Proving Circuit Functional Equivalence in Zero Knowledge

Sirui Shen
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
sirui.shen@cwi.nl

Zunchen Huang
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
zunchen.huang@cwi.nl

Chenglu Jin
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
chenglu.jin@cwi.nl

Abstract

The modern integrated circuit (IC) ecosystem is increasingly reliant on third-party intellectual property (3PIP) integration, which introduces security risks, including hardware Trojans and security bugs/vulnerabilities. Addressing the resulting trust deadlock between IP vendors and system integrators without exposing proprietary designs requires novel privacy-preserving verification techniques. However, existing privacy-preserving hardware verification methods are all simulation-based and therefore fail to offer formal guarantees. In this paper, we propose ZK-CEC, the first privacy-preserving framework for hardware formal verification. By combining formal verification and zero-knowledge proof (ZKP), ZK-CEC establishes a foundation for formally verifying IP correctness and security without compromising the confidentiality of the designs.

We observe that existing zero-knowledge protocols for formal verification are designed to prove statements of public formulas. However, in a privacy-preserving verification context where the formula is secret, these protocols cannot prevent a malicious prover from forging the formula, thereby compromising the soundness of the verification. To address these gaps, we first propose a general blueprint for proving the unsatisfiability of a secret design against a public constraint, which is widely applicable to proving properties in software, hardware, and cyber-physical systems. Based on the proposed blueprint, we construct ZK-CEC, which enables a prover to convince the verifier that a secret IP's functionality aligns perfectly with the public specification in zero knowledge, revealing the size of the proof and the gate count of the IP. We implement ZK-CEC and evaluate its performance across various circuits, including arithmetic units and cryptographic components. Experimental results show that ZK-CEC successfully verifies practical designs, such as the AES S-Box, within practical time limits.

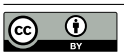
CCS Concepts

• **Hardware** → Equivalence checking; • **Security and privacy** → **Privacy-preserving protocols**.

Keywords

Zero-knowledge Proofs, Hardware formal verification, Combinational equivalence check, Hardware supply chain security

ACM Reference Format:
Sirui Shen, Zunchen Huang, and Chenglu Jin. 2026. Proving Circuit Functional Equivalence in Zero Knowledge. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3832697



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832697

1 Introduction

With the rapid growth of IC technology, chips have become essential components in everything from personal equipment to large-scale infrastructure. To meet the evolving needs and manage increasing design complexity, the system-on-chip (SoC) paradigm has become the dominant business model [23, 42]. In this model, system integrators (buyers) combine intellectual property (IP) cores into a single silicon die. This ecosystem builds a globalized supply chain consisting of IP Vendors and System Integrators.

Despite the efficiency, the increasing reliance on third-party IP (3PIP) introduces significant security risks to the hardware supply chain [37, 40]. A malicious vendor may provide an IP core embedded with hardware Trojans [3, 37], backdoors [47], or intentional logic bugs, leading to supply chain poisoning. Furthermore, malicious vendors might attempt to defraud buyers by providing forged designs. Conversely, if a buyer demands full transparency to verify the design, the vendor faces the risk of IP leakage before acquisition. Establishing a foundation of trust between vendors and buyers without compromising confidentiality has thus become a critical challenge in the hardware supply chain.

Existing research attempts to address the trust issue by privacy-preserving IP verification, which allows a prover to convince a verifier of an IP's correctness without revealing the underlying design. One line of research uses homomorphic encryption (HE) to encrypt test vectors and outsource simulation to third parties [13, 22]. Another research line leverages zero-knowledge proof (ZKP), where a verifier challenges a design encoded as a zero-knowledge virtual machine (zkVM) with test vectors to obtain verifiable simulation results [30, 32].

However, the aforementioned approaches are all simulation-based, which at best increase confidence in the validated designs but cannot completely rule out the presence of malicious logic. In hardware verification, simulation alone is insufficient to capture corner cases, particularly stealthy hardware Trojans that are activated only under rare or carefully crafted conditions [15, 16, 18, 36]. In contrast, formal verification provides a mathematical guarantee of correctness and security [19, 25]. Specifically, combinational equivalence checking (CEC) [12] ensures that a circuit is functionally equivalent to the golden model or specification by constructing a miter circuit and proving its unsatisfiability (UNSAT). In our framework, the functional specification is public and can be a truth table or an unoptimized reference design. While the functionality is public, the value of a highly optimized proprietary implementation still holds because the minimum circuit size problem is believed

to be NP-hard [17] (e.g., researchers are continually improving the implementations of the AES S-Box [29, 38]).

While recent studies [6, 21, 24, 26, 27] have proposed privacy-preserving formal methods for theorem proving and software analysis, their application to hardware is limited. Notably, ZKUNSAT represents the state-of-the-art in proving the unsatisfiability of a public formula in a zero-knowledge framework [27]. However, we observe that ZKUNSAT cannot be directly applied to secret formula verification because it requires the formula statement to be public. Simply treating the formula as a hidden input would compromise the soundness of the protocol, allowing a malicious prover to prove properties about a forged formula.

To bridge this gap, we first propose a blueprint for privacy-preserving formal verification based on ZKUNSAT. Our approach involves partitioning a formula into the secret and the public parts. By checking constraints on the two parts, our blueprint ensures that the protocol achieves soundness against a malicious prover. This blueprint provides a generalized solution for various tasks requiring privacy-preserving formal verification.

Building on the blueprint, we present **the first zero-knowledge protocol for combinational equivalence checking**. By proving the functional equivalence between a secret IP and a public specification, our protocol guarantees functional correctness for the secret IP. We design four sub-protocols corresponding to the constraints in our blueprint. Based on these building blocks, we propose $\Pi_{\text{ZK-CEC}}$ and prove that it satisfies completeness, soundness, and zero-knowledge. Finally, we implement our protocol in the EMP-toolkit and evaluate its performance on circuits of different scales, and demonstrate its practical utility. We also provide an optimization with leakage analysis to improve the efficiency of our protocol, which speeds up the proving time by up to 2.88×.

Our contributions.

- We propose a generalized blueprint for privacy-preserving formal verification. By partitioning formulas into secret and public components and enforcing cross-constraints, we enable formal proofs over hidden logic while maintaining protocol soundness.
- Leveraging the proposed blueprint, we introduce the first zero-knowledge protocol for combinational equivalence checking, which allows vendors to prove that their secret IP is functionally equivalent to a public specification, providing a formal guarantee against logic-level Trojans and design errors.
- We implement our protocol and evaluate it under a set of common combinational circuits. Our evaluation provides a detailed analysis of the communication and computation overhead, with a detailed performance breakdown across different stages of the protocol to identify primary bottlenecks.
- We propose an optimization that merges specific steps within the proof to reduce the time overhead. We present a lower bound analysis showing that additional leakage introduced by the optimization does not enable formula reconstruction by the verifier. By implementing this optimization, we achieve up to a 2.88× speedup in proving time.

Paper organization. The paper is organized as follows. Sec. 2 establishes the problem setting and the threat model. Sec. 3 reviews the preliminaries, including Boolean logic resolution and VOLE-based zero-knowledge proofs. We propose our general blueprint for privacy-preserving property checking in Sec. 4. Sec. 5 applies the blueprint to combinational equivalence checking, detailing the sub-protocols and the main protocol Π_{ZK-CEC}. Sec. 6 presents our implementation and evaluates the framework's performance across various circuit benchmarks. Finally, Sec. 8 discusses related work, and Sec. 9 concludes the paper. Additional proofs and examples are provided in the full version [43].

2 Problem Setting and Threat Model

Problem setting. In our problem setting, we consider a hardware IP vendor as the prover, and a prospective buyer as the verifier. Let C_{spec} denote a public specification of a circuit. The prover holds a circuit implementation C_{impl} and wants to convince the verifier that C_{impl} is functionally correct or free of logic-level Trojan. The vendor commits C_{impl} to the verifier and asserts its functional equivalence with C_{spec}. After the two parties run the zero-knowledge protocol, the verifier is convinced that $C_{\text{impl}} \equiv C_{\text{spec}}$. Finally, the prover opens the commitment, and the verifier pays the money via a fair exchange protocol.

Threat model. We consider a malicious adversarial model where both the prover and the verifier may deviate from the protocol to gain an advantage. A malicious vendor may attempt to defraud the buyer by forging proof for an incorrect design. Conversely, a malicious buyer may attempt to learn information about the IP design before completing the trade.

Goal. Our protocol aims to achieve completeness, soundness, and zero-knowledge.

- **Completeness.** If the prover and the verifier run the protocol honestly on two equivalent designs C_{impl} and C_{spec}, the two parties must accept.
- **Soundness.** If C_{impl} is not functionally equivalent to C_{spec}, the probability that a verifier accepts a proof from a malicious prover is $\text{negl}(\lambda)$, where λ denotes the security parameter. This guarantees that no prover can forge a valid proof for an incorrect design, except with a negligible probability.
- **Zero-knowledge.** The verifier learns that $C_{\text{impl}} \equiv C_{\text{spec}}$ but nothing else about C_{impl} except the publicly defined leakage below.

Leakage. ZK-CEC discloses the quantity and maximum width of the used clauses, as well as the total number of literals in the proof. The clause size encompasses both the number of resolution steps and the input clauses. Revealing the former is aligned with [27], which prevents the verifier from inferring information about the proof. The latter implies the gate-count of the circuit and is permissible to reveal in our scenario. It allows the prover to demonstrate the design's advantage in terms of gate-count, while the verifier still cannot reconstruct the circuit.

3 Preliminaries

3.1 Boolean Logic and Resolution

In this work, we model hardware circuits using Boolean functions in propositional logic. Let $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$ be a finite set of Boolean variables. We define the set of literals as $\mathcal{L} = \mathcal{V} \times \{0, 1\}$. Specifically, a literal $l = (x, b) \in \mathcal{L}$ is positive if $b = 1$ and negative if $b = 0$, conventionally denoted as x and $\neg x$, respectively.

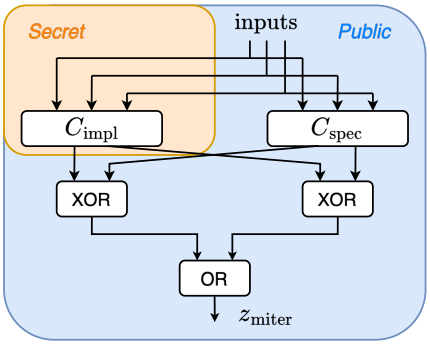


Figure 1: A miter circuit constructed by C_{spec} and C_{impl}.

A clause c is a finite set of literals, logically representing the disjunction (OR) of these literals. For example, $c = \{x_1, \neg x_2\}$ corresponds to the logical expression $(x_1 \vee \neg x_2)$. A clause containing no literals is called the empty clause, denoted as $\bot$.

A Conjunctive Normal Form (CNF) formula Φ is defined as a finite collection of clauses, $\Phi = \{c_1, \ldots, c_m\}$, representing the logical conjunction (AND) of these clauses $\Phi = c_1 \wedge \cdots \wedge c_m$.

A Boolean assignment ω is a mapping that assigns a Boolean value to each variable, which defines an assigned literal set. An assignment ω satisfies a clause c (denoted as $\omega \models c$) if and only if there exists at least one literal in c that evaluates to 1 under ω.

The Boolean Satisfiability (SAT) problem asks whether there exists an assignment ω such that all clauses in Φ are satisfied simultaneously. If such an assignment exists, Φ is satisfiable; otherwise, Φ is unsatisfiable. In our work, the satisfiability of a formula is certified by providing a satisfying assignment ω.

To certify unsatisfiability, we use the resolution proof system [39] to avoid enumerating of all possible assignments. Resolution is a sound and complete inference rule for propositional logic in CNF [7]. Given two clauses $c_1 = (x \vee \alpha)$ and $c_2 = (\neg x \vee \beta)$, where x is the pivot variable and α, β are disjunctions of literals, the resolution rule allows the derivation of a new clause called the resolvent: $\text{Res}(c_1, c_2) = (\alpha \vee \beta)$. The resolvent is logically implied by its antecedents, i.e., any assignment ω that satisfies both c_1 and c_2 also satisfies the resolvent $(\alpha \vee \beta)$. Therefore, adding the resolvent to the formula preserves its satisfiability.

A refutation proof of UNSAT for a CNF formula Φ is a sequence of clauses $D_1, \ldots, D_k$, where each D_i is either a clause from the original formula Φ or a resolvent derived from two previous clauses in the sequence. The proof is successfully concluded when the sequence produces the empty clause $\bot$, which shows a logical contradiction, thereby proving that Φ is UNSAT. The length of a resolution proof corresponds to the total number of intermediate resolvents generated, which is equivalent to the number of resolution derivation steps performed to reach $\bot$.

3.2 Combinational Equivalence Checking

Combinational Equivalence Checking (CEC) involves determining whether two combinational circuits, typically a specification C_{spec} and an implementation C_{impl}, have identical functional behavior. We formally treat these designs as Boolean functions mapping an input vector $X = (x_1, \ldots, x_n)$ to an output vector $Y = (y_1, \ldots, y_m)$, where n and m are the number of inputs and outputs, respectively. The two designs are considered logically equivalent if and only if for every possible input vector X, the output vectors are identical.

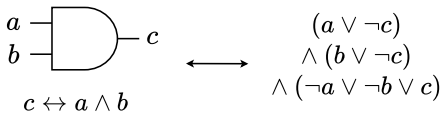


Figure 2: An AND gate and its corresponding CNF formula.

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

Commitment: On receiving (Commit, x) from the prover, where $x \in \mathbb{F}$, store x and send $[x]$ to each party.
Open: On receiving (Open, $[x], y$) from both parties, where $x, y \in \mathbb{F}$ and $[x]$ is a commitment of x. If the x from the two parties does not match, or $x \neq y$, the functionality aborts.
Circuit relation: On receiving (Relation, C, $[x_0], \ldots, [x_{n-1}]$) from both parties, where $x_i \in \mathbb{F}$ and $C \in \mathbb{F}^n \to \mathbb{F}^m$, compute $y_1, \ldots, y_m := C(x_0, \ldots, x_{n-1})$ and send $\{[y_1], \ldots, [y_m]\}$ to both parties.
Equal: On receving (Equal, $[x_0]$, $[x_1]$) from both parties, if $x_0 \neq x_1$ the functionality aborts.

Figure 3: Functionality for ZKPs of elements in $\mathbb{F}$.

To verify this property, we can construct a miter circuit C_{miter} as in Fig. 1 by tying the inputs of C_{spec} and C_{impl} together and comparing their corresponding outputs using XOR gates. All pairwise difference signals are aggregated using a multi-input OR gate to produce a single output $z_{\text{miter}} = \bigvee_{i=1}^{m} (y_i^{\text{spec}} \oplus y_i^{\text{impl}})$. The signal z_{miter} evaluates to 1 if and only if the outputs of the two designs differ for some input. C_{miter} contains three parts: C_{spec}, C_{impl}, and the output gates C_{IO}.

The Tseitin transformation [46] takes as input an arbitrary combinatorial circuit and produces an equi-satisfiable Boolean formula in CNF. Fig. 2 shows an example of a single AND gate and its corresponding CNF formula. Let l_{out} be the literal representing the assignment $z_{\text{miter}} = \text{true}$. We translate C_{miter} into a CNF formula and constrain the formula by asserting that z_{miter} must be true: $\Phi_{\text{miter}} = \Phi_{\text{spec}} \wedge \Phi_{\text{impl}} \wedge \Phi_{\text{IO}} \wedge l_{\text{out}}$. With this approach, the problem of whether $C_{\text{spec}} \equiv C_{\text{impl}}$ can be decided by checking the unsatisfiability of Φ_{miter}. If Φ_{miter} is unsatisfiable, it implies that no input can make the outputs differ. In this case, a resolution proof of unsatisfiability in Sec. 3.1 serves as the formal certificate that C_{spec} and C_{impl} are equivalent.

3.3 VOLE-based Zero-knowledge Protocols

Recent advancements [2, 9, 49–51] in zero-knowledge protocols rely on vector oblivious linear evaluation (VOLE) [34] over a finite field $\mathbb{F}$. A VOLE is a two-party primitive involving a prover and a verifier. In this setup, the verifier holds a secret global key $\Delta \in \mathbb{F}$, chosen uniformly at random. For a vector of secrets $\mathbf{u} \in \mathbb{F}^n$ chosen by the prover, the VOLE protocol distributes randomness such that

the prover obtains a vector $\mathbf{q} \in \mathbb{F}^n$ and the verifier obtains a vector of keys $\mathbf{v} \in \mathbb{F}^n$, satisfying the linear relation: $\mathbf{q} = \mathbf{v} + \mathbf{u} \cdot \Delta$.

The VOLE correlation can be viewed as an information-theoretic message authentication code (IT-MAC). For a single value u, the pair (u, q) held by the prover serves as a commitment to u, authenticated by the verifier's key (v, Δ). The binding property ensures that the prover cannot open the commitment to a different value $u' \neq u$ without guessing Δ. The verifier holds only the global key Δ and the local key v. As v is uniformly distributed and independent of u, the verifier learns no information about the secret u.

The VOLE-based commitment scheme supports efficient arithmetic operations due to the linearity of the correlation [45].

(1) **Linear Operations:** Given commitments to x and y, denoted as $[x]$ and $[y]$, the parties can locally compute a commitment to any linear combination $z = \alpha x + \beta y + \gamma$ without interaction.
(2) **Multiplication:** To verify a multiplicative relation $z = x \cdot y$, the protocols usually employ an interactive check. The prover and verifier interact to generate a new VOLE correlation for z and subsequently prove $[z] = [x \cdot y]$.

The VOLE-based zero-knowledge proofs follow the commit-and-prove paradigm. In this scheme, the prover first commits to the secret witness using the VOLE commitment. Then the prover proves to the verifier that the committed values satisfy the specific constraints of the computation. A functionality for zero-knowledge proofs of elements in $\mathbb{F}$ is shown in Fig. 3 to support arithmetization of CEC in zero-knowledge. Note that, in the paper, we treat the VOLE-based ZK backend as ideal functionalities, which is aligned with the existing work [26, 27, 45].

3.4 Zero-knowledge Proof of Univariate Polynomials

Our work arithmetizes the Boolean functions in univariate polynomials. In this paper, we present the ideal functionality of zero-knowledge proofs as $\mathcal{F}_{\text{ZK-Poly}}$ in Fig. 4 based on $\mathcal{F}_{\text{ZK}}$. **Commit** is realized by committing every coefficient in $P(x)$ using $\mathcal{F}_{\text{ZK}}$. Evaluation on *chal* is implemented using only linear operations of the VOLE commitment, which requires no interaction between the two parties.

In our protocol, we require verifying that the product of two sets of polynomials is identical. However, since $\mathcal{F}_{\text{ZK-Poly}}$ does not natively support polynomial multiplication, we cannot simply compute the product polynomial within the zero-knowledge domain and use **Equal** in $\mathcal{F}_{\text{ZK}}$. In $\mathcal{F}_{\text{ZK-Poly}}$, the parties can prove the identity of polynomials by applying the Schwartz-Zippel lemma. The lemma states that two non-zero distinct polynomials will evaluate to different values at a random point with overwhelming probability. To perform the check, the verifier generates a random challenge *chal*. The parties then evaluate the polynomials at *chal*. By computing the product of these evaluations for each set and comparing the results, they can verify with high probability that the product of the two sets of polynomials is identical.

For simplicity, in this paper, $[\Phi]$ and $[c]$ denote the commitments to all coefficients of the respective polynomial representations for formulas and clauses.

Functionality $\mathcal{F}_{\text{ZK-Poly}}$

Commit: On receiving (Commit, $P(x)$) from the prover, where $P(x) \in \mathbb{F}[x]$, store $P(x)$ and send $[P(x)]$ to each party.
Open: On receiving (Open, $[P(x)], Q(x)$) from both parties, where $P(x), Q(x) \in \mathbb{F}[x]$ and $[P(x)]$ is a commitment of $P(x)$. If the $Q(x)$ from the two parties does not match, or $P(x) \neq Q(x)$, the functionality aborts.
Evaluation: On receiving (Eval, $[P(x)]$, *chal*) from both parties, where $P(x) \in \mathbb{F}[x]$ and *chal* $\in \mathbb{F}$, if *chal* from the two parties does not match, the functionality aborts; otherwise, compute $P(chal) \in \mathbb{F}$ and send $[P(chal)]$ to both parties.
Product-of-polynomials identity: On receiving (PoPIdt, $\{[P_i(x)]\}_{i\in n}, \{[Q_j(x)]\}_{j\in m}$) from both parties, if $\prod_i \{[P_i(x)]\} \neq \prod_j \{[Q_j(x)]\}$, the functionality aborts.

Figure 4: Functionality for ZKPs of univariate polynomials.

3.5 ZKUNSAT

For the UNSAT proof component in our work, we adopt the protocol for checking resolution proof in [27] (referred to as ZKUNSAT). In the ZKUNSAT framework, the protocol begins with the initialization of a public formula Φ. The prover uses a solver to compute an UNSAT certificate of Φ. The UNSAT certificate is structured as a sequence of resolution derivations, as described in Sec. 3.1. For each derivation in the sequence, the commitment to the involved clauses is operated through the functionality $\mathcal{F}_{clause}$ in Fig. 5.

In ZKUNSAT, all clauses are encoded as univariate polynomials. Accordingly, the **Input** and **Equal** instructions in $\mathcal{F}_{clause}$ call the **Commit** and **PoPIdt** instructions of $\mathcal{F}_{\text{ZK-Poly}}$ to conduct clause commitment and equality verification. The **Res** instruction performs a single resolution step. Let w_0 and w_1 denote two witness clauses, and p denote the pivot. A resolution $\{c_0, c_1\} \vdash c_r$ is transformed into the following constraints via clause weakening.

$$\begin{aligned} w_0 \vee c_0 &= c_r \vee p \\ w_1 \vee c_1 &= c_r \vee \neg p \end{aligned} \tag{1}$$

For the pivots encoded as polynomials, the two parties must perform an additional check to ensure that the corresponding polynomial represents exactly one literal. Furthermore, the literal encoding scheme must constrain the relationship between a literal and its negation. By implementing these constraints, the resolution derivation in the polynomial encoding is proved to be both sound and complete [27], while overcoming the deduplication issue.

The sequence of resolution steps can be modeled as a RAM (random-access memory) program executing only one **Res** instruction. For further optimization, the prover can pre-store all intermediate clauses $[c_r]$ in memory. By validating the result of each derivation step against these pre-stored entries, the RAM program execution is reduced to a ROM (read-only memory) consistency check. The ROM in zero-knowledge is instantiated as the functionality $\mathcal{F}_{\text{ZK-ROM}}$ in Fig. 6. The prover and verifier first send all commitments of the input and intermediate clauses to initialize the ZK-ROM. During a resolution step, the two parties get fresh commitments of fetched clauses in the ZK-ROM and check if they

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

Input: On receiving (Input, $l_0, \cdots, l_{k-1}, w$) from the prover and (Input, w) from the verifier where $l_i \in \mathcal{L}$, the functionality checks that $k \leq w$ and abort if it does not hold. Otherwise, store $c = l_0 \vee \cdots \vee l_{k-1}$, and send $[c]$ to each party.
Equal: On receiving (Equal, $[c_0]$, $[c_1]$) from the two parties, check if $c_0 = c_1$; if not, the functionality aborts.
Res: On receiving (Res, $[c_0]$, $[c_1]$, $[c_r]$) from the two parties, check if $\{c_0, c_1\} \vdash c_r$; if not the functionality aborts.
IsFalse: On receiving (IsFalse, $[c]$) from the two parties, check if $c = \bot$; if not, the functionality aborts.

Figure 5: Functionality for clause operations in ZK.

Functionality $\mathcal{F}_{\text{ZK-ROM}}$

Initialization: On receiving (Init, N, $[m_0]$, ..., $[m_{N-1}]$) from the prover and the verifier, where $m_i \in \mathbb{F}$, store the $\{m_i\}$ and set $f :=$ honest and ignore subsequent initialization calls.
Read: On receiving (Read, i, d, t) from the prover, and (Read, t) from the verifier, where $d \in \mathbb{F}$ and $i, t \in \mathbb{N}$, send $[d]$ to each party. If $d \neq m_i$ or t from both parties do not match or $i \geq t$, then set $f :=$ cheating.
Check: Upon receiving (Check) from the verifier, do: If the prover sends (cheat) then send cheating to the verifier. If the prover sends (continue), then send f to the verifier.

Figure 6: Functionality for read-only memory in ZK.

comply with the resolution constraints. After the ROM being accessed by t_{th} times, the two parties use **Check** instruction to check if those fetched commitments are consistent with the pre-stored commitments by the permutation check proposed in [11].

Finally, the two parties verify whether the last derived clause is the empty clause $\bot$ by calling **IsFalse** in $\mathcal{F}_{\text{Clause}}$. According to Theorem 4.7 in [27], the ZKUNSAT protocol is a zero-knowledge proof of knowledge of refutation proof. The Verifier is thus convinced of the unsatisfiability of the public formula (i.e., Φ is unsatisfiable) with only knowing the existence of a valid refutation proof.

4 A Blueprint for Privacy-Preserving Property Checking

4.1 A Dilemma in Applying ZKUNSAT

While ZKUNSAT suggests "proving the safety of secret programs" as a potential application, we identify two issues when deploying this protocol in practical property verification.

As the original ZKUNSAT protocol treats Φ as public, the primary utility of zero-knowledge vanishes. Since UNSAT belongs to co-NP, a verifier could simply execute a state-of-the-art SAT solver to check the formula directly. We do notice that under a specific problem construction in [27], checking refutation proof in ZK can be faster than SAT solving. However, in practical scenarios, the computational overhead of verifying a refutation proof in zero-knowledge does not show enough advantage in efficiency, leaving the protocol not well-motivated in public settings where secrecy is not required.

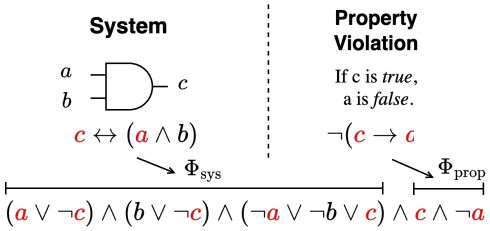


Figure 7: An AND-gate example of secret formula property checking.

If the protocol is modified to treat Φ as a private input, which is consistent with the claimed application of secret program safety, it faces a critical soundness vulnerability. Because the verifier has no knowledge of the problem statement, it cannot ensure that Φ faithfully represents the semantics of the program being verified. For instance, a malicious prover could trivially append contradictory clauses (e.g., $(x) \wedge (\neg x)$) to a satisfiable formula. While the prover can still demonstrate "knowledge of a refutation proof", the proof is vacuous as it no longer correlates with the property of the original secret program.

Consequently, ZKUNSAT is trapped in a dilemma: in the original public formula setting, it is challenged by direct SAT solving; in a potential private formula setting, it loses its soundness due to the absence of a semantic correspondence check. To address this issue, our proposed protocol introduces a partially secret formula setting, which bridges the construction of ZKUNSAT and practical verification problems. Furthermore, the follow-up works [20, 26] in this research line did not explicitly discuss the secrecy of the formula, and the dilemma still exists.

4.2 A Blueprint for Secret Formula Property Checking

To resolve the aforementioned dilemma, we propose a refined setting for secret formula property checking. In this framework, the formula Φ_{sys} representing the prover's system remains secret, while the property formula Φ_{prop} to be verified is public. The verification goal is to demonstrate that the conjunction of the private system model and the public property violation condition is unsatisfiable. Fig. 7 gives a toy example of verifying a property of an AND gate, where the variables colored in red are shared by both formulas.

We formally define the secret formula property checking problem. Let Φ_{sys} be a secret CNF formula that represents the system in question, and Φ_{prop} be a public property CNF formula. The formulas are defined over the variable sets $\mathcal{V}_{\text{sys}}$ and $\mathcal{V}_{\text{prop}}$, respectively. The goal is to provide a zero-knowledge proof that the conjunction of the secret system and the public property is unsatisfiable.

$$\Phi_{\text{sys}} \wedge \Phi_{\text{prop}} \models \bot \tag{2}$$

Table 1: Comparison between ZKUNSAT and Our Proposed Blueprint

Feature	ZKUNSAT (Original)	ZKUNSAT (Modified)	Our Blueprint
Formula Φ	Entirely Public	Entirely Secret	Private System + Public Property
Utility	Low	Unknown	**High**
Soundness in property checking	Yes	No	**Yes**
Use case	Proof of co-NP statement	Secret program verification	Secret HW and SW verification

We also define the variable set $\mathcal{V}_{\text{IO}} = \mathcal{V}_{\text{sys}} \cap \mathcal{V}_{\text{prop}}$, representing the shared interface between the system and the property. The rest part in $\mathcal{V}_{\text{sys}}$ is defined as $\mathcal{V}_{\text{sec}}$, where $\mathcal{V}_{\text{IO}} \cup \mathcal{V}_{\text{sec}} = \mathcal{V}_{\text{sys}}$.

To ensure the soundness of the protocol against a malicious prover, the verifier must check the following properties:

(1) **Property correctness:** Φ_{prop} correctly describes the property, which also implies Φ_{prop} is satisfiable.
(2) **System satisfiability:** Φ_{sys} is satisfiable.
(3) **Variable isolation** $\mathcal{V}_{\text{sec}} \cap \mathcal{V}_{\text{prop}} = \emptyset$.

The property correctness check ensures the validity of the statement, verifying that it corresponds to the security or functional requirements. The system satisfiability check ensures that the prover's secret does not contain any contradiction. The variable isolation ensures that Φ_{sys} and Φ_{prop} only contradict through the explicitly defined interface $\mathcal{V}_{\text{IO}}$. Lemma 4.1 states that when the conjunction of two satisfiable formulas is unsatisfiable, the projections of their satisfying assignment sets on the shared variables are disjoint. A proof of Lemma 4.1 is available in Appendix A.1.

Lemma 4.1. *Let ϕ_a and ϕ_b be two satisfiable formulas with variable sets $\mathcal{V}_a$ and $\mathcal{V}_b$, respectively. We define $\mathcal{V}'_a = \mathcal{V}_a \setminus \mathcal{V}_b$ and $\mathcal{V}'_b = \mathcal{V}_b \setminus \mathcal{V}_a$. If $\phi_a \wedge \phi_b$ is unsatisfiable, then the sets of assignments on $\mathcal{V}_a \cap \mathcal{V}_b$ are disjoint:*

$$\{\omega \mid \omega \models \exists \mathcal{V}'_a \phi_a\} \cap \{\omega \mid \omega \models \exists \mathcal{V}'_b \phi_b\} = \emptyset$$

As established by Lemma 4.1, properties (2) and (3) ensure that the contradiction proved in our protocol is rooted in the conflict between the two satisfiable solution spaces. Specifically, Φ_{sys} and Φ_{prop} project constraints onto $\mathcal{V}_{IO}$, defining two distinct sets of valid partial assignments ω_{sys} and ω_{prop}. The unsatisfiability of the conjunction $(\Phi_{\text{sys}} \wedge \Phi_{\text{prop}})$ is a consequence of ω_{sys} and ω_{prop} being disjoint. In other words, the refutation proof serves as a formal certificate that no behavior of the secret system exposed via $\mathcal{V}_{\text{IO}}$ can satisfy the requirements of the public property.

By conducting the property checks in zero-knowledge and composing them with ZKUNSAT, we establish a blueprint for privacy-preserving secret formula property checking. Our blueprint transforms ZKUNSAT from a theoretical primitive into a practical tool. Table 1 summarizes the differences between the ZKUNSAT and our proposed blueprint regarding their utility. Our blueprint allows a prover to convince a verifier that its black-box system satisfies a public property without revealing the underlying implementation details. Our proposed blueprint provides a framework for privacy-preserving verification, adaptable to hardware, software, or general model checking. To demonstrate its practical utility, we apply our framework to combinational equivalence checking in our paper.

5 Arithmetization and Protocol

5.1 Overview of Our Protocol

ZK-CEC achieves correctness, soundness, and zero-knowledge, meaning both the prover and the verifier may be malicious. The goal of ZK-CEC is to check the equivalence of two circuits C_{spec} and C_{impl} by proving UNSAT of the formula Φ_{miter} representing the miter circuit. However, as mentioned in Sec. 3.5, simply applying ZKUNSAT [27] for proving UNSAT of a secret formula lacks soundness. Hence, we achieve soundness in ZK-CEC by utilizing the structure of the CEC problem itself.

The formula $\Phi_{\text{miter}} = \Phi_{\text{spec}} \wedge \Phi_{\text{impl}} \wedge \Phi_{\text{IO}} \wedge l_{\text{out}}$ can be divided into a public formula $\Phi_{\text{pub}} = \Phi_{\text{spec}} \wedge \Phi_{\text{IO}} \wedge l_{\text{out}}$ and a secret formula $\Phi_{\text{sec}} = \Phi_{\text{impl}}$. Aligning with the proposed blueprint in Sec. 4.2, we ensure the correctness and soundness of proving UNSAT of Φ_{miter} by checking the following properties:

(1) **P1. Correctness and Satisfiability of Φ_{pub}.** The public clauses must follow the specification and the structure of a miter circuit, and they must be satisfiable.
(2) **P2. Unsatisfiability of Φ_{miter}.** The miter circuit must be unsatisfiable.
(3) **P3. Satisfiability of Φ_{sec}.** The formula corresponding to the secret circuit must be satisfiable, which means it cannot derive any local contraction.
(4) **P4. Non-intersection of Φ_{pub} and Φ_{sec}.** The literals and their negations in the public clauses and private clauses must not have any intersection except the input and output.

P1 is ensured by opening the commitments, which is discussed in Sec.5.3. The other properties are checked under arithmetization under the encoding scheme described in Sec. 5.2 that processes the public and the secret literals separately. For P2, Sec. 5.4 adopts the solution in [27] that checks the UNSAT as a RAM program. Sec. 5.5 discusses our approach to proving P3 using the same commitments in P2. We present another building block to prove P4 in Sec. 5.6 to ensure the separation of $[\Phi_{\text{pub}}]$ and $[\Phi_{\text{sec}}]$. Finally, Sec. 5.7 puts everything together and proposes the complete protocol for CEC in zero-knowledge.

In our framework, the arithmetization of elements and operations is performed over the finite field $\mathbb{F}_{2^k}$. We refer to this field simply as $\mathbb{F}$ in the remainder of this paper.

5.2 Encoding of Formula

To put all the proof checking into a zero-knowledge protocol, we need to transform the logic deduction into an arithmetic system. A CNF formula can be represented as a set of clauses, and a clause can be represented as a set of literals. The propositional calculus can be mapped into set operations. In our protocol, we encode clauses

into univariate polynomials as in [27, 35], where the literals are represented as different roots of polynomials. In this method, we convert the SAT problem to the existence of specific roots, and the set operations can be checked by polynomial operations. The encoding scheme in our framework adopts the approach in [27].

Let c denote a clause of n literals in a CNF formula. $\mathcal{L}$ denotes the set of all literals. $l_i \in \mathcal{L}$ denotes the i-th literal in the clause c. We define an encoding $\text{Poly}_\phi : \text{clause} \to \mathbb{F}[x]$:

$$\text{Poly}_\phi[l_0 \vee l_1 \vee \cdots \vee l_n] = (x - \phi(l_0))(x - \phi(l_1)) \cdots (x - \phi(l_n))$$

that represents a clause as a univariate polynomial over $\mathbb{F}$. Each literal is encoded to an element in $\mathbb{F}$ under ϕ and embedded as a root of the polynomial. We give an example of converting a circuit and a property into encoded polynomials and committing those polynomials in Appendix B. For brevity, we denote the polynomial corresponding to the encoding of clause c as $\text{Poly}[c]$ in the remainder of this paper.

Through this encoding, we transform a set of literals into a set of roots. Consequently, the following properties can be verified via polynomial operations:

(1) **Membership test:** A literal belongs to a clause if and only if its corresponding value is a root of the encoding polynomial.

$$l \in c \iff \text{Poly}[c](\phi(l)) = 0 \tag{3}$$

(2) **Entailment check:** A clause c is a sub-clause of the clause c' if and only if the polynomial corresponding to c divides the polynomial corresponding to c'.

$$c \to c' \iff \text{Poly}[c] \big| \text{Poly}[c'] \tag{4}$$

(3) **Unique disjunction:** The conjunction of two disjoint clauses corresponds to the product of their encoded polynomials.

$$\forall c_1, c_2,\ c_1 \wedge c_2 = \emptyset \implies \text{Poly}[c_1] \cdot \text{Poly}[c_2] = \text{Poly}[c_1 \vee c_2] \tag{5}$$

Note that, in this polynomial encoding, deduplication of the union of two clauses is represented as the removal of the repeated roots, which introduces huge computational overhead in the ZKP backend. Therefore, we only check the disjunction of two disjoint clauses in our protocol.

Following the clause encoding, we define the literal encoding $\phi : \text{literal} \to \mathbb{F}$. Let w_{lit} denote the maximum bitwidth of an encoded literal. We define two constants $const_{\text{pub}} = (1 \ll (w_{\text{lit}} - 1))_{\mathbb{F}}$ and $const_{\text{lit}} = (1 \ll (w_{\text{lit}} - 2))_{\mathbb{F}}$ that point to the two most significant bits (MSBs) of an encoded literal.

First, we establish the encoding constraint regarding a literal and its negation:

$$\phi(x) + \phi(\neg x) = const_{\text{lit}} \tag{6}$$

This constraint enables us to check the relationship between complementary literals using addition. Additionally, under Eq. 6, we need to define only the positive literals, since their negations are naturally determined.

Based on the constraint of Eq. 6, we design a new adaptive literal encoding that maps secret and public literals to field elements separately. In our protocol, the formula consists of Φ_{pub} and Φ_{sec}. Let $\mathcal{L}_{\text{pub}}$ denote the set of literals corresponding to the variables in Φ_{pub}, and let $\mathcal{L}_{\text{impl}}$ denote the set of literals corresponding to the variables in Φ_{impl}. We define the set of secret literals $\mathcal{L}_{\text{sec}} = \mathcal{L}_{\text{impl}} \setminus \mathcal{L}_{\text{pub}}$. To map literals into field elements, we define the mapping $\text{idx}(\cdot) : \mathcal{L}_{\text{pos}} \to \{1, \ldots, n\}$ that assigns a unique index to every positive literal, where n is the number of positive literals. Let $H_k(\cdot)$ be a collision-resistant keyed hash function with key k. The output bit-length of $H_k(\cdot)$ is $w_{\text{lit}} - 1$ and smaller than that of $\mathbb{F}$; therefore, we can directly map the output bitstring into $\mathbb{F}$. We define ϕ as a deterministic mapping from positive literals to $\mathbb{F}$ as follows:

- For $l \in \mathcal{L}_{\text{pub}}$, $\phi(l) = \text{idx}(l) + const_{\text{pub}}$.
- For $l \in \mathcal{L}_{\text{sec}}$, $\phi(l) = H_k(\text{idx}(l))$.

where the field elements of negative literals are defined by Eq. 6.

For public literals, we directly use the index as the encoded element, enabling both parties to compute it locally. We require the maximum index to be strictly less than $2^{w_{\text{lit}}-2}$, which ensures that the bit-flip introduced by $const_{\text{lit}}$ in Eq. 6 does not cause a collision between the encoding of the negation $\phi(\neg l)$ and another positive literal $\phi(l')$. We set the MSB to 1 to mark the encoded literal as public, ensuring that public and secret encoded literals cannot collide.

For secret literals, the verifier should be prevented from determining their polarity by observing the encoded values. To achieve this, we use a collision-resistant keyed hash function $H_k(\cdot)$ for the encoding. The $H_k(\cdot)$ we use can be modeled as a pseudorandom function (PRF). Therefore, this scheme ensures that the polarity of a literal is indistinguishable from a random bit to the verifier, due to the indistinguishability of PRF. Furthermore, the collision resistance of $H_k(\cdot)$ ensures it is computationally infeasible for a malicious prover to find a key where two secret literals have collided encoding, i.e., $\phi(l) = \phi(l')$. The verifier's recomputation of the hash during commitment opening forces the prover to use the specified hash function.

Lemma 5.1. *For any secret variable $x \in \mathcal{V}_{sec}$, and any adversary $\mathcal{A}$, the probability of distinguishing between $\phi(x)$ and $\phi(\neg x)$ is $negl(\lambda)$.*

To prevent the leakage of clause width information in the proof, we ensure that all polynomials have the same length. We identify the maximum degree w among polynomials and pad the high-order coefficients of shorter polynomials with zeros, which ensures that all polynomial commitments do not leak individual clause widths.

5.3 Proving P1

In Fig. 8, we propose the protocol for the verifier to check the commitments of Φ_{pub}. In this protocol, the prover first sends the index assignments for each literal within the arithmetized formula to the verifier. This enables the verifier to locally reconstruct the encoded polynomial. By opening the commitments, the verifier validates the correctness of the components corresponding to the public specification and the miter. Additionally, as a valid specification and miter, Φ_{pub} is naturally satisfiable.

Completeness. The completeness of the protocol Π_{P1} is guaranteed by the injectivity of Poly_ϕ.

Soundness. The soundness of the protocol Π_{P1} is guaranteed by the binding property verified during the opening of the commitments.

5.4 Proving P2

To verify P2, we construct the protocol Π_{P2} based on the main protocol introduced in [27]. Unlike the original ZKUNSAT, Π_{P2} operates

Protocol $\Pi_{\mathbf{P1}}$

Parameters: A finite field $\mathbb{F}$. A clause encoding scheme Poly_ϕ and a literal encoding scheme ϕ.
Input: Both parties have a public specification C_{spec} and hold a commitment $[\Phi_{\text{pub}}]$ of all public clauses to check.
Protocol:

(1) The verifier constructs the IO C_{IO} of the miter upon C_{spec}.
(2) The verifier transform C_{spec} and C_{IO} to Φ'_{pub} using Tseytin transformation.
(3) The prover sends the index assignment $\text{idx}(\cdot)$ of all literals in Φ_{pub} to the verifier. The two parties compute $\mathcal{L}_{\text{pub}}$.
(4) For i-th clause c_i in Φ'_{pub}, the verifier checks the correctness by the following:
   (a) The verifier computes $P(x) = \text{Poly}_\phi[c_i]$ using $\text{idx}(\cdot)$.
   (b) Both parties send (Open, $[c_i], P(x)$) to $\mathcal{F}_{\text{ZK-Poly}}$. If the functionality aborts, the protocol aborts.

Figure 8: Protocol for proving P1.

Protocol $\Pi_{\mathbf{P2}}$

Input: The two parties have a committed formula $[\Phi] = [c_0] \wedge \cdots \wedge [c_{|\Phi|-1}]$ corresponding to the prover's secret. The prover has a key k for the hash function H in the literal encoding scheme ϕ and a refutation proof $((k_{l_0}, k_{r_0}, c_{|\Phi|}), \cdots, (k_{l_{R-1}}, k_{r_{R-1}}, c_{|\Phi|+R-1}))$; both parties know the length of the refutation proof R and the maximum width of all clauses $w = \max(c_i)$.
Protocol:

(1) The prover encodes $\{c_i\}_{i\in[|\Phi|,R+|\Phi|-1]}$ with $\text{Poly}_\phi(\cdot)$. The two parties send the Input instruction to $\mathcal{F}_{\text{Clause}}$ and obtain $[c_i]_{i\in[|\Phi|,R+|\Phi|-1]}$.
(2) The two parties send (Init, $R + |\Phi| - 1$, $[c_0], \cdots, [c_{|\Phi|+R-1}]$) to $\mathcal{F}_{\text{ZK-ROM}}$.
(3) For the i-th iteration, the two parties check the refutation proof by doing the following.
   (a) The prover looks up the tuple $(k_{l_i}, k_{r_i}, c_{|\Phi|+i})$ from the refutation proof such that $(c_{k_{l_i}}, c_{k_{r_i}}) \vdash c_{|\Phi|+i}$.
   (b) The prover sends (Read, $k_{l_i}, c_{k_{l_i}}, i$) to $\mathcal{F}_{\text{ZK-ROM}}$; the verifier sends (Read, i) to $\mathcal{F}_{\text{ZK-ROM}}$, from which the two parties obtain $[c_{k_{l_i}}]$. Similarly, the two parties obtain $[c_{k_{r_i}}]$ and $[c_{|\Phi|+i}]$.
   (c) The two parties send (Res, $[c_{k_{l_i}}], [c_{k_{r_i}}], [c_{|\Phi|+i}]$) to $\mathcal{F}_{\text{Clause}}$.
(4) After R iterations, two parties use $\mathcal{F}_{\text{Clause}}$ to check that $[c_{|\Phi|+R-1}]$ equals $\perp$; if $\mathcal{F}_{\text{Clause}}$ aborts, the verifier aborts.
(5) Two parties send (Check) to $\mathcal{F}_{\text{ZK-ROM}}$, if $\mathcal{F}_{\text{ZK-ROM}}$ aborts, the verifier aborts.

Figure 9: Protocol for proving P2.

on a committed formula, which replaces the initial commitment phase. Thus Π_{P2} proves the unsatisfiability of a secret formula. Π_{P2} assumes the refutation proof length R and the maximum clause width w are public. The detailed construction of Π_{P2} is illustrated in Fig. 9.

In Π_{P2}, the prover has a refutation proof in the form of a tuple. There are three items in each element of the tuple, where k_{l_i} and k_{r_i} denote the indices of two antecedents in a resolution step, and $c_{|\Phi|+i}$ denotes the resolvent. All the polynomials corresponding to the resolvents are padded to the same length, hence their commitments $[c_{|\Phi|+i}]$ are also the same in length.

Π_{P2} has three steps: First, the prover commits the resolvents, and the two parties initialize $\mathcal{F}_{\text{ZK-ROM}}$ with the commitments. Next, the two parties fetch clauses from $\mathcal{F}_{\text{ZK-ROM}}$ and verify the resolution steps. Each resolution step has two antecedents and one resolvent, and those clauses are hidden from the verifier as VOLE-based commitments. Note that all these clauses are fresh commitments fetched from the ZK-ROM; thus, the verifier remains unknown to their contents, regardless of whether the original clauses are public or secret. Finally, the two parties check that the last derived clause is empty and execute the permutation check of $\mathcal{F}_{\text{ZK-ROM}}$.

Completeness. The completeness of the protocol Π_{P2} holds because the protocol is deterministic.

Conditional soundness. Note that Π_{P2} does not guarantee the integrity of the input formula. Under the condition that the input formula is valid, the soundness of Π_{P2} is reducible to the soundness of the main protocol in ZKUNSAT.

5.5 Proving P3

To check P3, the prover demonstrates to the verifier that Φ_{sec} is satisfiable without revealing its content. We propose a privacy-preserving protocol to prove SAT over our encoding scheme.

Let Φ_{sec} be a Boolean formula defined over a set of n variables $\mathcal{V}_{\text{impl}} = \{x_1, x_2, \ldots, x_n\}$. $\mathcal{V}_{\text{impl}}$ has two parts: $\mathcal{V}_{\text{io}}$ and $\mathcal{V}_{\text{sec}}$. The variables in $\mathcal{V}_{\text{io}}$ and $\mathcal{V}_{\text{sec}}$ correspond to the input/output wires and the internal wires of C_{impl}, respectively. Since C_{impl} shares inputs with C_{spec} and connects its outputs to C_{IO}, $\mathcal{V}_{\text{IO}}$ is known by both parties, implying $\mathcal{L}_{\text{IO}} \in \mathcal{L}_{\text{pub}}$. The internal wires of C_{impl}, represented by $\mathcal{V}_{\text{sec}}$, corresponds to $\mathcal{L}_{\text{sec}}$. Let $\omega_{\text{sec}} : \mathcal{V}_{\text{sec}} \to \{0,1\}^n$ be a Boolean assignment. The SAT problem of Φ_{sec} is defined:

$$\exists \omega_{\text{sec}},\ \forall c \in \Phi_{\text{sec}},\ \omega_{\text{sec}} \models c \tag{7}$$

We transform the problem that ω_{sec} satisfies a clause c into a membership testing problem as in Eq. 3; therefore, the verifier can verify P3 using polynomial arithmetic. We propose the protocol for proving P3 in Fig. 10.

Completeness. When the two parties are honest, the completeness of the protocol Π_{P3} is guaranteed by the completeness of the membership test in Eq. 3 and the ZK backend. Provided that the assignment is valid, $\text{Poly}[c]$ vanishes at at least one encoded literal index $\phi(l_i)$, which implies that the product of its evaluations over the domain is zero.

Conditional soundness. The soundness of the protocol Π_{P3} is defined as: Assume that $\forall l \in \mathcal{L}, \phi(l) + \phi(\neg l) = const_{\text{lit}}$, if Φ_{sec} is unsatisfiable, the verifier will accept that Φ_{sec} is unsatisfiable with negligible probability $\text{negl}(\lambda)$. A proof sketch is in Appendix A.2.

Protocol $\Pi_{\mathbf{P3}}$

Parameters: A finite field $\mathbb{F}$. A clause encoding scheme Poly_ϕ and a literal encoding scheme ϕ.
Input: Both parties hold a commitment $[\Phi_{\text{sec}}]$ of all secret clauses to be checked. The number of clauses in $[\Phi_{\text{sec}}]$ is n. The prover holds a valid assignment $\omega \models \Phi_{\text{sec}}$. Let $\mathcal{L}$ denote the set of literals corresponding to ω. The prover also knows the number of literals m in $\mathcal{L}$.
Protocol:

(1) The prover sends m to the verifier.
(2) For every $i \in \{0, 1, \cdots, m-1\}$, the prover sends $\phi(l_i)$ to the verifier.
(3) The verifier checks for the presence of complementary literals. For every $i, j \in \{0, 1, \cdots, m-1\}$, if $\phi(l_i) + \phi(l_j) = const_{\text{lit}}$, the protocol aborts.
(4) The prover sends (Commit, 0) to $\mathcal{F}_{\text{ZK}}$. The two parties receive $[0]$ and send (Open, $[0]$, 0) to $\mathcal{F}_{\text{ZK}}$. If the functionality aborts, the protocol aborts.
(5) For every $j \in \{0, 1, \cdots, n-1\}$, the verifier checks the satisfiablity of $c_j \in \Phi_{\text{sec}}$ by the following:
   (a) For every $i \in \{0, 1, \cdots, m-1\}$, both parties send (Eval, $[c_j], l_i$) to $\mathcal{F}_{\text{ZK-Poly}}$ and receive $[\text{Poly}[c_j](l_i)]$.
   (b) Both parties send (PoPIdt, $\{[\text{Poly}[c_j](l_i)]\}_{i \in m}, \{[0]\}$) to $\mathcal{F}_{\text{ZK-Poly}}$. If the functionality aborts, the protocol aborts.

Figure 10: Protocol for proving P3.

5.6 Proving P4

To verify P4, the prover shows the verifier that the set of literals in Φ_{sec} is disjoint from the literals in Φ_{pub}, except for the interface literals $\mathcal{L}_{\text{IO}}$. Based on the membership test defined in Eq. 3, a literal l is not in clause c if and only if $\text{Poly}[c](\phi(l)) \neq 0$. Hence the protocol verifies that for every literal $l \in \mathcal{L}_{\text{pub}} \setminus \mathcal{L}_{\text{IO}}$ and every clause $c \in \Phi_{\text{sec}}$, the polynomial evaluation $\text{Poly}[c](\phi(l))$ yields a non-zero value. We give the detailed protocol to prove P4 in Fig.11.

To prove that $\text{Poly}[c](\phi(l)) \neq 0$ without compromising zero-knowledge, we use a standard non-zero proof technique. The evaluation is performed through the **Eval** in $\mathcal{F}_{\text{ZK-Poly}}$, and the result is a commitment. Rather than opening the commitment, the prover locally calculates $inv = (\text{Poly}[c](\phi(l)))^{-1}$ and provides a commitment $[inv]$. The protocol then verifies that $[\text{Poly}[c](\phi(l))] \cdot [inv] = [1]$. The existence of the multiplicative inverse in $\mathbb{F}$ guarantees that the evaluation is non-zero, while preventing any leakage of Φ_{sec}.

Completeness. When the two parties are honest, the completeness of the protocol Π_{P4} is guaranteed by the completeness of the membership test in Eq. 3 and the ZK backend.

Soundness. The soundness of the protocol Π_{P4} is defined as: if $\exists l \in \mathcal{L}_{\text{pub}} \setminus \mathcal{L}_{\text{IO}}, l \in \Phi_{\text{sec}}$, the verifier will accept Π_{P4} with negligible probability $\text{negl}(\lambda)$. Consider a malicious prover $\mathcal{P}^*$. The protocol assumes that both parties know $\mathcal{L}_{\text{pub}} \setminus \mathcal{L}_{\text{IO}}$. If there exists a literal $l \in \mathcal{L}_{\text{pub}} \setminus \mathcal{L}_{\text{IO}}$ such that $l \in \Phi_{\text{sec}}$, the only way for $\mathcal{P}^*$ to convince the verifier is to cheat in $\mathcal{F}_{\text{ZK-Poly}}$ and $\mathcal{F}_{\text{ZK}}$. However, the soundness of the ZK backend ensures that the verifier will accept the protocol with negligible probability $\text{negl}(\lambda)$.

Protocol $\Pi_{\mathbf{P4}}$

Parameters: A finite field $\mathbb{F}$. A clause encoding scheme Poly_ϕ and a literal encoding scheme ϕ.
Input: Both parties hold a commitment $[\Phi_{\text{sec}}]$ of all secret clauses to be checked. The number of clauses in $[\Phi_{\text{sec}}]$ is n. The two parties know the public literal set $\mathcal{L}_{\text{pub}}$ and the interface literal set $\mathcal{L}_{\text{IO}}$. Let m denote the number of literals in $\mathcal{L}_{\text{pub}} \setminus \mathcal{L}_{\text{IO}}$.
Protocol:

(1) The prover sends (Commit, 1) to $\mathcal{F}_{\text{ZK}}$. The two parties receive $[1]$ and send (Open, $[1]$, 1) to $\mathcal{F}_{\text{ZK}}$. If the functionality aborts, the protocol aborts.
(2) For every $i \in \{0, 1, \cdots, n-1\}$ and every $j \in \{0, 1, \cdots, m-1\}$, the verifier checks $c_i \in \Phi_{\text{sec}}$ with all disjoint public variables:
   (a) The prover evaluates $\text{Poly}[c_i](l_j)$ and computes the multiplicative inverse $inv = \text{Poly}[c_i](l_j)^{-1}$ locally.
   (b) The prover sends (Commit, inv) to $\mathcal{F}_{\text{ZK}}$. The two parties receive $[inv]$.
   (c) Both parties send (Eval, $[c_i], l_j$) to $\mathcal{F}_{\text{ZK-Poly}}$ and receive $[\text{Poly}[c_i](l_j)]$.
   (d) Both parties send (Relation, $y = x_0 \cdot x_1$, $[\text{Poly}[c_i](l_j)]$, $[inv]$) to $\mathcal{F}_{\text{ZK}}$, and receive $[\text{Poly}[c_i](l_j) \cdot inv]$.
   (e) Both parties send (Equal, $[\text{Poly}[c_i](l_j) \cdot inv]$, $[1]$) to $\mathcal{F}_{\text{ZK}}$. If the functionality aborts, the protocol aborts.

Figure 11: Protocol for proving P4.

5.7 Putting Everything Together

We propose our main protocol $\Pi_{\text{ZK-CEC}}$ for zero-knowledge combinational equivalence checking in Fig. 12 by composing Π_{P1}, Π_{P2}, Π_{P3}, and Π_{P4} together. The soundness of the complete protocol relies on the sequential enforcement of the blueprint properties defined in Sec. 4.2. While Π_{P2} conditionally proves the unsatisfiability of the miter, it relies on Π_{P3} and Π_{P4} to guarantee that the underlying secret formula Φ_{sec} is satisfiable and isolated from the public property. This composition proves that the prover committed design $[\Phi_{\text{impl}}]$ is equivalent to the public specification Φ_{spec}.

Theorem 5.1. *$\Pi_{\text{ZK-CEC}}$ is a zero-knowledge proof of combinational equivalence.*

We provide a proof sketch for Theorem 5.1 in Appendix A.3, which argues that the protocol satisfies completeness, soundness, and zero-knowledge.

5.8 Practical Optimization with Extra Leakage

To mitigate the computational overhead associated with large refutation proofs, we introduce a resolvent compression optimization designed to improve the performance of Π_{P2}. Notably, this optimization is also applicable to the original ZKUNSAT framework.

The insight is that only part of the resolvents would be used as antecedents multiple times. Those resolvents are *learned clauses* in the conflict-driven clause learning (CDCL) algorithm [28] that is widely used in modern SAT solvers. Therefore, we merge multiple

Protocol $\Pi_{\text{ZK-CEC}}$

Parameters: A finite field $\mathbb{F}$. A clause encoding scheme $\text{Poly}_\phi[\cdot]$ and a literal encoding scheme $\phi(\cdot)$.
Input: The prover has a secret circuit $\Phi_{\text{sec}} = C_{\text{impl}}$. Both parties know a public specification C_{spec}.
Protocol:

(1) The prover builds the miter circuit, and transform Φ_{sec} from C_{impl} and Φ_{pub} from C_{spec}, C_{IO}, and l_{out}.
(2) The prover generates a secret key k for the hash function H in $\phi(\cdot)$.
(3) The prover encodes Φ_{sec} with $\text{Poly}_\phi[\cdot]$, and sends $(\text{Commit}, \text{Poly}_\phi[\Phi_{\text{sec}}])$ to $\mathcal{F}_{\text{ZK-Poly}}$. The two parties receives $[\Phi_{\text{sec}}]$.
(4) The prover encodes Φ_{pub} with $\text{Poly}_\phi[\cdot]$, and sends $(\text{Commit}, \text{Poly}_\phi[\Phi_{\text{pub}}])$ to $\mathcal{F}_{\text{ZK-Poly}}$. The two parties receives $[\Phi_{\text{pub}}]$.
(5) The two parties run Π_{P1} on input of $[\Phi_{\text{pub}}]$ and C_{spec}, and obtain $\mathcal{L}_{\text{pub}}$.
(6) The prover locally solves $\Phi_{\text{miter}} = \Phi_{\text{sec}} \wedge \Phi_{\text{pub}}$ and obtains a refutation proof. The prover also knows the length of the refutation proof R and the maximum width of all clauses w.
(7) The two parties run Π_{P2} on input of $[\Phi_{\text{miter}}]$, R, and w.
(8) The prover locally solves Φ_{sec} and obtains an assignment ω that satisfies Φ_{sec}.
(9) The two parties run Π_{P3} on input of $[\Phi_{\text{sec}}]$ and ω.
(10) The two parties run Π_{P4} on input of $[\Phi_{\text{sec}}]$ and $\mathcal{L}_{\text{pub}}$.

Figure 12: Protocol for zero-knowledge combinational equivalence checking.

intermediate resolution steps into a single derivation and store only those *learned clauses* in the ZK-ROM, which reduces the overhead of permutation checks. We provide a simplified refutation proof structure example in the full version [43] to give an intuitive understanding.

While the verifier remains unknown about the antecedents and resolvents used in a resolution step, this optimization introduces additional leakage of proof structure, where the verifier can determine the length of each sub-tree. To reconstruct the original formula, the verifier is required to first correctly guess the proof tree structure and then use it to recover the formula.

We demonstrate that formula reconstruction remains sufficiently difficult under the optimization by giving a lower bound on the number of possible proof tree structures in the first step. For the sake of analysis, we assume each sub-tree is used only once and composed in the sequence they are checked, which only loosens the lower bound. For a compressed resolution proof with R' steps, let n_j denote the number of possible antecedent positions at the j-th sub-tree. The number of potential structures for constructing the proof tree from the sub-trees is given by:

$$N > \prod_{i=1}^{R'-1} \left(\sum_{j=i+1}^{R'} n_j \right) \geq (R'-1)! \tag{8}$$

In the case of a sub-tree with a single resolution, the placement of a subsequent sub-tree occupies only one of the two antecedent positions. For sub-trees involving multiple resolutions, we assume that the subsequent sub-tree always occupies a possible position. Consequently, n_j is defined as:

$$n_j = \begin{cases} 1 & \text{if } n_{\text{res}_j} = 1 \\ n_{\text{res}_j} - 1 & \text{if } n_{\text{res}_j} > 1, j \neq R' \\ n_{\text{res}_j} & \text{if } n_{\text{res}_j} > 1, j = R'; \end{cases} \tag{9}$$

where n_{res_j} denotes the number of resolvents within the j-th sub-tree. The term $(n_{\text{res}_j} - 1)$ is because each preceding sub-tree ($j \neq R'$) occupies one possible position. We provide the step-by-step derivation of N for the example in the full version [43]. The loose lower bound N exceeds the factorial of $(R' - 1)$, and R' is large enough in practical design. For example, a 4-bit adder typically yields $R' = 90$. Therefore, the factorial complexity renders the enumeration of potential proof structures sufficiently complex.

5.9 Discussion on Fair Exchange

In the context of the trustworthy 3PIP supply chain, ensuring the soundness of the verification process alone is insufficient; guaranteeing fair exchange between the vendor and the system integrator is equally important. While existing works allow vendors to present valid proofs [30, 32], they lack a binding between the IP being transacted and the proof.

Our protocol addresses this issue through the commitment scheme in VOLE-based ZK, where the IP and the input commitments of the proof are bound. During the exchange, these commitments are opened and verified. By shifting the verification focus from design properties to key validity, our framework decouples the checking process from the design while reducing complexity. Furthermore, applying individual commitments to each clause within the formula enables gradual release during the transaction. By facilitating fair transactions through optimistic fair exchange [1, 10] or smart contracts [52], the two parties can achieve a fair trade.

6 Implementation and Results

6.1 Implementation

We implement our protocol $\Pi_{\text{ZK-CEC}}$ based on the open-source code of ZKUNSAT [27]. We use the EMP-ZK library of the EMP-toolkit [48] as our VOLE-based ZK backend. We use the NTL library [44] for polynomial arithmetic over finite fields. The finite field instantiated in our implementation is $\mathbb{F}_{128}$. We use libsodium [8] for generating the key k and implementing BLAKE2b [41] as the keyed hash function in our protocol, where BLAKE2b can serve directly as a PRF. The largest bitwidth of literals w_{lit} is set to be 64. We represent the clause indices as 24-bit integers, which supports refutation proof with at most 2^{23} resolutions. Additionally, we make some optimizations to the original ZKUNSAT code.

6.2 Evaluation Setup

We evaluate ZK-CEC on a cloud computing node that equips a Lenovo ThinkSystem SR645 server with an AMD Rome 7H12 processor. In our experimental configuration, we run the prover and the verifier on a single node and allocate 80GB RAM for each party.

Table 2: Evaluation results.

Circuit	Lits.	Cls.	R	W	$t_{\mathbf{P1}}$ (s)	$t_{\mathbf{P2}}$ (s)	$t_{\mathbf{P3}}$ (ms)	$t_{\mathbf{P4}}$ (ms)	$t_{\mathbf{Total}}$ (s)	R'	$t'_{\mathbf{Total}}$ (s)	Speedup(×)
Adder_4x4	25	58	1225	22	0.01	3.30	0.41	1.29	3.31	90	1.76	1.88
Adder_5x5	32	75	1980	22	0.02	4.85	0.65	2.06	4.88	103	2.38	2.05
Adder_6x6	39	92	1938	18	0.02	4.33	0.82	2.85	4.35	138	2.16	2.01
Adder_7x7	47	111	1500	19	0.01	3.78	1.24	4.14	3.80	136	1.97	1.93
Adder_8x8	54	128	1671	17	0.02	4.01	1.49	5.10	4.04	145	2.05	1.97
Adder_9x9	61	145	2272	17	0.02	5.22	2.11	31.71	5.27	183	2.54	2.07
Adder_10x10	68	162	2565	19	0.02	5.76	2.72	67.24	5.85	207	2.94	1.99
Comp_6x6	53	121	2803	23	0.05	6.61	1.74	3.87	6.66	163	2.94	2.27
Comp_7x7	59	135	3850	29	0.08	9.42	2.61	19.35	9.52	212	4.22	2.26
Comp_8x8	72	168	4770	31	0.14	11.87	4.35	8.55	12.02	220	5.21	2.31
Comp_9x9	70	156	3777	26	0.08	8.75	3.43	27.44	8.87	226	3.87	2.29
Comp_10x10	82	86	5570	33	0.21	14.20	5.76	12.39	14.43	298	6.03	2.39
Mul_fp8	130	354	10828	33	0.32	26.66	16.72	281.25	27.27	373	11.44	2.38
Mul_2x2	12	24	199	14	0.00	1.22	0.07	0.25	1.22	34	1.16	1.05
Mul_3x3	37	101	1970	25	0.02	4.98	1.09	2.88	5.01	97	2.42	2.07
Mul_4x4	84	246	16250	50	0.83	52.00	10.90	136.99	52.97	412	20.13	2.63
Mul_5x5	144	436	100619	72	9.06	414.48	46.44	506.48	424.09	1640	159.85	2.65
Mul_6x6	222	683	667234	106	112.87	4010.25	282.75	1376.32	4124.78	7253	1454.74	2.84
Arbiter_4	17	21	97	12	0.00	1.16	0.08	0.19	1.16	27	1.00	1.16
Barrel_shifter_4	26	58	879	19	0.01	2.57	0.38	1.03	2.58	72	1.55	1.66
Barrel_shifter_8	72	180	1891	39	0.10	6.25	5.60	56.41	6.41	94	2.80	2.29
CRC_8	37	84	3304	24	0.05	7.62	0.89	2.38	7.67	231	3.42	2.24
CRC_16	87	206	16742	36	0.54	42.23	7.19	71.58	42.85	938	17.97	2.38
Decoder_3to8	19	47	443	22	0.01	1.70	0.27	1.04	1.70	37	1.30	1.31
Decoder_4to16	31	78	1615	38	0.04	5.20	1.04	4.37	5.24	71	2.49	2.10
Decoder_5to32	58	157	5842	72	0.62	26.15	6.87	134.10	26.91	139	10.34	2.60
FSM	20	48	229	12	0.00	1.29	0.17	0.47	1.29	31	1.13	1.14
Parity_8	15	28	330	11	0.00	1.43	0.09	0.14	1.43	53	1.19	1.20
Encoder_8	26	51	326	16	0.00	1.43	0.29	0.90	1.44	58	1.20	1.20
Encoder_16	67	147	1145	24	0.01	3.34	2.94	21.80	3.38	144	1.85	1.83
Voting_9	52	140	3332	24	0.06	7.51	2.09	3.46	7.57	186	3.33	2.27
Voting_17	103	282	60565	40	2.16	159.63	12.69	101.17	161.91	2063	63.51	2.55
GFMul_4x4	42	120	10441	30	0.24	23.90	1.71	3.63	24.14	392	9.72	2.48
GFMul_8x8	169	538	1582909	109	284.76	9691.58	98.72	896.82	9977.33	30303	3646.55	2.74
S-box_AES	211	700	236862	298	275.60	4660.74	720.31	14 298.10	4951.36	1754	1763.09	2.81
S-box_ASCON	24	68	3159	35	0.09	9.21	0.65	43.44	9.34	120	3.77	2.48
S-box_PRESENT	57	155	1476	29	0.02	4.60	2.94	29.13	4.66	98	2.24	2.08
S-box_SM4	124	403	188098	300	228.91	3770.04	144.12	7727.02	4006.82	1420	1450.96	2.76

The prover and verifier communicate via the local network. As noted in [26, 27], the bandwidth requirement between the two parties is approximately 100 Mbps; therefore, network communication does not constitute a performance bottleneck in our evaluation.

A set of 37 combinational circuits is used as the benchmark of our evaluation. These include arithmetic operators (adders, multipliers, comparators, and floating-point multipliers) and control logic such as barrel shifters, decoders, priority encoders, FSM transition logic, and arbiters. We also include error-detection modules (parity and cyclic redundancy checks) and cryptographic components, which involve finite field multipliers and S-Boxes for AES, SM4, Ascon, and Present. Finally, voting circuits are included as a common logic in multi-party computation.

We use Synopsys Design Compiler to synthesize the benchmark into gate-level netlists. For each circuit, we manually constructed a miter circuit by combining the netlist with its specification, which is then transformed into a CNF formula. We use PicoSAT [4] to solve the formula and generate the refutation proof.

6.3 Evaluation Results

Table 2 presents the evaluation results for the benchmark circuits. In the table, Lits. and Cls. denote the number of literals and clauses, respectively, in the CNF formula derived from the implementation

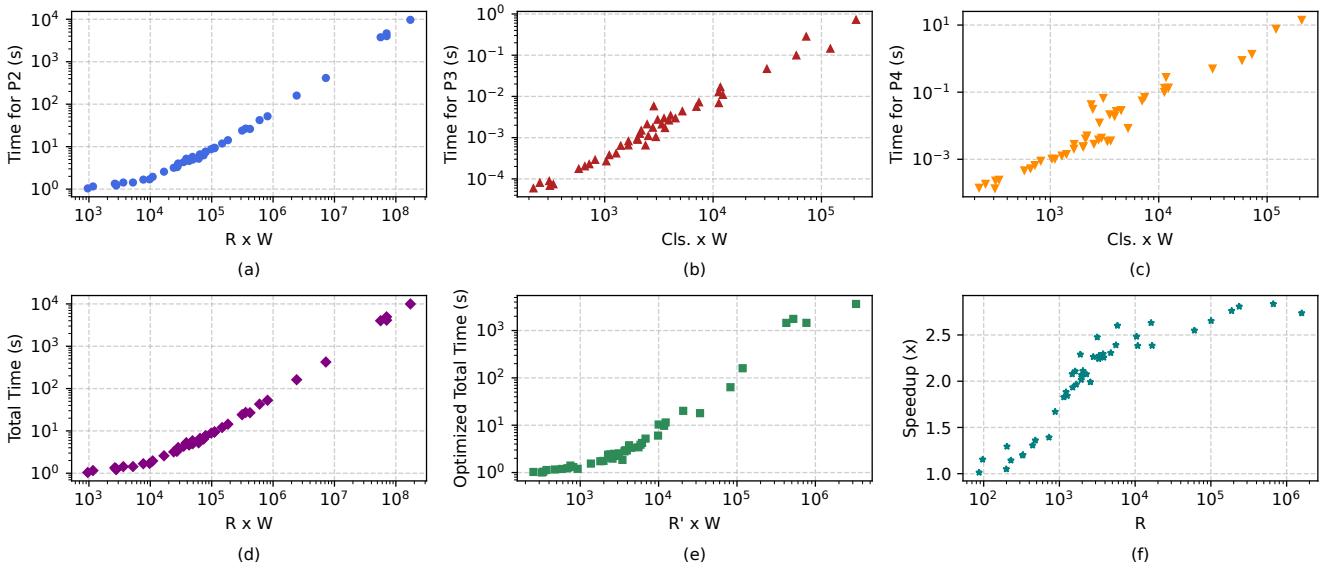


Figure 13: Proving time vs. input size: (a) Proving time or P2 (b) Proving time for P3 (c) Proving time for P4 (d) Total proving time (e) Total proving time after optimization (f) Speedup in the total proving time after optimization.

circuit. R and W represent the total number of stored resolvents (i.e., the proof length) and the padded width of the encoded polynomials. The execution time for verifying each property is denoted by t_{Pi}. In our implementation, the prover commits the resolvent clauses during the execution of Π_{P1}; therefore, the running times for Π_{P1} include the time of commitment. We also implement and evaluate the optimization in Sec. 5.8. R' and t'_{Total} denote the compressed proof length and the total execution time under the optimization, and the last column provides the speedup over the baseline after optimization.

As shown in Table 2, the verification of the UNSAT (Π_{P2}) accounts for the majority of the execution time. This observation is consistent with the fact that UNSAT is a co-NP-complete problem; the length of the resolution refutation R can grow exponentially relative to the size of the initial formula Φ. In contrast, the verification costs for Π_{P3} (i.e. proving SAT) and Π_{P4} scale linearly with the size of Φ_{impl}. As the scale of the design increases, the refutation becomes the dominant performance bottleneck.

Fig. 13 (a) illustrates the correlation between t_{P2} and the refutation proof size. Fig. 13 (b) and 13 (c) depict the proving times for Π_{P3} and Π_{P4} as a function of the size of Φ_{impl}, respectively. Given that proving UNSAT is dominant, Fig. 13 (d) compares the total execution time against the refutation proof size. These four time metrics exhibit a near-linear relationship on a log-log scale. Notably, as we scaled our evaluation to more complex combinational logic designs, we inevitably encountered resource constraints. The explosive growth of the refutation proofs for larger circuits led to out-of-memory and time-outs, which are the current scalability limits of our framework. We further evaluate our protocol with optimization. Fig. 13 (e) illustrates the correlation between total execution time and the compressed proof size, while Fig. 13 (f) depicts the speedup compared to the baseline. Notably, the speedup grows with the scale of design.

7 Limitations and Discussion

While ZK-CEC represents a foundational step toward zero-knowledge formal hardware verification, its practical deployment is currently constrained by the inherent scalability limits of both formal equivalence checking and the underlying zero-knowledge backend. Although the proposed framework imposes no theoretical bounds on circuit size, the required resolution refutation proofs can grow exponentially as the scale and complexity of the implementation circuit increase. This exponential growth translates directly into significant memory consumption and extended proving times. Additionally, the cryptographic operations required by the zero-knowledge framework introduce substantial computational overhead. As demonstrated in Table 3, ZK-CEC incurs a roughly 2500× overhead when verifying 4-bit and 5-bit multipliers compared to plaintext SAT solving. Together, these factors dictate the scalability limits of our current implementation.

To address these limitations and enable the verification of larger hardware IPs, future deployments of ZK-CEC can adopt a modular verification strategy. Following standard practices in electronic design automation (EDA), complex designs can be partitioned into smaller, more manageable modules for independent verification, thereby drastically reducing overall proof complexity [14, 33]. Furthermore, this modular approach could naturally extend the capabilities of the ZK-CEC framework to verify sequential circuits by checking the combinational logic between pipeline registers.

Table 3: Runtime Comparison between ZKCEC and SAT Solving

Design	SAT Solver	ZKCEC	Overhead
Mult_4x4	0.0082 s	20.13 s	2454.88×
Mult_5x5	0.0636 s	159.55 s	2508.65×

8 Related Work

Privacy Preserving Hardware Verification. The line of research most closely aligned with our work is privacy-preserving hardware verification. Notably, Pythia [32] encodes the vendor's circuit and puts it into a zkVM as witness input. This framework allows the verifier to perform evaluations using private stimuli and prove output correctness, thereby enabling simulation-based verification between IP vendors and buyers in zero-knowledge. Building on simulation, zk-Sherlock [30] calculates the toggle rates of signals in the netlist within a zkVM, and the verifier can analyze the switching activity to ensure that the design is free of hardware Trojan triggers. Leveraging multi-party computation and logic locking, [31] realizes privacy-preserving IP evaluation with high scalability and performance. Romeo [13] and [22] rely on HE for privacy-preserving IP verification. Simulation-based verification often struggles to catch corner cases. Our work is the first zero-knowledge formal hardware verification framework, providing a proof of functional equivalence between the IP and the specification.

Formal Methods in Zero-Knowledge. Recent advancements have introduced specialized arithmetization techniques to prove logic properties within zero-knowledge frameworks. ZK-SMT [26] enables the validation of SMT formulas in zero-knowledge based on a set of inference rules, while zkPi [24] provides a zkSNARK for proofs in Lean, capable of expressing virtually all systems of formal reasoning. Other works focus on specific applications: Cheesecloth [6] is a statement compiler that proves program vulnerabilities, and Crepe [21] proves regular expression equivalence (a PSPACE-complete problem). However, these existing approaches to formal reasoning operate exclusively on public inputs, hiding only the proof witness. Although the authors of [27] state secret program analysis as a potential direction, they do not implement it due to unresolved consistency challenges. Our work presents the first framework for the formal verification of private inputs, leveraging the underlying structure of the formula.

VOLE-based Zero-Knowledge Proof. VOLE-based zero-knowledge proof protocols have been a line of research in recent years [2, 9, 49–51]. Wolverine [49] first proposes a ZKP system that enables an efficient prover in both running time and memory usage. LPZK [9] focuses on utilizing the linear relationships of VOLE-based commitments. QuickSilver [51] is a hybrid approach of Wolverine and LPZK, which supports both large and small fields while avoiding cut-and-choose. Mystique [50] allows the conversions between arithmetic and Boolean commitments and improves the matrix multiplication operation in the protocol, which enables a zero-knowledge proof of high-precision deep neural network inference.

9 Conclusion

This paper proposes the first privacy-preserving hardware formal verification protocol, ZK-CEC, for proving circuit functional equivalence without revealing the structure of the secret design, which aims to build trust between IP vendors and buyers. The proposed main protocol $\Pi_{\text{ZK-CEC}}$ achieves completeness, soundness, and zero-knowledge. In the paper, we first analyze the soundness issue of the existing ZK framework on the secret formula verifying problem. Next, we propose a general blueprint for privacy-preserving property checking. Based on our blueprint, we design four sub-protocols and compose them into our main protocol. Finally, we implement and evaluate our protocol upon a diverse set of common circuits, which demonstrates that our blueprint only brings little overhead to the existing ZK framework. For future work, the proposed blueprint can be applied to other verification scenarios, such as software and cyber-physical systems. Future research may also explore leveraging LLMs to translate text-based specs into formulas [5] for formal verification through our proposed blueprint.

Acknowledgments

Zunchen Huang and Chenglu Jin are (partially) supported by project CiCS of the research programme Gravitation, which is (partly) financed by the Dutch Research Council (NWO) under the grant 024.006.037. We thank SURF for providing computational resources. We thank anonymous reviewers for their constructive feedback. We thank Marten van Dijk and Ruzica Piskac for their insightful discussions regarding zero-knowledge proofs for formal verification. This paper was edited for grammar using Gemini.

References

[1] Nadarajah Asokan, Victor Shoup, and Michael Waidner. 1998. Optimistic fair exchange of digital signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 591–606.
[2] Carsten Baum, Alex J Malozemoff, Marc B Rosen, and Peter Scholl. 2021. Mac n Cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *Annual International Cryptology Conference*. Springer, 92–122.
[3] Alex Baumgarten, Michael Steffen, Matthew Clausman, and Joseph Zambreno. 2011. A case study in hardware Trojan design and implementation. *International Journal of Information Security* 10, 1 (2011), 1–14.
[4] Armin Biere. 2008. PicoSAT essentials. *Journal on Satisfiability, Boolean Modelling and Computation* 4, 2-4 (2008), 75–97.
[5] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*. Springer, 383–396.
[6] Santiago Cuéllar, Bill Harris, James Parker, Stuart Pernsteiner, and Eran Tromer. 2023. Cheesecloth:{Zero-Knowledge} proofs of real world vulnerabilities. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6525–6540.
[7] Martin Davis and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the ACM (JACM)* 7, 3 (1960), 201–215.
[8] Frank Denis. [n. d.]. *libsodium*. https://github.com/jedisct1/libsodium
[9] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. 2020. Line-point zero knowledge and its applications. *Cryptology ePrint Archive* (2020).
[10] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. 2020. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 543–557.
[11] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. 2021. Constant-overhead zero-knowledge for RAM programs. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 178–191.
[12] Evguenii I Goldberg, Mukul R Prasad, and Robert K Brayton. 2001. Using SAT for combinational equivalence checking. In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*. IEEE, 114–121.
[13] Charles Gouert and Nektarios Georgios Tsoutsos. 2020. Romeo: Conversion and evaluation of HDL designs in the encrypted domain. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[14] Orna Grumberg and David E Long. 1994. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 3 (1994), 843–871.
[15] Syed Kamran Haider, Chenglu Jin, Masab Ahmad, Devu Manikantan Shila, Omer Khan, and Marten van Dijk. 2019. Advancing the state-of-the-art in hardware trojans detection. *IEEE Transactions on Dependable and Secure Computing* 16, 1 (2019), 18–32.
[16] Syed Kamran Haider, Chenglu Jin, and Marten van Dijk. 2017. Advancing the state-of-the-art in hardware trojans design. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 823–826.
[17] John M Hitchcock and Aduri Pavan. 2015. On the NP-completeness of the minimum circuit size problem. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 236–245.
[18] Yier Jin and Yiorgos Makris. 2008. Hardware Trojan detection using path delay fingerprint. In *2008 IEEE International workshop on hardware-oriented security and trust*. IEEE, 51–57.
[19] Yier Jin and Yiorgos Makris. 2012. Proof carrying-based information flow tracking for data secrecy protection and hardware trust. In *2012 IEEE 30th VLSI Test Symposium (VTS)*. IEEE, 252–257.
[20] Ashwin Karthikeyan, Hengyu Liu, Kuldeep S Meel, and Ning Luo. 2025. Towards Practical Zero-Knowledge Proof for PSPACE. *arXiv preprint arXiv:2511.15071* (2025).
[21] John C Kolesar, Shan Ali, Timos Antonopoulos, and Ruzica Piskac. 2025. Coinductive Proofs of Regular Expression Equivalence in Zero Knowledge. *Proceedings of the ACM on Programming Languages* 9, OOPSLA2 (2025), 357–385.
[22] Charalambos Konstantinou, Anastasis Keliris, and Michail Maniatakos. 2015. Privacy-preserving functional IP verification utilizing fully homomorphic encryption. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 333–338.
[23] Santanu Kundu and Santanu Chattopadhyay. 2018. *Network-on-chip: the next generation of system-on-chip integration*. Taylor & Francis.
[24] Evan Laufer, Alex Ozdemir, and Dan Boneh. 2024. zkpi: Proving lean theorems in zero-knowledge. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4301–4315.
[25] Eric Love, Yier Jin, and Yiorgos Makris. 2011. Proof-carrying hardware intellectual property: A pathway to trusted module acquisition. *IEEE Transactions on Information Forensics and Security* 7, 1 (2011), 25–40.
[26] Daniel Luick, John C Kolesar, Timos Antonopoulos, William R Harris, James Parker, Ruzica Piskac, Eran Tromer, Xiao Wang, and Ning Luo. 2024. {ZKSMT}: A {VM} for Proving {SMT} Theorems in Zero Knowledge. In *33rd USENIX Security Symposium (USENIX Security 24)*. 3837–3845.
[27] Ning Luo, Timos Antonopoulos, William R Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. 2022. Proving UNSAT in zero knowledge. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2203–2217.
[28] Joao P Marques-Silva and Karem A Sakallah. 2002. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48, 5 (2002), 506–521.
[29] Alexander Maximov and Patrik Ekdahl. 2019. New circuit minimization techniques for smaller and faster AES SBoxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), 91–125.
[30] Dimitris Mouris, Charles Gouert, and Nektarios Georgios Tsoutsos. 2022. Zksherlock: Exposing hardware trojans in zero-knowledge. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 170–175.
[31] Dimitris Mouris, Charles Gouert, and Nektarios Georgios Tsoutsos. 2023. MPoC: Privacy-Preserving IP Verification Using Logic Locking and Secure Multiparty Computation. In *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–8.
[32] Dimitris Mouris and Nektarios Georgios Tsoutsos. 2020. Pythia: Intellectual property verification in zero-knowledge. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[33] Rajdeep Mukherjee, Daniel Kroening, Tom Melham, and Mandayam Srivas. 2015. Equivalence checking using trace partitioning. In *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 13–18.
[34] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. 2012. A new approach to practical active-secure two-party computation. In *Annual Cryptology Conference*. Springer, 681–700.
[35] Daniel Perrucci and Juan Sabia. 2007. Real roots of univariate polynomials and straight line programs. *Journal of Discrete Algorithms* 5, 3 (2007), 471–478.
[36] Reza M Rad, Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. 2008. Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 632–639.
[37] Sandip Ray, Wen Chen, and Rosario Cammarota. 2018. Protecting the supply chain for automotives and IoTs. In *Proceedings of the 55th Annual Design Automation Conference*. 1–4.
[38] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. 2018. Smashing the implementation records of AES S-box. *IACR transactions on cryptographic hardware and embedded systems* (2018), 298–336.
[39] John Alan Robinson. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)* 12, 1 (1965), 23–41.
[40] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. 2014. A primer on hardware security: Models, methods, and metrics. *Proc. IEEE* 102, 8 (2014), 1283–1295.
[41] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. 2015. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693. doi:10.17487/RFC7693
[42] Resve Saleh, Steve Wilton, Shahriar Mirabbasi, Alan Hu, Mark Greenstreet, Guy Lemieux, Partha Pratim Pande, Cristian Grecu, and Andre Ivanov. 2006. System-on-chip: Reuse and integration. *Proc. IEEE* 94, 6 (2006), 1050–1069.
[43] Sirui Shen, Zunchen Huang, and Chenglu Jin. 2026. Proving Circuit Functional Equivalence in Zero Knowledge. *arXiv preprint arXiv:2601.11173* (2026).
[44] Victor Shoup et al. 2001. NTL: A library for doing number theory.
[45] Yunqing Sun, Hanlin Liu, Kang Yang, Yu Yu, Xiao Wang, and Chenkai Weng. 2025. Committed Vector Oblivious Linear Evaluation and Its Applications. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*. 3635–3648.
[46] Grigori S Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*. Springer, 466–483.
[47] Adam Waksman and Simha Sethumadhavan. 2011. Silencing hardware backdoors. In *2011 IEEE Symposium on Security and Privacy*. IEEE, 49–63.
[48] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.
[49] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1074–1091.
[50] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*. 501–518.
[51] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2986–3001.
[52] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. 2020. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems* 105 (2020), 475–491.

A Proofs

A.1 Proof of Lemma 4.1

We prove Lemma 4.1 by contradiction. Given $\Phi_a \wedge \Phi_b$ is unsatisfiable, and two assignment sets that are not disjoint, then there must exist an assignment ω_s on $\mathcal{V}_s = \mathcal{V}_a \cap \mathcal{V}_b$, such that $\omega_s \in \{\omega \mid \omega \models \exists \mathcal{V}'_a \Phi_a\} \cap \{\omega \mid \omega \models \exists \mathcal{V}'_b \Phi_b\}$. By the definition of existential quantification, any model (i.e., assignment) of $\exists x \Phi$ can be lifted to a model of the original formula Φ. Therefore, we lift ω_s to ω_a and ω_b for Φ_a and Φ_b, respectively. We define a new assignment ω_{union} over $\mathcal{V}_a \cup \mathcal{V}_b$ as:

$$\omega_{union}(x) = \begin{cases} \omega_s(x) & \text{if } x \in \mathcal{V}_a \cap \mathcal{V}_b \\ \omega_a(x) & \text{if } x \in \mathcal{V}_a \setminus \mathcal{V}_s \\ \omega_b(x) & \text{if } x \in \mathcal{V}_b \setminus \mathcal{V}_s \end{cases}$$

Since $\omega_{union} \models \Phi_a$ and $\omega_{union} \models \Phi_b$, we have $\omega_{union} \models \Phi_a \wedge \Phi_b$. This implies that $\Phi_a \wedge \Phi_b$ is satisfiable, which contradicts the initial assumption that $\Phi_a \wedge \Phi_b$ is unsatisfiable. This concludes the proof.

A.2 Proof Sketch of $\Pi_{\mathbf{P3}}$ Conditional Soundness

Consider a malicious prover $\mathcal{P}^*$. If Φ_{sec} is unsatisfiable, there are two potential strategies for the prover to cheat in the protocol:

(1) Cheating within functionalities $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{ZK-Poly}}$. The prover may attempt to manipulate the protocol by modifying its own shares in the VOLE-based commitments, thereby binding the

commitments to malicious values to force the verifier to accept. However, according to the proofs in [51], as long as the verifier does not disclose its own commitment shares to $\mathcal{P}^*$, the probability that $\mathcal{F}_{ZK}$ and $\mathcal{F}_{ZK\text{-}poly}$ accept during the Equal and PoPIdt instructions, when they should abort, is $\text{negl}(\lambda)$.

(2) Cheating on the assignment of Φ_{sec}. Assuming no cheating occurs within $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{ZK-Poly}}$, the prover may attempt to provide a malicious assignment ω^* to convince the verifier. By the soundness of the resolution rule [39], if a formula is satisfiable, any clause derived from it via resolution must also be satisfied by the same assignment. Since Φ_{sec} is unsatisfiable, there exists a resolution derivation that results in a contradiction, specifically two complementary literals l and $\neg l$. For ω^* to satisfy both, it must contain this pair of complementary literals. If $\phi(l)$ and $\phi(\neg l)$ satisfy the encoding constraint in Eq. 6, the protocol will abort at Step (3) of Π_{P3}. If $\phi(l)$ and $\phi(\neg l)$ do not satisfy this constraint, the inconsistency will be detected, and the preceding protocol Π_{P2} will abort. Consequently, the probability of $\mathcal{P}^*$ successfully cheating in Case 2 is zero.

Based on the analysis of the cases, when Φ_{sec} is unsatisfiable, a malicious prover has only a negligible probability of convincing the verifier that Φ_{sec} is satisfiable.

A.3 Proof Sketch of Theorem 5.1

We now prove that the protocol $\Pi_{\text{ZK-CEC}}$ satisfies completeness, soundness, and zero-knowledge.

Completeness. Completeness follows directly from the deterministic construction of the protocol.

Soundness. First, the probability of $\mathcal{P}^*$ successfully cheating in the functionalities $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{ZK-Poly}}$ is bounded by a negligible function $\text{negl}(\lambda)$. Therefore, the ZK backend ensures that any attempt to cheat in sub-protocols Π_{P1} and Π_{P4} will make the protocol abort with overwhelming probability. The soundness of the refutation in Π_{P2} depends on the validity of the commitment $[\Phi_{\text{sec}}]$. According to Lemma 4.1, Π_{P3} and Π_{P4} guarantee that the statement $[\Phi_{\text{sec}}]$ in Π_{P2} is valid and non-contradictory. This ensures that any derived contradiction originates from the interface $\mathcal{V}_{\text{IO}}$ rather than a forged internal conflict, thereby establishing the soundness of Π_{P2}. Regarding the conditional soundness in Π_{P3}, if the complementary literals do not satisfy the enforced constraints, the resolution process in Π_{P2} will result in an abort. Since each component is either cryptographically secure or logically constrained by the sub-protocols, $\Pi_{\text{ZK-CEC}}$ satisfies the soundness property.

Zero-knowledge. We construct a simulator S that holds a circuit C'_{impl}, where $C'_{\text{impl}} \equiv C_{\text{spec}}$. Under our defined leakage model, C'_{impl} is indistinguishable from the real implementation C_{impl} with respect to refutation proof size and gate-count. Throughout the protocol execution, the verifier's view is restricted to VOLE commitments and the abort signal. The VOLE-based ZK backend instantiated in $\Pi_{\text{ZK-CEC}}$ ensures that all commitments and polynomial evaluations achieve information-theoretic security [51]. Therefore, in the verifier's view, all commitments are indistinguishable from uniform random values. Consequently, the simulated interaction is indistinguishable from the real protocol execution, thereby satisfying the zero-knowledge property.

B An Example of the Commitments of Polynomials

We first provide a simple example of the conversion from a verification problem to polynomials. We use the example in Fig. 7, which contains an AND gate and a property for verification. The formula is $\Phi = (a \vee \neg c) \wedge (b \vee \neg c) \wedge (\neg a \vee \neg b \vee c) \wedge c \wedge \neg a$, where a and c is public variable, while b is secret.

We rename each variable with an index from 1 to 3, and let l_i denote the literal corresponding to the i-th variable, such that $a = l_1$, $b = l_2$, and $c = l_3$. The formula is:

$$\Phi = (\neg l_1 \vee \neg l_2 \vee l_3) \wedge (l_1 \vee \neg l_3) \wedge (l_2 \vee \neg l_3) \wedge l_3 \wedge \neg l_1$$

The formula Φ is a conjunction of 5 clauses; therefore, we represent Φ with 5 polynomials. Take the first clause as an example; the encoded polynomial is:

$$\begin{aligned}&\text{Poly}[(\neg l_1 \vee \neg l_2 \vee l_3)](x)\\ =&(x + \phi(\neg l_1))(x + \phi(\neg l_2))(x + \phi(l_3))\\ =&(x + (1 + const_{\text{lit}}) + const_{\text{pub}})(x + (H_k(2) + const_{\text{lit}}))\\ &(x + 3 + const_{\text{pub}})\end{aligned}$$

When we commit the polynomial, we transform the polynomial to its coefficient representation $a_3x^3 + a_2x^2 + a_1x^1 + a_0$ and commit it as $\{[a_3], [a_2], [a_1], [a_0]\}$. By committing all clauses in Φ, we complete the commitment of the entire formula.

Next, an example of a resolution step encoded in polynomial relations is provided. Let the two antecedent clauses be $c_0 = l_1 \vee \neg l_3$ and $c_1 = l_3$. Their encoded polynomials are:

$$\begin{aligned}&\text{Poly}[c_0](x) = (x + 1 + const_{\text{pub}})(x + (3 + const_{\text{lit}}) + const_{\text{pub}})\\ &\text{Poly}[c_1](x) = (x + 3 + const_{\text{pub}})\end{aligned}$$

The witness clauses are $w_0 = \emptyset$ and $w_1 = l_1$, and the resolvent clause and the pivot are $c_r = l_1$ and $p = l_3$, respectively. Their corresponding polynomials are:

$$\begin{aligned}&\text{Poly}[w_0](x) = 1\\ &\text{Poly}[w_1](x) = (x + 1 + const_{\text{pub}})\\ &\text{Poly}[c_r](x) = (x + 1 + const_{\text{pub}})\\ &\text{Poly}[p](x) = (x + 3 + const_{\text{pub}})\\ &\text{Poly}[\neg p](x) = (x + (3 + const_{\text{lit}}) + const_{\text{pub}})\end{aligned}$$

Following Eq. 1, we check a single resolution step by proving the following polynomial relation:

$$\begin{aligned}&\text{Poly}[w_0](x) \cdot \text{Poly}[c_0](x) = \text{Poly}[c_r](x) \cdot \text{Poly}[\neg p](x)\\ &\text{Poly}[w_1](x) \cdot \text{Poly}[c_1](x) = \text{Poly}[c_r](x) \cdot \text{Poly}[p](x)\end{aligned}$$

C Ethical Considerations

This work follows ethical research guidelines and aims to improve hardware supply-chain security without compromising vendors' intellectual property.

D Open Science

Our research adheres to the Open Science policy to ensure transparency, reproducibility, and long-term impact. The full source code for our ZK-CEC protocol implementation is available at https://github.com/NomadShen/ZKCEC.