti, "Evaluating qbfs via symbolic skolemization," in *LPAR 2005*. Springer, 2005, pp. 285–300.

[25] T. Jussila, C. Sinz, and A. Biere, "Extended resolution proofs for symbolic sat solving with quantification," in *SAT 2006*. Springer, 2006, pp. 54–60.

[26] C. Sinz and A. Biere, "Extended resolution proofs for conjoining bdds," in *CSR 2006*. Springer, 2006, pp. 600–611.

[27] O. Beyersdorff, L. Chew, and M. Janota, "On unification of qbf resolution-based calculi," *Electron. Colloquium Comput. Complex.*, vol. TR14, 2014.

[28] O. Beyersdorff and B. Böhm, "Understanding the relative strength of qbf cdcl solvers and qbf resolution," *LMCS 2023*, vol. 19, 2023.

[29] O. Beyersdorff, J. Blinkhorn, and M. Mahajan, "Building strategies into qbf proofs," *Journal of Automated Reasoning*, vol. 65, pp. 125 – 154, 2020.

[30] O. Beyersdorff, L. Chew, and M. Janota, "Proof complexity of resolution-based qbf calculi," *Electron. Colloquium Comput. Complex.*, vol. TR14, 2015.

[31] O. Beyersdorff, L. Hinde, and J. Pich, "Reasons for hardness in qbf proof systems," *TOCT 2018*, vol. 12, pp. 1 – 27, 2018.

[32] O. Beyersdorff, I. Bonacina, L. Chew, and J. Pich, "Frege systems for quantified boolean logic," *Journal of the ACM (JACM)*, vol. 67, pp. 1 – 36, 2020.

[33] A. Das, "Alternating time bounds from variants of focussed proof systems," *Preprint*, 2017.

[34] P. Czerner, J. Esparza, and V. Krasotin, "A resolution-based interactive proof system for unsat," in *FoSSaCS*. Springer, 2024, pp. 116–136.

[35] J. Kolesar, S. Ali, T. Antonopoulos, and R. Piskac, "Coinductive proofs of regular expression equivalence in zero knowledge," 2025. [Online]. Available: https://arxiv.org/abs/2504.01198

[36] N. Luo, T. Antonopoulos, W. R. Harris, R. Piskac, E. Tromer, and X. Wang, "Proving unsat in zero knowledge," in *ACM CCS 2022*. ACM Press, 2022, pp. 2203–2217.

[37] D. Luick, J. C. Kolesar, T. Antonopoulos, W. R. Harris, J. Parker, R. Piskac, E. Tromer, X. Wang, and N. Luo, "Zksmt: A vm for proving smt theorems in zero knowledge," in *USENIX Security*. USENIX Association, 2024, pp. 3837–3845.

[38] E. Laufer, A. Ozdemir, and D. Boneh, "zkpi: Proving lean theorems in zero-knowledge," in *ACM CCS 2024*. CCS Press, 2024, pp. 4301–4315.

[39] K. Mastel and W. Slofstra, "Two prover perfect zero knowledge for mip," in *STOC 2024*. ACM Press, 2024, pp. 991–1002.

[40] A. Chiesa, M. A. Forbes, T. Gur, and N. Spooner, "Spatial isolation implies zero knowledge even in a quantum world," *ACM Journal of the ACM (JACM)*, vol. 69, no. 2, pp. 1–44, 2022.

[41] E. Ben-Sasson, A. Chiesa, M. A. Forbes, A. Gabizon, M. Riabzev, and N. Spooner, "On probabilistic checking in perfect zero knowledge," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 988, 2016.

[42] R. Jain, Z. Ji, S. Upadhyay, and J. Watrous, "Qip= pspace," *Journal of the ACM (JACM)*, vol. 58, no. 6, pp. 1–27, 2011.

[43] "Resolution for quantified boolean formulas," *Information and Computation*, vol. 117, no. 1, pp. 12–18, 1995.

[44] V. Balabanov and J.-H. R. Jiang, "Resolution proofs and skolem functions in qbf evaluation and applications," in *CAV 2021*. Springer, 2011, pp. 149–164.

[45] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere, "Resolution-based certificate extraction for qbf: (tool presentation)," in *SAT 2012*. Springer, 2012, pp. 430–435.

[46] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *17th ACM STOC*. ACM Press, May 1985, pp. 291–304.

[47] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *Journal of the ACM*, vol. 38, no. 3, pp. 691–729, Jul. 1991.

[48] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *39th ACM STOC*, D. S. Johnson and U. Feige, Eds. ACM Press, Jun. 2007, pp. 21–30.

[49] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *40th ACM STOC*, R. E. Ladner and C. Dwork, Eds. ACM Press, May 2008, pp. 113–122.

[50] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT 2010*, ser. LNCS, M. Abe, Ed., vol. 6477. Springer, Berlin, Heidelberg, Dec. 2010, pp. 321–340.

[51] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently," in *ACM CCS 2013*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 955–966.

[52] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *34th ACM STOC*. ACM Press, May 2002, pp. 494–503.

[53] D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos, "Zero-knowledge proofs for set membership: Efficient, succinct, modular," *Designs, Codes and Cryptography*, vol. 91, no. 11, pp. 3457–3525, 2023.

[54] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning," in *USENIX Security 2021*. USENIX Association, 2021, pp. 501–518.

[55] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *IEEE SP 2021*. IEEE, 2021, pp. 1074–1091.

[56] C. Baum, L. Braun, A. Munch-Hansen, B. Razet, and P. Scholl, "Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and z2k," in *ACM CCS 2021*. ACM Press, 2021, pp. 192–211.

[57] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[58] M. N. Rabe and L. Tentrup, "Caqe: a certifying qbf solver," in *FMCAD 15*. IEEE, 2015, pp. 136–143.

[59] "QBFEVAL 2023 Benchmark Gallery," https://qbf23.pages.sai.jku.at/gallery/, accessed: 2025-06-01.

[60] I. Shah, "Q-Planner: Quantified Planning with QBF," https://github.com/irfansha/Q-Planner, 2024, accessed: 2025-04-15.

[61] M. Seidl, F. Lonsing, and A. Biere, "qbf2epr: A tool for generating epr formulas from qbf." *PAAR@ IJCAR 2012*, vol. 21, pp. 139–148, 2012.

[62] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[63] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Clause/term resolution and learning in the evaluation of quantified boolean formulas," *J. Artif. Int. Res.*, vol. 26, no. 1, p. 371–416, Aug. 2006.

[64] A. Mishchenko, "Abc: A system for sequential synthesis and verification," https://people.eecs.berkeley.edu/~alanmi/abc/, 2025, accessed: 2025-04-15.

A Proving True PCNF in ZK via Q-Res

A QBF $\Psi = \Pi\psi$ in PCNF, is true if and only if the empty cube is derivable by Q-cube resolution and the model generation rule [63]. We now introduce Q-cube resolution proofs and the model generation rule.

Cubes. A cube is defined to be the conjunction of a set of literals. For example, $(a \wedge b \wedge \neg c)$.

Q-Cube Resolution Step. A Q-cube resolution step is the derivation of a non-contradictory cube through the resolution of 2 cubes over a universally quantified variable.

Existential Reduction. An Existential reduction of a cube is the removal of all existential variables that do not preceed any universally quantified variable.

Q-Cube Resolution Proof. A Q-cube resolution proof is the derivation of the empty cube from a set of initial cubes through the application of Q-cube resolution steps and existential reduction. The initial cubes of a QBF in PCNF are a subset of the cubes derived by the model generation rule.

Model Generation Rule. A set of cubes Φ is said to be derivable by the model generation rule if all of the following are true:

- For each cube $D \in \Phi$ and each clause $C \in \psi$, $D \cap C \neq \phi$ (where ϕ is the empty set) and D is non-contradictory.
- The disjunction of all cubes in Φ is propositionally logically equivalent to ψ.

Proving True QBFs in ZK. By switching the quantifiers in the Q-Res protocol, we can convert it from a Q-Res checking protocol into a Q-Cube-Res checking protocol. However, since the public QBF is in PCNF, as an additional step, the prover needs to convince the verifier that the set of initial cubes they are operating with are valid (i.e. derivable via model generation rule and the existential reduction of this derived cube). We call the initial cube before existential reduction the pre-initial cube.

Proving Initial Cube Validity in ZK. Suppose the prover wants to prove that the pre-initial cube $s = \{l_1, \ldots, l_n\}$ is valid. If s was a valid pre-initial cube, then by definition of the model generation rule for each clause C in the public QBF ψ, there must exist some $l_{C_s} \in C \cap s$. For each clause $C \in \psi$, the prover gathers the literal l_{C_s}, then the prover and verifier then invoke $\mathcal{F}_{\mathsf{ZKSet}}$ to check the subset relation $\bigcup_{C \in \psi} l_{C_s} \subseteq s$ and $l_{C_s} \subseteq C$. Now, all that remains is to prove that s is non-contradictory. After this the prover and verifier can use the existentially reduced version of s as the initial cube for the Q-cube resolution.

Proving Non-Contradictory (Non-Tautological) Cubes (Clauses) in ZK To prove that the cube (clause) s is non-contradictory (non-tautological), the prover commits another cube (clause)

s' such that $l \in s \iff \neg l \in s'$ and proceeds to prove that the GCD of the two polynomials $GCD(\gamma(s)(X), \gamma(s')(X)) = 1$ by committing $p(X)$ and $q(X)$ such that $1 = p(X) * (\gamma(s)(X)) + q(X) * (\gamma(s')(X))$.

B Functionalties

C ZKUNSAT Optimization

Algorithm 1 Clause Bucketing and Memory Array Construction

1: Choose a parameter k that determines the number of clauses per bucket.
2: Partition the set of clauses into buckets $\{G_1, G_2, \dots\}$ such that each bucket G_j contains k clauses.
3: **for** each bucket G_j **do**
4: Compute $w_j \leftarrow \max\{\text{width}(C) \mid C \in G_j\}$
5: Construct a clause memory array storing all clauses in G_j, each padded to width w_j

D Proof of Theorem 5

Proof: We prove both directions.

(Only if). Assume H is a well-formed Herbrand function. Then by definition:

- Each output variable is uniquely defined and drawn from $\mathcal{X}_\forall \cup \mathcal{X}_{\mathsf{aux}}$, implying Condition (1).
- The dependency graph induced by H is acyclic. Since H is topologically ordered, each variable x_i only depends on earlier ones, satisfying Condition (2).
- Well-formedness requires that every universal variable $x_i \in \mathcal{X}_\forall$ only depends on literals of variables that appear before it in the quantifier prefix. The recursive function $\lambda(x_i)$ captures this dependency level. Thus, $\lambda(x_i) \leq \xi(x_i)$, satisfying Condition (3).

(If). Assume Conditions (1), (2), and (3) hold.

- Condition (1) ensures that H defines a function: each x_i is assigned exactly once and from a valid domain.
- Condition (2) ensures acyclicity: since x_i depends only on variables x_j with $j < i$, the dependency graph has no cycles. This allows topological evaluation and confirms that H is well-founded.
- Condition (3) ensures prefix-consistency. The function $\lambda(x_i)$ computes the highest-order dependency of x_i. If $\lambda(x_i) \leq \xi(x_i)$, then all literals used in defining x_i come from earlier in the prefix. Thus, the universal variable x_i does not depend on any variable that appears later in the quantifier prefix.

Hence, H is a syntactically well-defined and prefix-consistent Herbrand function. This completes the proof.

E Detailed Evaluation Results

E.1 QBFEVAL 2007 Instance Selection

We evaluate the performance of ZKQRES on 351 out of 1,136 instances from the benchmark set. DepQBF produces complete Q-resolution proof traces for 398 false QBFs and 81 true QBFs within a 5-minute timeout. This timeout is chosen as DepQBF already generates very large traces (exceeding 10 GB) within this time frame. Among these instances, the proof preprocessing step times out on 47 false QBFs, resulting in 351 false QBFs and 81 true QBFs against which ZKQRES can be evaluated.

We evaluate the performance of ZKWS on 405 out of 1,136 instances from the benchmark set. CAQE successfully generates Skolem or Herbrand function certificates for 425 instances—332 unsatisfiable and 93 satisfiable. We apply ABC [64] to minimize the extracted strategies. ABC's circuit minimization times out on 10 unsatisfiable and 3 satisfiable instances. For the remaining cases, SAT solving and proof processing complete successfully on 322 unsatisfiable and 20 satisfiable instances, resulting in ZKWS evaluation on 342 instances [††].

E.2 Detailed Evaluation on QBFEVAL 2023

E.3 Detailed Evaluation on True QBFs.

E.4 Verifying Skolemization

We are unable to benchmark Skolemization functions on as many instances as we do for Herbrand functions for true QBFs, primarily due to the fact that the number of variables in the Skolemization

[††]For each QBF clause with $n \geq 2$ literals, skolemization introduces at least $n - 1$ new variables, likely making to frequent timeouts in picosat and our resolution proof preprocessor.

Instance Name	ZKWS			ZKQRES		
	Time (s)	Degree	#Steps	Time (s)	Degree	QRES Steps
arbiter-05-comp-error01-qbf-hardness-depth-8	27.3519	76	5966	3.73424	41	458
arbiter-06-comp-error01-qbf-hardness-depth-11	1375.63	346	51800	–		
gttt_2_1_001020_4x4_torus_w	8172.08	704	118329	–		
gttt_2_2_000111_4x4_torus_w	6033.31	644	95601	–		
gttt_2_2_000111_4x4_w	6690.66	680	97363	–		
hein_14_5x5-07.pg	7576.82	650	127124	438.279	107	38766
neclaftp4001	9.14755	9	3716	–		
ntrivil_query71_1344n	7.85795	50	666	17.8503	102	1276
W4-Umbrella_tbm_05.tex.moduleQ3.8S.000001	1695.84	1349	22621	3718.06	1355	12816
W4-Umbrella_tbm_25.tex.moduleQ3.7S.000003	2021.18	1568	21876	5807.78	1570	15799
W4-Umbrella_tbm_26.tex.moduleQ3.7S.000003	420.314	875	8014	1639.92	878	11519
GuidanceService	–			68.9327	181	2337
hein_07_4x4-07.pg	–			566.292	90	61943

Table 1: Evaluation on QBFEVAL 2023 Benchmarks.

is significantly higher than the number of variables in the QBF. This increases clause width in the proofs produced by picosat, making it hard for us to use ZKUNSAT. Figure 18 presents our evaluation results for the subset of instances we are able to verify.

E.5 Q-Res for True QBFs via Cube Resolution

E.6 Detailed Evaluation on Use Cases.

Name	Herbrand Time (s)	Herbrand Comm. (MB)	QRP Time (s)	QRP Comm. (MB)
z4ml.blif_0.10_0.20_0_1_inp_exact	1.18	8.84	0.85	4.09
z4ml.blif_0.10_0.20_0_1_out_exact	–	–	0.71	2.13
z4ml.blif_0.10_1.00_0_1_inp_exact	1.17	8.84	0.79	4.09
z4ml.blif_0.10_1.00_0_1_out_exact	1.17	5.93	0.75	2.13
comp.blif_0.10_0.20_0_1_out_exact	1.56	24.57	1.54	15.03
C499.blif_0.10_0.20_0_1_out_exact	2.14	51.68	0.90	7.52
C499.blif_0.10_1.00_0_1_out_exact	–	–	0.96	8.12
C499.blif_0.10_1.00_0_1_inp_exact	–	–	1.35	10.98
C880.blif_0.10_1.00_0_1_out_exact	2.34	62.91	–	–
C880.blif_0.10_1.00_0_1_inp_exact	3.23	72.93	–	–
term1.blif_0.10_0.20_0_1_out_exact	2.74	72.09	–	–
term1.blif_0.10_1.00_0_1_out_exact	3.00	77.27	–	–
term1.blif_0.10_1.00_0_1_inp_exact	3.23	80.77	–	–
C6288.blif_0.10_0.20_0_1_out_exact	7.70	326.57	55.49	387.23
C6288.blif_0.10_1.00_0_1_out_exact	8.65	371.79	165.72	933.17
C5315.blif_0.10_1.00_0_1_out_exact	–	–	0.73	2.96
C432.blif_0.10_1.00_0_1_out_exact	–	–	1.60	16.25
comp.blif_0.10_1.00_0_1_inp_exact	–	–	2.14	25.15

Table 2: ZK Validation: PEC Benchmarks

Instance Name	Herbrand Time (s)	Herbrand Comm. (MB)	ZKUNSAT Time (s)	ZKUNSAT Comm. (MB)
prob01, plan length = 1	6.01	13.82	2.20	16.56
prob01, plan length = 2	18.55	72.60	4.15	49.86
prob01, plan length = 3	153.28	599.54	155.99	1238.06
prob01, plan length = 4	915.44	2879.52	–	–
prob01, plan length = 5	1105.94	8373.44	–	–

Table 3: ZK QRP validation: Q-Planner Benchmarks (Blocks Q-Planner)

Protocol CheckProof

Inputs: Both parties are given a QBF of the form $Q_1\mathcal{X}_1Q_2\mathcal{X}_2\cdots Q_k\mathcal{X}_k\psi$, where $\psi = C_1 \vee \cdots \vee C_L$. The prover ($\mathcal{P}$) holds a Q-RES refutation encoded as a sequence of tuples $(k_1, l_1), \ldots, (k_R, l_R)$. Both parties know the proof length R, proof width w, and deduction degree d.

Protocol:

1. $\mathcal{P}$ and $\mathcal{V}$ obtain $[C_i]$ for $i \in [1, L]$ using $\mathcal{F}_{\mathsf{OrdLiteral}}$ with the order induced by the quantifier prefix $Q_1\mathcal{X}_1 \cdots Q_k\mathcal{X}_k$, and $\mathcal{F}_{\mathsf{Clause}}$. They also compute $[\mathcal{L}_\exists]$ and $[\mathcal{L}_\forall]$ using $\mathcal{F}_{\mathsf{Quantifier}}$.
2. $\mathcal{P}$ locally derives each clause C_{L+i} from the Q-RES proof and commits them using $\mathcal{F}_{\mathsf{Clause}}$.
3. Both parties send $(\mathsf{Init}, L + R, [C_1], \ldots, [C_{L+R}])$ to $\mathcal{F}_{\mathsf{FlexZKArray}}$.
4. For each $i \in [1, R]$, the parties perform the following steps:
   (a) $\mathcal{P}$ looks up the tuple (k_i, l_i) such that C_{k_i} and C_{l_i} resolve on pivot ℓ_i to yield $\bar{C}_i$, and $\bar{C}_i$ reduces via $\forall$-REDto C_i. The parties compute $[\ell_i]$ and $[\bar{C}_i]$ using $\mathcal{F}_{\mathsf{OrdLiteral}}$ and $\mathcal{F}_{\mathsf{Clause}}$.
   (b) **Fetching premises:** $\mathcal{P}$ sends $(\mathsf{Read}, l_i, C_{l_i}, i)$ to $\mathcal{F}_{\mathsf{FlexZKArray}}$, while $\mathcal{V}$ sends (Read, i); both obtain $[C_{l_i}]$. They similarly retrieve $[C_{k_i}]$ and $[C_i]$.
   (c) **Checking $\exists$-Res:** $\mathcal{P}$ finds the corresponding pivot literal ℓ_{p_i} and authenticates it. $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{res}, [C_{l_i}], [C_{k_i}], [\ell_{p_i}], [\bar{C}_i],)$ to $\mathcal{F}_{\mathsf{Clause}}$, and $(\mathsf{Check}, [\ell_{p_i}], \forall)$ to $\mathcal{F}_{\mathsf{Quantifier}}$.
   (d) **Checking $\forall$-Red:**
      i. **Preparing extended witness:** $\mathcal{P}$ prepares and commits W_{res}, W_{rem}, and w_ℓ based on Equations 9, 8, 10, and 11, corresponding to the universal reduction step $\bar{C}i \vdash_{Q,\forall-RED} C_i$. $\mathcal{P}$ sends $(\mathsf{Retrieval}, \{\ell_1, \ldots, \ell_k\}, [W_{\mathsf{res}}])$ and $\mathcal{V}$ sends $(\mathsf{Retrieval}, [W_{\mathsf{res}}])$ to $\mathcal{F}_{\mathsf{ZKClause}}$, both obtaining $\{[\ell_1], \ldots, [\ell_k]\}$. The same for W_{rem} and obtain $\{[\ell'_1], \ldots, [\ell'_d]\}$. $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{order}, [w_\ell], \mathsf{ord}_w)$ and $(\mathsf{order}, [w_\ell])$ to $\mathcal{F}_{\mathsf{OrdLiteral}}$ to obtain $[\mathsf{ord}_w]$.
      ii. **Checking Equation (8):** For each $[\ell_i] \in \{[\ell_1], \ldots, [\ell_k]\}$, $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{order}, [\ell_i], \mathsf{ord}_i)$ and $(\mathsf{order}, [\ell_i])$ to $\mathcal{F}_{\mathsf{OrdLiteral}}$, then check $(>, [\mathsf{ord}_i], [\mathsf{ord}_w])$ via $\mathcal{F}_{\mathsf{ZK}}$.
      iii. **Checking Equation (9):** For each $\ell'_i \in \{[\ell'_1], \ldots, [\ell'_d]\}$, $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{order}, [\ell'_i], \mathsf{ord}'_i)$ and $(\mathsf{order}, [\ell'_i])$ to $\mathcal{F}_{\mathsf{OrdLiteral}}$, then check $(<, [\mathsf{ord}'_i], [\mathsf{ord}_w])$ via $\mathcal{F}_{\mathsf{ZK}}$.
      iv. **Checking Equation (10):** $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{Check}, [w_\ell], \exists)$ to $\mathcal{F}_{\mathsf{Quantifier}}$. For each $\ell'_i \in \{[\ell'_1], \ldots, [\ell'_d]\}$, $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{Check}, [\ell'_i], \forall)$ to $\mathcal{F}_{\mathsf{Quantifier}}$
      v. **Checking Equation (11):** $\mathcal{P}$ and $\mathcal{V}$ obtain $[C_\ell]$ for $C_\ell = \{w_\ell\}$ using $\mathcal{F}_{\mathsf{Clause}}$, then invoke $(\mathsf{Equal}, [C_r], \{[W_{\mathsf{res}}], [C_\ell]\})$ and $(\mathsf{Equal}, [C], \{[W_{\mathsf{res}}], [C_\ell], [W_{\mathsf{rem}}]\})$ via $\mathcal{F}_{\mathsf{Clause}}$.
5. After all R iterations, the parties use $\mathcal{F}_{\mathsf{Clause}}$ to verify that $[C_R] = \bot$; if not, $\mathcal{V}$ aborts.
6. The parties send (check) to both $\mathcal{F}_{\mathsf{FlexZKArray}}$ and $\mathcal{F}_{\mathsf{ZKQuantifier}}$; if either returns $(\mathsf{cheating})$, $\mathcal{V}$ aborts.

Figure 11: Protocol for checking a Q-RES proof in zero knowledge.

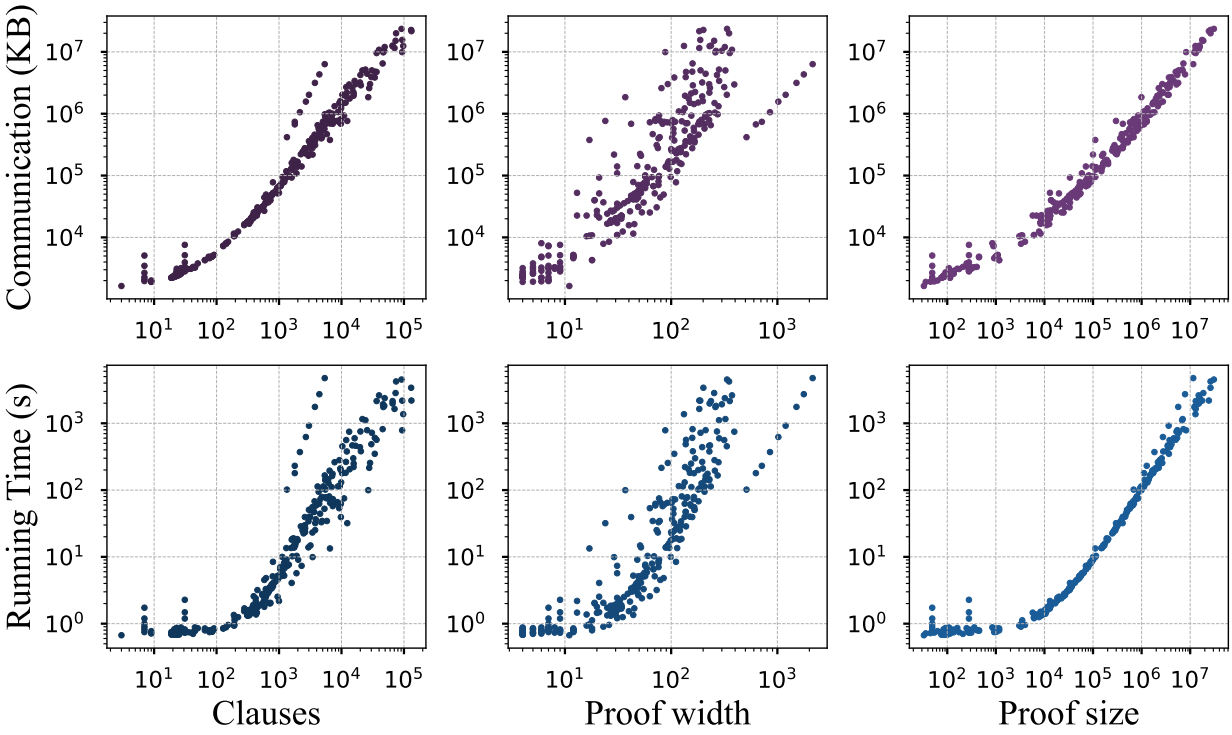

Figure 12: **Communication and time cost of ZKQRES.** Results are sorted by the number of clauses, proof width, and their product (which approximates the total proof size). The results indicate that the protocol's performance on Q-Res proofs scales closely with this product.

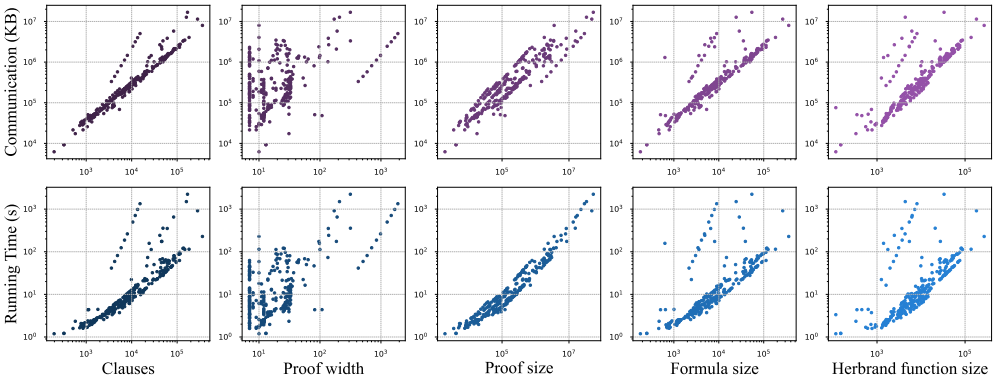

Figure 13: **Communication cost and prover running time for Herbrand function benchmarks.** Results are sorted by number of clauses, proof width, and their product, reflecting the approximate proof size. Also shown are the size of CNFs resulting from Herbrandization and the size of the Herbrand functions themselves. Both communication and running time correlate most closely with the product of clause count and width, consistent with theoretical expectations of ZKUNSAT's complexity.

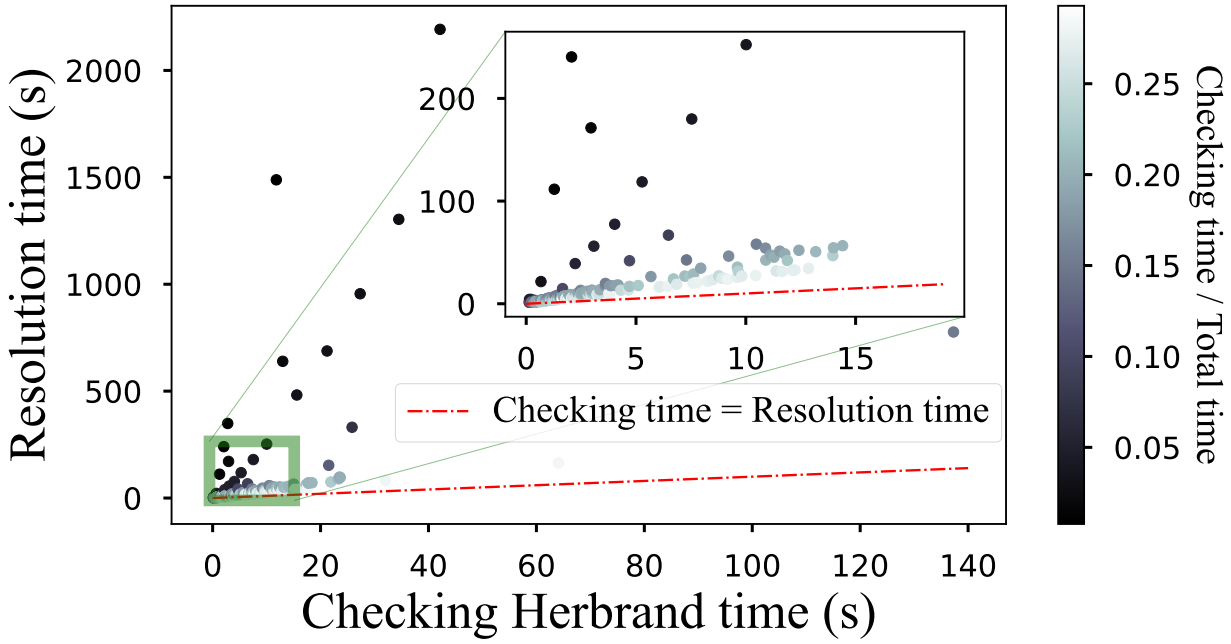

Figure 14: **Decomposition of the verification time of ZKWS.** The time costs are split into two components: (1) the time for checking Herbrandization correctness, which includes both well-formedness and correctness of the corresponding CNF encoding; and (2) the time for ZKUNSAT to verify UNSAT of the CNF produced by Herbrandization. Darker points indicate a larger proportion of time spent in ZKUNSAT, with the red line marking equal contribution from both components. The results show that ZKUNSAT is the primary performance bottleneck in ZKWS.

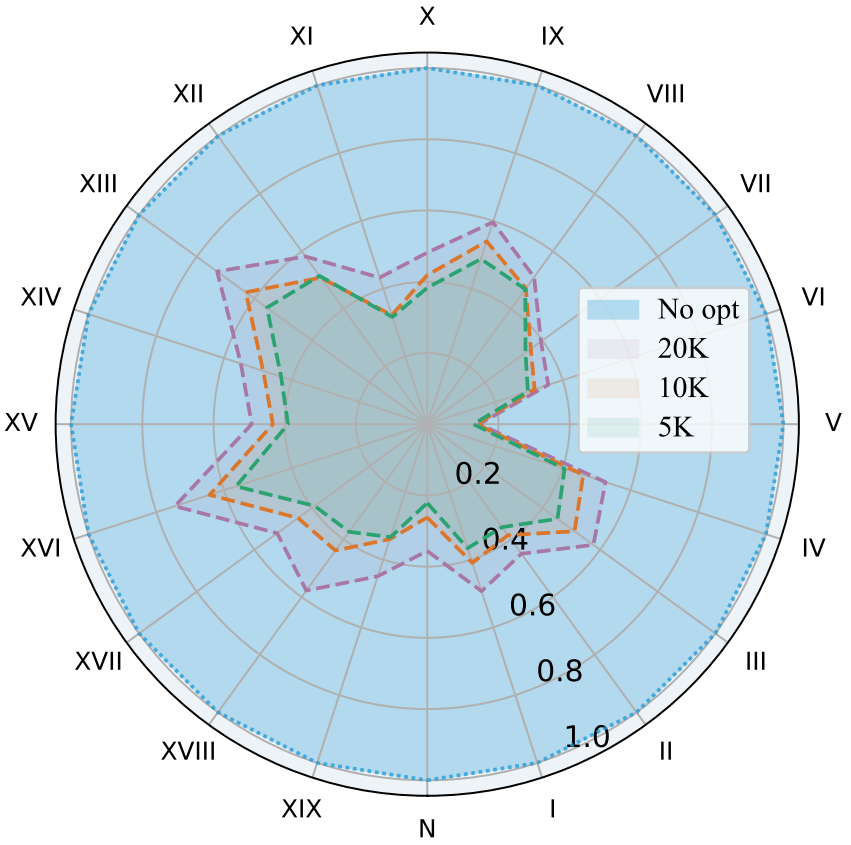

Figure 15: **Running time for instance verification via Herbrandization w/wo clause partitioning optimization.** Each radius represents a QBF instance. We present the ratio of verification time after applying our optimization scheme with 5K, 10K, and 20K buckets, respectively. The results show that, for most instances, the time cost is reduced by half when using 10K buckets. For certain instances, the optimization achieves up to a 90% improvement with 5K buckets.

Functionality $\mathcal{F}_{\mathsf{FlexZKArray}}$

Array initialization: On receiving $(\mathsf{Init}, N, [m_0], \ldots, [m_{N-1}])$ from $\mathcal{P}$ and $\mathcal{V}$, where $m_i \in \mathbb{F}$, store the $\{m_i\}$ and set $f := \mathsf{honest}$ and ignore subsequent initialization calls.

Array read: On receiving $(\mathsf{Read}, \ell, d, t)$ from $\mathcal{P}$, and (Read, t) from $\mathcal{V}$, where $d \in \mathbb{F}$ and $\ell, t \in \mathbb{N}$, send $[d]$ to each party. If $d \neq m_\ell$ or t from both parties do not match or $\ell \geq t$ then set $f := \mathsf{cheating}$.

Array check: Upon receiving (check) from $\mathcal{V}$ do: If $\mathcal{P}$ sends (cheat) then send $\mathsf{cheating}$ to $\mathcal{V}$. If $\mathcal{P}$ sends $(\mathsf{continue})$ then send f to $\mathcal{V}$.

Figure 16: Functionality for append-only arrays in ZK.

Protocol Π_{Clause}

Res [36]:

1. $\mathcal{P}$ locally computes $W_0(X), W_1(X)$ and ℓ_p, such that $W_0(X) \cdot \gamma(C_0)(X) = \gamma(C_r)(X) \cdot (X + \epsilon(\ell_p))$ and $W_1(X) \cdot \gamma(C_1)(X) = \gamma(C_r)(X) \cdot (X + \epsilon(\neg\ell_p))$. Note that the degree of $W_0(X)$ and $W_1(X)$ are bounded by w.
2. $\mathcal{P}$ locally computes $\rho(X) = X - \epsilon(\ell_p)$, of which the degree is bounded by 1.
3. Two parties use $\mathcal{F}_{\mathsf{ZK}}$ to authenticate all $w+1$ polynomial coefficients in $W_0(X)$ and $W_1(X)$, and two polynomial coefficients in $\rho(X)$. As a result, two parties get $[W_0(X)]$, $[W_1(X)]$ and $[\rho(X)]$.
4. Using $\mathcal{F}_{\mathsf{ZK}}$, two parties check that the highest coefficient in $[\rho(X)]$ is non-zero, this make sense that $[\rho(X)]$ has degree exactly 1.
5. $\mathcal{P}$ locally computes polynomial $\bar{\rho}(X) = \rho(1_{\mathbb{F}} - X)$ and commits its 2 coefficients to obtain $[\bar{\rho}(X)]$. Then two parties check that the committed coefficients satisfy $\bar{\rho}(X) = \rho(1_{\mathbb{F}} - X)$.
6. Both parties send $(\mathsf{PoPEqCheck}, X, ([W_0(X)], [\gamma(C_0)(X)]), ([\gamma(C_r)(X)], [\rho(X)]))$ to $\mathcal{F}_{\mathsf{ZK}}$.
7. Both parties send $(\mathsf{PoPEqCheck}, X, ([W_1(X)], [\gamma(C_1)(X)]), ([\gamma(C_r)(X)], [\bar{\rho}(X)]))$ to $\mathcal{F}_{\mathsf{ZK}}$.
8. As an additional step for Q-RES or Q-Cube Res, Non-tautological clause checking is performed on C_r as outlined in Appendix-Appendix A

Figure 17: Checking resolution in ZKUNSAT.

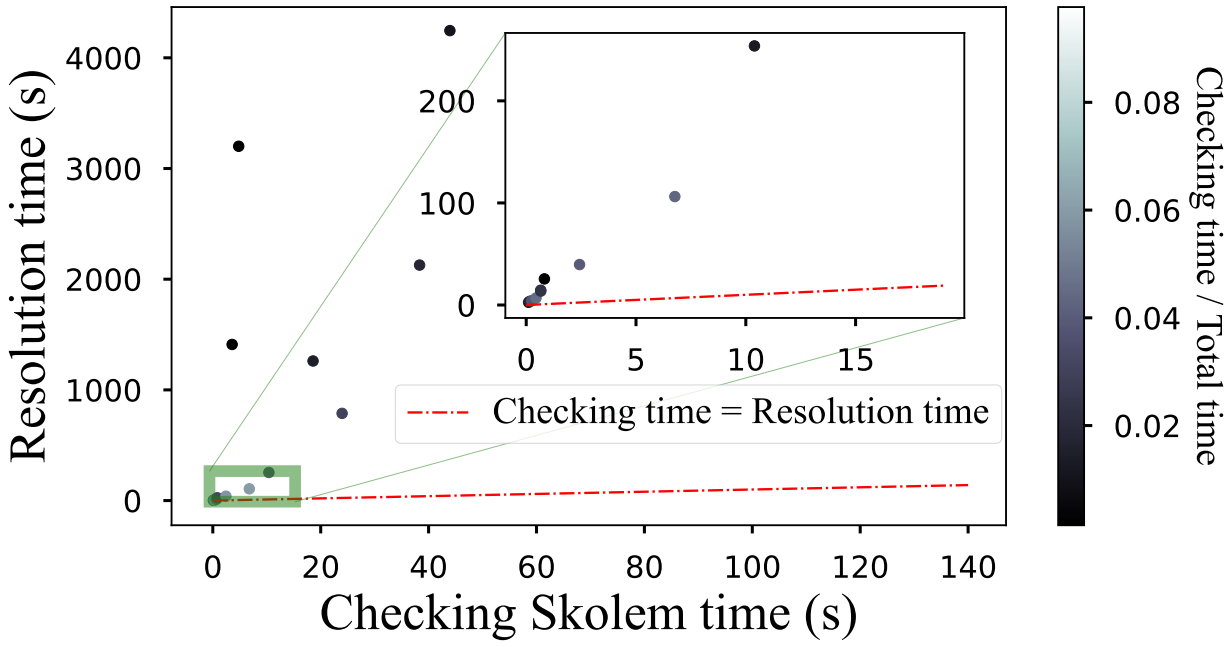

Figure 18: Comparison of the verification time for Skolemization against the time required for resolution-based unsatisfiability checking of the CNF formula produced by Skolemization. The results also indicate that resolution proof checking remains the primary bottleneck for Skolemization.

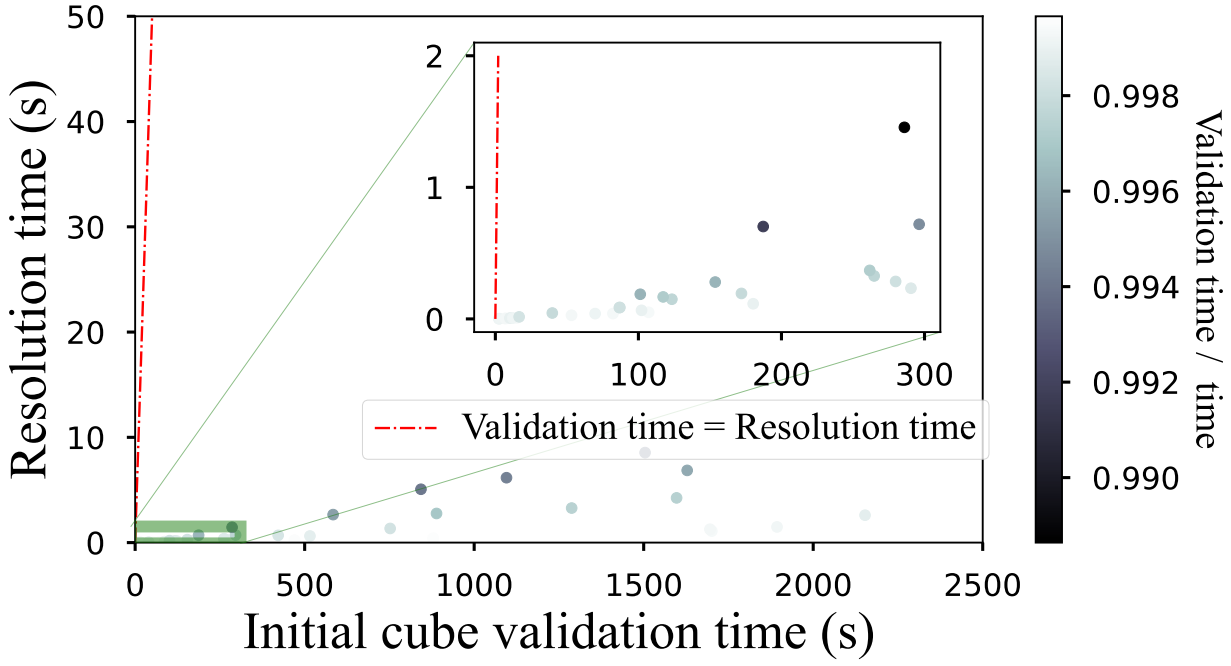

Figure 19: We compare the time cost of initial cube validation with that of cube resolution. The results show that initial cube validation is the primary bottleneck in verifying Q-Res proofs for true QBFs in ZK.

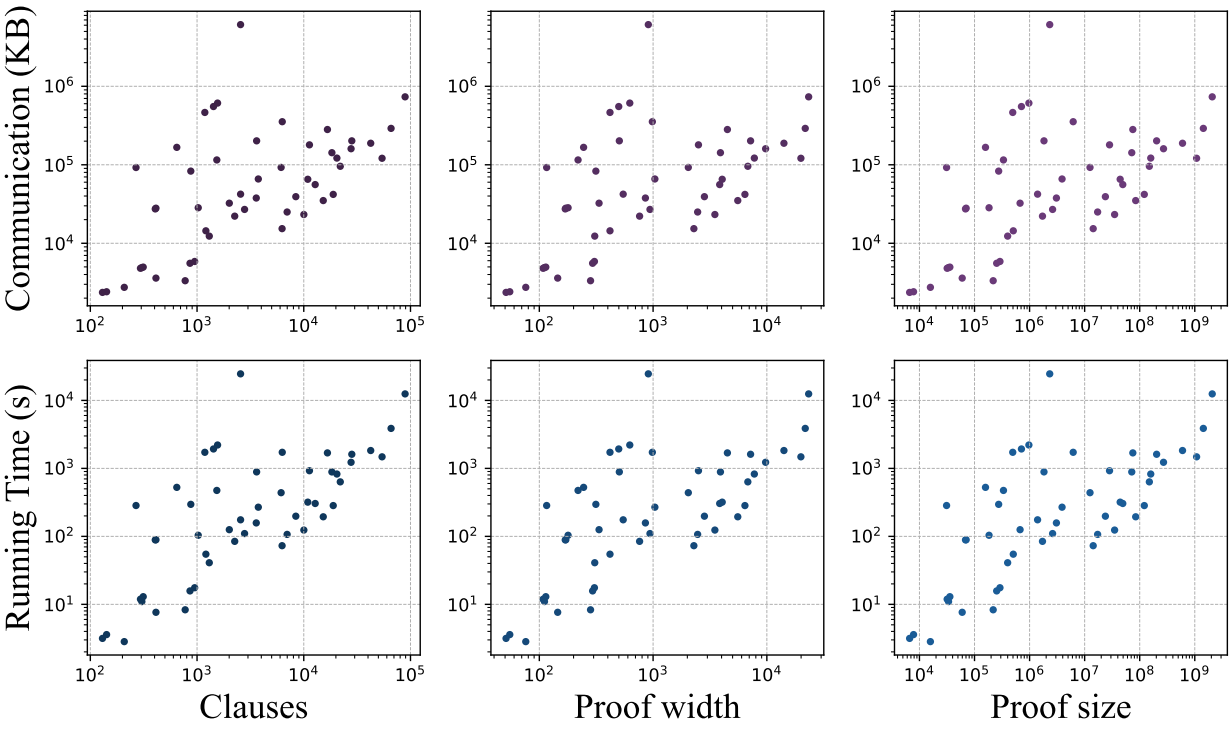

Figure 20: **Communication and time cost of ZKQRES for True QBFs.** Results are sorted by the number of clauses, proof width, and their product (which approximates the total proof size). Unlike ZKQRES for false QBFs, the proportional relationship between the cost and the proof size is less apparent. This is because most of the cost arises from the validity checking of initial cubes.

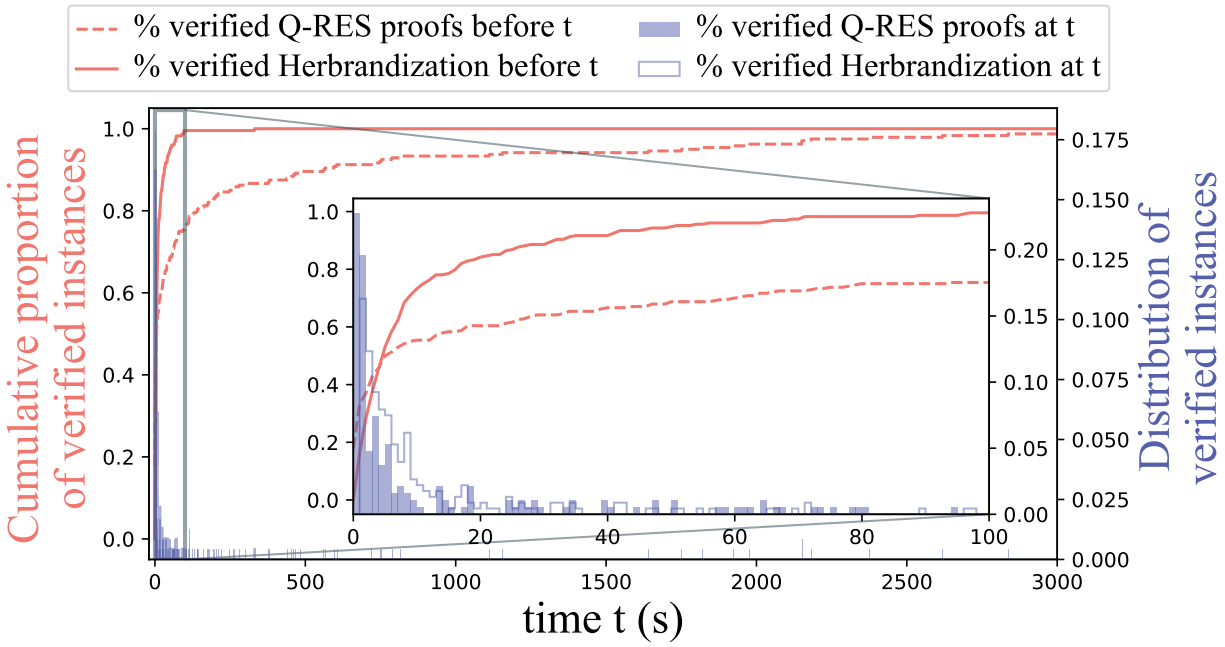

Figure 21: **Evaluation results on BBC benchmarks.**We present the cumulative fraction of BBC QBF instances successfully verified via Q-RES proofs and winning strategies within a given time threshold (left Y-axis), as well as the fraction of instances verified around each time point (right Y-axis)