

BlindMarket: Enabling Verifiable, Confidential, and Traceable IP Core Distribution in Zero-Trust Settings

Zhaoxiang Liu*, Samuel Judson†, Raj Dutta‡, Mark Santolucito§, Xiaolong Guo*, Ning Luo¶

*Kansas State University {zxliu, guoxiaolong}@ksu.edu

†Trail of Bits samuel.judson@trailofbits.com

‡Silicon Assurance rajgautamdutta@siliconassurance.com

§Barnard College, Columbia University msantolu@barnard.edu

¶University of Illinois Urbana-Champaign nl27@illinois.edu

Abstract—We present BlindMarket, an end-to-end zero-trust distribution framework for hardware IP cores that operates without relying on any trusted third party. BlindMarket allows two parties, the IP user and the IP vendor, to complete an IP trading process with strong guarantees of verifiability and confidentiality before the transaction, and then traceability after. We propose verification heuristics and adapt the cone of influence-based design pruning to overcome the limited scalability common to cryptographic protocols and the hardness of the underlying hardware verification. We systematically evaluate our framework on a diverse set of real-world hardware benchmarks. The results demonstrate that BlindMarket successfully verifies 12 out of 13 IP designs and achieves substantial performance improvements enabled by design pruning and control-flow guided heuristics.

Index Terms—Hardware Security & Piracy, IP Verification, Secure Multiparty Computation

I. INTRODUCTION

The integration of third-party intellectual property (IP) cores into system-on-chip (SoC) designs is now a standard practice in semiconductor development. By 2021, about 1.15 trillion devices had been shipped with chips containing various IP cores, highlighting the market's scale [1]. In this market, *IP vendors* such as Arm, Intel, Cadence, and Synopsys provide the IP core design together with supplementary documentation and test benches to the *IP user*. Their revenue primarily comes from licensing these high-performance hardware designs and charging a fee for each chip manufactured that incorporates the IP.

Formal verification enables IP users to exhaustively verify design properties before purchase. SAT-based hardware verification stands out due to recent breakthroughs in SAT solving [2], [3], [4], [5], [6], [7]. However, the nature of formal verification requires the exposure of the IP core to the IP user so that the latter can undertake the verification, which raises significant risks of IP theft [8], [9], [10], [11]. The IP vendor is at risk of IP theft and commercial loss without strict protections when they provide temporary access to their products and disclose detailed product information prior to a

transaction closing. Privacy leakage is, however, bidirectional. An untrusted IP vendor can exploit the specifications to infer the IP user's design needs and purchasing intentions. Such knowledge not only exposes system details of value to competitors [12], but also enables the vendor to manipulate pricing, posing additional commercial risks.

Existing IP core protection methods focus on preventing illegal hardware copying through techniques such as obfuscation and watermarking [13], [14], [15], [16]. This line of work is orthogonal to ours, which aims to preserve confidentiality throughout the IP verification phase. While legitimate efforts to shield hardware IP, such as the IEEE 1735 standard [17], can help, they have been proven insufficient to prevent theft [18], leading to financial losses across the industry. Recent demands for *zero-trust* approaches highlight the need for a secure framework that enables IP exchange without reliance on trusted third parties [17].

There remains a strong need for IP core traceability, which allows users to identify the origin of an IP and avoid redundant verification. The risks of untraceable hardware gained renewed attention in 2024 during the Israel–Hezbollah conflict, where explosive-laden pagers were falsely labeled as Gold Apollo products [19], underscoring the security implications of missing traceability in critical systems.

In sum, an ideal IP core distribution platform should achieve the following three goals, all without a trusted party:

Verifiability. The platform should guarantee that an acquired IP core behaves as advertised. To this end, the IP user must be able to perform IP verification before entering an agreement with the vendor.

Confidentiality. The IP vendor's proprietary designs and the IP user's sensitive data should both remain private until a legal framework is established, enabling integration.

Traceability. The platform should offer verifiable provenance and integrity checks for IPs, ensuring that distributed IPs are authentic while remaining agnostic to post-acquisition use.

Achieving verifiability, confidentiality, and traceability simultaneously in hardware IP distribution is challenging due to inherent tensions among these goals. Verifiability often

† Work completed while at Yale University.

* Zhaoxiang Liu's work was completed while visiting UIUC.

requires exposing design details to the user, compromising confidentiality. Likewise, embedding traceability markers can alter control or data paths, potentially breaking logic equivalence checking. To mitigate such tensions, a natural direction is to leverage cryptographic techniques for verification. ppSAT [20] was originally designed to protect the privacy of Boolean satisfiability queries, and its techniques, though applicable in principle, have not been applied to hardware and do not provide an end-to-end zero-trust workflow for real-world IP trading. Its limited scalability further restricts its practicality for hardware IP verification.

To address the above limitations, we propose BlindMarket. Our contributions are summarized as follows.

- We propose an end-to-end “zero-trust” framework, BlindMarket, that allows the IP user and the vendor to establish a trusted IP trading process to privately select and verify IP cores of interest before the transaction takes place without relying on any trusted third-party.
- We turn the idea of privacy-preserving verification into a practical end-to-end hardware verification framework with tailored formula generation and hardware-focused optimizations.
- We design a cryptographic ledger-based IP ownership auditing mechanism that complements BlindMarket by enabling post-trade validation of provenance and ownership. This mechanism provides verifiable traceability of IP cores across transactions.
- We implement our framework and evaluate it on real-world IP cores. The results demonstrate that our approach efficiently verifies functional properties, achieving performance improvements in 12 out of 13 benchmarks. Its effectiveness is further illustrated in Figure 5.

II. PRELIMINARIES

A. SAT-based Hardware Verification

Many hardware verification tasks can be reduced to SAT problems. The hardware design can be modeled as a transition system, and its behavior over a bounded number of time steps can be encoded as a Boolean formula ϕ_D by unrolling the transition relation across multiple cycles. The property to be verified in a hardware design is expressed as a Boolean formula φ . In the open-source hardware community, temporal properties are of much interest. In this paper, we focus on the property using the operator of Overlap Implication (OI), Non-Overlap Implication (NOI), and Concatenation (Concat) [21]. Together, these operators provide a composable basis for expressing temporal relationships over bounded execution traces. Specifically, OI ($A \rightarrow B$) asserts that whenever A holds, B must hold at the same clock cycle, whereas NOI ($A \Rightarrow B$) requires B to hold in the next clock tick if A holds in the current clock tick. The Concat operator ($A \## NB$) checks that A must be true on the current clock tick and B on the N^{th} clock tick. Here, A and B are boolean expressions defined as a sequence [22]. One typically constructs the conjunction $\phi_D \wedge \neg\varphi$ in conjunctive normal form (CNF). If this formula is satisfiable, it reveals a counterexample that violates the

intended property; otherwise, the property holds within the given bound.

B. Cryptographic Tools

Privacy-Preserving SAT Solving Secure Two-Party Computation (2PC) is a cryptographic protocol that enables two parties with private inputs to jointly compute a function $f(a, b)$ without revealing anything beyond the final output.

Privacy-preserving SAT solving (ppSAT) [20] leverages 2PC to allow two parties to check the satisfiability of the conjunction of their CNF formulas without revealing the formulas themselves. In ppSAT, each party encodes its CNF formula as a pair of matrices. Let $\phi = \bigwedge_{j=1}^m C_j$ denote a CNF formula with m clauses over n variables. Each clause C_j is represented by a pair of binary vectors $(P[j], N[j])$ of length n , where the encoding is defined as follows:

$$(P[j][i], N[j][i]) = \begin{cases} (1, 0) & \text{if } x_i \text{ appears in } C_j, \\ (0, 1) & \text{if } \neg x_i \text{ appears in } C_j, \\ (0, 0) & \text{otherwise.} \end{cases}$$

The complete formula ϕ is encoded as a pair of binary matrices $(P, N) \in 0, 1^{n \times m}$, where each column j corresponds to the clause representation $(P[j], N[j])$.

We consider ppSAT as consisting of two phases: the **sharing** (ppSAT.share) phase and the **solving** (ppSAT.solving) phase. Each party first encodes its private Boolean formula as a pair of matrices, denoted by (P_a, N_a) and (P_b, N_b) , respectively. In the sharing phase, the two parties secret-share the concatenated matrices using Yao’s sharing [23]. A value v is said to be *Yao-shared*, denoted by $\llbracket v \rrbracket$, if it is the output of a GC-based 2PC computation and has not been revealed to either party. In the subsequent solving phase, ppSAT performs SAT solving by updating the shared matrix representation within the 2PC framework. Each update is referred to as a *giant step*, which consists of three procedures: *unit propagation*, *pure literal elimination*, and *variable decision branching*. In particular, *variable decision branching* relies on an external heuristic to determine which variable assignment to try when no variable can be directly decided, and the solving process must proceed by branching. **Oblivious transfer.** Oblivious transfer (OT) is a fundamental cryptographic primitive that enables a sender to transfer one of many pieces of information to a receiver, while preserving the privacy of both parties. In a *1-out-of- N* OT protocol, the sender holds a vector of messages $(m_1, m_2, \dots, m_N)$, and the receiver holds an index $\alpha \in \{1, \dots, N\}$. At the end of the protocol, the receiver learns m_α and nothing about $\{m_j\}_{j \neq \alpha}$ while the sender learns nothing about the receiver’s choice α . We denote this invocation as

$$(m_i, \perp) \leftarrow \text{OT}(i, \{m_1, m_2, \dots, m_N\})$$

Ledger System. A ledger is a record-keeping system that tracks the state and ownership of assets. A cryptographic ledger extends this concept with cryptographic primitives to provide verifiable storage and consensus. BlindMarket requires such a ledger for transparency but is agnostic to the

underlying consensus mechanism (centralised, decentralised, permissionless, *etc.*), as long as it supports reliable storage of cryptographically validated records and the simple queries defined below.

III. RELATED WORK

We compare our work against prior efforts from three perspectives: verifiability, confidentiality, and traceability.

TABLE I: Existing methods cannot provide privacy protection for formal-based IP verification without a third party.

Approach	Parties	Verification	Stage
TIPP [13], [14], [15]	-	-	Post-fabrication
IEEE 1735 [17]	3	Simulation/Synthesis	Pre-silicon
Garbled EDA [24]	3	Simulation	Pre-silicon (Netlist)
MP ℓ oC [25]	2	Simulation	Pre-silicon (Netlist)
Pythia [26]	2	Simulation	Pre-silicon (Netlist)
This Work	2	Formal Logic	Pre-silicon(RTL)

Verifiability & Confidentiality. As summarized in Table I, existing approaches either lack verification support [13], [14], [15], [16], rely on simulation-based verification [25], [26], or require a trusted third party [17], [24]. Whereas our work enables privacy-preserving formal verification for a wide range of IP designs in a two-party setting.

Various IP protection techniques, including watermarking, active metering, and logic locking [13], [14], [15], [16], as summarized in Table I, aim to deter IP misuse. However, these approaches primarily protect IP after trading or fabrication and provide no guarantees of verifiability. The IEEE 1735 standard [17] provides an encryption and management framework involving a trusted EDA tool vendor, yet attacks [18], [27] have exposed inherent weaknesses in its deployment within current commercial EDA tools. Garbled EDA [24] enables an IP user to obtain simulation results without revealing sensitive information such as the process design key, IP core, or EDA binaries, but it relies on a trusted EDA tool provider. MP ℓ oC [25] combines secure multiparty computation (MPC) and logic locking to support privacy-preserving IP simulation, while Pythia [26] allows the IP vendor to generate a zero-knowledge proof demonstrating the correctness of the computation to the IP user. As reflected in Table I, these methods are limited to simulation-based verification at the netlist level and are primarily evaluated on the ISCAS'85 benchmarks, leaving open questions regarding scalability and applicability to formal logic verification.

Traceability. Blockchain can enhance hardware security by tracking and authenticating ICs across supply chains. Islam et al. [28] used smart contracts to log IC identity, ownership, and PUF-derived data, while Chaudhary et al. [29] reduced on-chain storage by moving CRPs off-chain with on-chain references. ICToken [30] extends tracking from fabrication to end users via NFT tokenization [31] and logic locking. However, existing approaches primarily focus on post-fabrication traceability and provide limited coverage in earlier design

stages. Our work fills this gap by integrating soft IP traceability into a blockchain to enable secure and transparent lifecycle management.

IV. THREAT MODEL

BlindMarket operates in a zero-trust setting without relying on any trusted broker or verification provider. The system consists of two phases: (1) secure verification and (2) IP distribution, and its security relies solely on cryptographic protocols and on-chain records. During the secure verification phase, we adopt the semi-honest adversary model, where both parties follow the protocol but may attempt to infer additional information from the transcript. Under this model, the IP user learns only the final verification result and public metadata, while the IP vendor learns neither the user's selected IP index nor the property specification. A malicious party may abort the protocol or provide malformed inputs, which may affect availability or correctness. Preventing such behaviors would require malicious-secure computation with input-consistency checks, which is beyond the scope of this work. In the *IP distribution phase*, a dishonest IP user may actively deviate from the protocol to obtain license authorization without approval from the IP vendor fraudulently. Such adversaries can collude with compromised authorized users to forge a license for protected IP content.

V. BLINDMARKET SYSTEM DESIGN

A. BlindMarket Workflow

BlindMarket involves two parties: the IP user and the IP vendor. In practice, an IP vendor typically maintains a portfolio of IP designs available for licensing. We model this portfolio as a set of abstract specifications, each describing the functionality of a design without disclosing implementation details or internal structures. Based on these high-level descriptions, an IP user selects a target design to verify while keeping their selection and property of interest hidden from the vendor. One of the core challenges is to allow the user to verify the correctness of a selected design without revealing the design itself or the user's verification intent.

In addition to confidentiality and verifiability, BlindMarket enforces traceability after the IP core is delivered. Specifically, the vendor issues a usage certificate bound to the authorized user. This certificate enables the user to incorporate the IP into their design. BlindMarket ensures that even in the presence of collusion, an authorized user cannot transfer or replicate this certificate for the unauthorized user.

To achieve these goals, BlindMarket consists of four core phases: (1) IP and property preprocessing, where both the user and the vendor convert IP designs and property to appropriate formats; (2) oblivious design selection, where the user selects a design without learning its content or revealing their selection; (3) secure design verification, allowing the user to check correctness without accessing the full design; and (4) license authorization, where the vendor issues a certificate that binds the authorized user to the selected IP. Figure 1 summarizes the process.

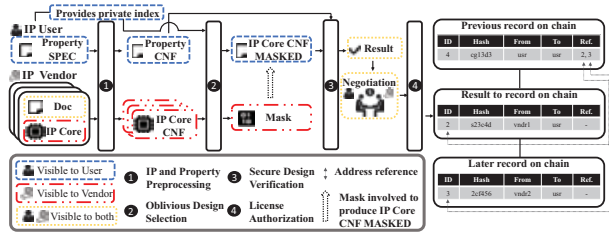


Fig. 1: End-to-end workflow of secure IP verification and licensing in BlindMarket, organized into four phases. (1) In the preprocessing phase, the vendor encodes each IP core into CNF and masks it using randomness, while the user encodes the verification property. (2) During oblivious design selection, the user retrieves the masked CNF of the selected design via 1-out-of- N OT, without revealing the selection to the vendor. (3) In the secure design verification phase, the vendor and user engage in a 2PC protocol to reconstruct the selected design and jointly verify it against the property using $hw\text{-}ppSAT$. (4) Finally, if verification succeeds, the vendor issues a certificate bound to the authorized user and selected IP, publishing it to the ledger for traceability.

Customized IP and property translations. For each commercial IP product, the vendor discloses a high-level functional specification and a set of *observable signals* to the IP user. The set of observable signals typically includes the primary I/O and a subset of internal signals. To support secure verification, the vendor locally converts the design into a Boolean formula and provides *semantic map* of the observable signals, which are detailed in Section V-B.

The IP vendor then reveals the *semantic map* to the IP user so that they can model the property as a Boolean formula using variables with the same semantics as the vendor does. The value of IP usually lies in the implementation of logical operations and interconnection, which is not shared in BlindMarket. To balance efficiency and confidentiality, BlindMarket provides the optimization approach for design pruning when the user is willing to reveal assertion signals. We detail the optimization in Section VI, and our evaluation results (see Section VII-A).

Oblivious design selection. The IP user selects a design of interest from the vendor's portfolio, while keeping the selection hidden to protect purchase intentions. This phase enables private design selection for subsequent verification without revealing the chosen design to either party.

After preprocessing, the Boolean formula of all designs in the vendor portfolio are obfuscated with the same randomness. The obfuscated formula is then transmitted via an oblivious transfer (OT) protocol. The way we obfuscate the formula is information-theoretically secure, and therefore ensures confidentiality of the design. Meanwhile, the security of OT guarantees the privacy of the IP users' choice and thus interests. The technical details of the protocol are presented in Section V-C.

Secure design verification. The secure design verification protocol is initialized after the user obtains the obfuscated design, as detailed in Section V-D. In this phase, the user provides the obfuscated Boolean formula of the design and

the Boolean formula of their property. At the same time, the vendor inputs the randomness he uses to mask all formulas. The protocol first demasks the design formula using input from two parties in 2PC. The demasked design and the property are then checked for satisfiability using a subsequent privacy-preserving SAT-solving protocol. The output of the solver is revealed to users, indicating that the design satisfies the expected property.

License authorization. The vendor registers each design in a ledger system using a unique identifier. To support a zero-trust framework, we implement the ledger using a blockchain-based infrastructure. Upon the completion of a sale, the vendor authenticates the transaction by appending a new entry to the ledger. During integration, the IP vendor uploads an additional record that references the previously recorded purchase. The structure of the record and operations are described in detail in Section V-F.

B. Customized IP and Property Translations

Existing translation tools built on hardware synthesis frameworks typically bind the design to a specific cell library and flatten it into gate-level netlists. Such flattening often removes high-level semantic structures, making it difficult to express verification properties that are naturally defined at the behavioral level. Moreover, complex operators (e.g., multiplexers) are decomposed into primitive logic gates, which introduces a large number of auxiliary variables and clauses during CNF generation. Since BlindMarket encodes the CNF as a pair of binary matrices $(P, N) \in \{0, 1\}^{n \times m}$, larger values of n and m directly increase the computational and communication overhead of the subsequent secure protocols. Therefore, we customize a word-level translation for IP designs and properties that maintains a small number of variables, and both users and vendors independently encode the design and properties into Boolean formulas based on a publicly agreed-upon bound.

1) Formula Generation of Design

On the IP vendor side, the design is encoded as a Boolean formula through the following steps. Our static analysis frontend (based on [32], [33]) takes the synthesizable RTL as input and emits a word-level intermediate representation, which is then translated into a quantifier-free bit-vector SMT formula (QF-BV) [34], where each variable is modelled as a *fixed-size bit vector*. For instance, in Example 1, the signal *counter*, defined starting at line 3, is translated into its corresponding SMT bit-vector representation beginning at line 10. The resulting SMT formula is then converted into CNF via *bit-blasting* [35] and *Tseytin transformation* [36]. Bit-blasting translates bit-vector operations into Boolean logic, while the Tseytin transformation encodes Boolean formulas in CNF using auxiliary variables without exponential growth.

Meanwhile, BlindMarket frontend allows the vendor to automatically construct a *semantic map* for the *observable signals*, which correspond to a small subset of design variables. The semantic map assigns each observable signal a unique literal, as shown in lines 15–16, where each bit of the

variables `counter` and `reset` is assigned a unique literal. This mapping enables the IP user to formulate properties in Section V-B2. The signals used in the control expressions defined as control signals are also tracked in the semantic map (such as `reset` in Example 1 line 2), which remains private and is weighted by its depth in its statement's abstract syntax tree (AST) [37]. We leverage this metric to accelerate the SAT solver, as detailed in Section V-E. Example 1 demonstrates how a simple counter updates `counter` based on previous value of control signal `reset` and `enable` symbolically. The SMT constraints in lines 10–13 specify an equivalent logical formula that encodes its behaviour, where we append $\#i$ to each signal to denote the timestamp. Specifically, `counter#2` denotes the symbolic value of `counter` at the second clock cycle and is updated by a nested ternary expression referencing `counter#1` from the first clock cycle. Furthermore, the vendor provides a semantic map that associates the 3-bit signal `counter#2` with the literal $\{\ell_4, \ell_3, \ell_2\}$ and the 1-bit signal `reset#1` with ℓ_1 , while also recording the depths of `reset#1` and `enable#1` as 1 and 2, respectively.

```

1  always @(posedge clk) begin
2      if (reset)
3          counter <= 3'b0; // Reset the counter to 0
4      else if (enable)
5          counter <= counter + 3'b10; // Increment the counter
6          by 2
7      else
8          counter <= counter + 3'b1; // Increment the counter
9          by 1
10     end
11 /* SMT Formula */
12 counter#2 <--> (ite (|reset#1| #b1)
13     #x00(ite (|enable#1| #b1)
14         (bvadd |counter#1| #x02)
15         (bvadd |counter#1| #x01)))
16 /* Semantic map */
17 {reset#1 : 1, counter#2[0] : 2,
18   counter#2[1] : 3, counter#2[2] : 4}
19 /*Ctrl Depth*/
20 {reset#1 : 1, enable#1 : 2}

```

EXAMPLE 1: Incrementing counter with synchronous reset

2) Boolean Encodings of Properties

The IP user then applies the *semantic map* to encode its property as a Boolean formula. BlindMarket focuses on properties composed of three common primitives, namely NOI, OI, and Concat, each of which can be translated into basic Boolean logic. NOI is represented by $\neg a_i \vee b_{i+1}$, indicating that if signal a holds at time step i , then b must hold at the next time step $i + 1$. In contrast, OI is encoded as $\neg a_i \vee b_i$, requiring that b hold in the same cycle as a . Concat expresses a sequential pattern and is encoded as $a_i \wedge b_{i+n}$, meaning that signal a must hold at time i , and signal b must hold exactly n cycles later.

In Example 1, the user obtains the semantic map from the IP vendor, and a property such as `reset#1` $\rightarrow$ (`counter#2` = 0), can be used to check whether `counter` is reset to zero whenever `reset` is asserted. Under the semantic map, this requirement is translated into the Boolean formula $\neg \ell_1 \vee (\neg \ell_2 \wedge \neg \ell_3 \wedge \neg \ell_4)$.

C. Oblivious Design Selection

Oblivious selection is performed prior to secure verification to prevent the IP vendor from knowing the IP design of the user's interests. In standard verification workflows, the user must disclose the target they wish to verify to initiate the process. Such disclosure can be problematic, particularly when the vendor may exploit knowledge of user preferences to manipulate pricing or restrict offerings.

BlindMarket leverages oblivious transfer (OT) to address this challenge. OT enables an IP user (the receiver) to obtain one design from a set offered by the vendor (the sender) without revealing which design is selected. However, directly applying OT will compromise confidentiality by exposing the full IP design to the user. To preserve design privacy while enabling verification, the OT procedure must be integrated with the privacy-preserving framework. This integration requires the SAT solver to operate in a white-box manner, as detailed in Section V-D.

Initially, the CNF of each design ϕ_k in the design portfolio Φ is encoded as a pair of binary matrices (P_k, N_k) on the vendor side. Then each pair (P_k, N_k) is masked using a shared pair of random binary matrices (R_P, R_N) . We require (R_P, R_N) to be refreshed for every verification session. Otherwise, colluding users could XOR masked instances obtained from different runs to cancel the masks and infer structural relationships between protected designs. Specifically, the masked representation is computed as $(\tilde{P}_k, \tilde{N}_k) = (P_k \oplus R_P, N_k \oplus R_N)$. The collection of masked matrices $(\tilde{P}_k, \tilde{N}_k)$ is used as the vendor's input to the oblivious transfer (OT) protocol.

On the IP user side, the user selects an index i corresponding to the desired design based on the public metadata provided in the vendor's design portfolio. This index serves as the receiver's input to a 1-out-of- N OT protocol. During the OT execution, the two parties jointly run the protocol such that, at the conclusion, the user obtains the pair of masked matrices $(\tilde{P}_i, \tilde{N}_i)$ corresponding to the selected design. The IP vendor retains the random masking matrices (R_P, R_N) .

D. Secure Design Verification

Secure design verification aims to determine whether a proprietary design satisfies a user-specified property, without revealing the design to the user or disclosing the property to the vendor. However, applying ppSAT directly is insufficient in our setting as it requires both parties to agree on a common verification target in the setup. This synchronization step reveals the IP user's selection to the vendor.

To resolve this issue, after the oblivious design selection protocol, the user obtains a pair of obfuscated matrices $(\tilde{P}_i, \tilde{N}_i)$ corresponding to the selection index i . Following this, BlindMarket initiates a secure 2PC protocol in which the user inputs the masked design matrices and the property φ , while the vendor inputs the retained random masking shares (R_P, R_N) . The design matrices are then reconstructed by demasking in 2PC:

$$(P_i, N_i) = (\tilde{P}_i \oplus R_P, \tilde{N}_i \oplus R_N). \quad (1)$$

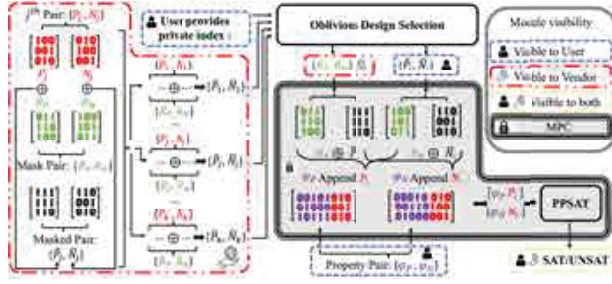


Fig. 2: Secure hardware design verification workflow in BlindMarket. The vendor masks each CNF-encoded design using random matrices, and the user selects a target design via oblivious transfer (OT), receiving only the masked version. The vendor and user then run a 2PC protocol to reconstruct the selected design and combine it with the user's property to form $\psi = \phi_i \wedge \neg\varphi$. The satisfiability of ψ is evaluated using hw-ppSAT under 2PC, revealing only the final result to both parties while preserving mutual confidentiality.

This demasked pair of matrices encodes the selected design ϕ_i . The verification task is then formulated as the satisfiability of the Boolean formula $\psi = \phi_i \wedge \neg\varphi$, which is secret-shared between the two parties within the 2PC. The ppSAT.Solve protocol is then invoked to evaluate the satisfiability of ψ and the satisfiability result is revealed to both parties at the end. Throughout this process, the IP design remains hidden from the user, and the verification property remains unknown to the vendor, ensuring mutual confidentiality.

E. hw-ppSAT

Numerous techniques have been proposed in the formal method community to improve solver efficiency through conflict analysis [38], [39], structural analysis [40], [41], etc. However, lifting these techniques into a privacy-preserving setting remains an open challenge [20]. Although ppSAT.Solve adapts DPLL as its base routine with various branching heuristics for acceleration, as a general-purpose SAT solver, it doesn't exploit structural information derived from hardware designs. Prior work has shown that incorporating structural information into variable ordering can substantially improve solving efficiency in SMT solving [42]. To enable ppSAT.Solve to benefit from hardware semantics under encryption, BlindMarket augments the ppSAT.Solve with a hardware control-flow guided heuristic, referred to as hw-ppSAT.

1) Control-flow Guided Heuristic

Formally, let $\mathcal{V}$ denote the set of all variables in the Boolean formula generated from a hardware IP design. Our heuristic defines a *partial order* relation $\prec$ over $\mathcal{V}$. For any two variables $x, y \in \mathcal{V}$, if $y \prec x$, then x should always be justified before y . That is, when both x and y are undecided, and the solver must make a branching decision, it will always assign a value to x before considering y . The order captures the three principles as follows:

Principle 1: Variables encoding control signals are prioritized for branching over all other signals. In hardware designs, a control signal may govern multiple data paths. Once it

is assigned, many downstream data values become either constants or irrelevant to the evaluation, allowing the solver to prune a large portion of the search space.

Example. Figure 3a depicts the gate structure for $S_1 \wedge S_2$, where the values of S_1 and S_2 depend on the signal ctrl1. When ctrl1 is true, S_1 and S_2 propagate the values of a and c , respectively; otherwise, b and d are passed. Without prioritizing the control signal ctrl1, verifying $S_1 \wedge S_2$ requires evaluating four possible cases: $a \wedge c$, $a \wedge d$, $b \wedge c$, and $b \wedge d$. Two of these lead to conflicts. However, when ctrl1 is fixed, the conflicting cases are eliminated, leaving only two scenarios to verify.

Formally, let $\mathcal{V}_C \subseteq \mathcal{V}$ be the set of *control signals*, and $\mathcal{V}_D = \mathcal{V} \setminus \mathcal{V}_C$ be the set of *non-control signals*. $\forall c \in \mathcal{V}_C, \forall d \in \mathcal{V}_D : d \prec c$.

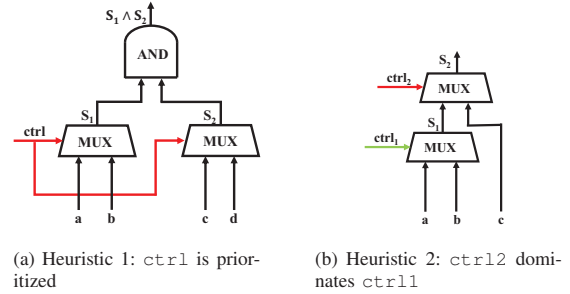


Fig. 3: Illustration of two signal prioritization principles. In (a), the control signal ctrl1 is prioritized during branching, since assigning ctrl1 to false renders the values of a and c irrelevant to the satisfiability of the encoded design. In (b), for two control signals ctrl1 and ctrl12, ctrl12 is prioritized, because once ctrl12 is assigned false, ctrl1 no longer influences the evaluation.

Principle 2: Within a single time frame, outer control signals dominate inner control signals. In a nested structure, an outer control signal governs the scope that encloses the logic of inner signals. Evaluating an inner control before its corresponding outer signal leads to unnecessary computation, because if the outer control disables that block, the inner assignment becomes irrelevant and its effect is masked by the outer logic.

Example. If ctrl12 is false in Figure 3b, the value of S_2 is determined by c , while ctrl1 does not affect the result due to the blocked data flow from a and b . Therefore, the interpretation of ctrl1 is dominated by ctrl12.

Principle 3: Across unrolled time frames, control signals at later time stages dominate those at earlier time stages. When a design is unrolled, signals at earlier time stages are structurally nested within those of later time stages, making them analogous to inner controls. Following the same prioritisation rationale as Principle 2 and classic tuning practices [43], the solver should evaluate later-stage control signals before earlier ones. Formally, for signals $c_i \in \mathcal{V}_C$ at time i and $c_j \in \mathcal{V}_C$ at time j , if $j > i$, then $c_i \prec c_j$, meaning that c_j should be interpreted first.

Figure 4 illustrates the unrolled gate structure of the counter signal from Example 1. In this case, the symbolic

value of count_3 is updated from count_1 through the unrolled circuit. To verify a property involving count_3 , the solver should justify reset_2 and enable_2 before reset_1 and enable_1 .

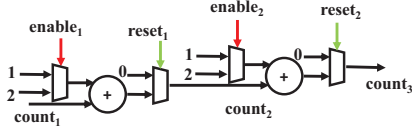


Fig. 4: When the circuit is unrolled, later-stage control signals dominate earlier-stage ones. Thus enable_2 dominates reset_1 . Based on Principle 2, reset_1 dominates enable_1 . Consequently, the priority chain becomes $\text{enable}_2 \succ \text{reset}_1 \succ \text{enable}_1$.

In general, determining the partial order set of control signals from a CNF is non-trivial because structural information disappears once the design is encoded into Boolean clauses. BlindMarket allows the IP vendor to determine the partial-order set locally during preprocessing (see Section V-B1), without leaking any sensitive design information to the IP user.

Specifically, control signals are extracted through AST traversal. The prioritisation of control signals is derived from dataflow analysis using Depth First Search (DFS). For signals $c_1, c_2 \in \mathcal{V}_C$ within a single time frame, $c_1 \prec c_2$ if $\text{Dep}(c_1) > \text{Dep}(c_2)$, where $\text{Dep} : \mathcal{V}_C \rightarrow \mathbb{N}$ maps each signal to its depth obtained by the DFS from dataflow. Both extraction and prioritisation can be performed based on RTL static analysis tools such as [32], [33].

2) hw-ppSAT Implementation

Unlike `ppSAT.Solve`, `hw-ppSAT` requires the IP user to provide a partial-order set $\prec_C$ as an additional input, which is generated on the IP vendor side.

To prevent information leakage, the heuristic set associated with each IP design is retrieved through the same Oblivious Design Selection protocol described in Section V-C. Concretely, the IP vendor generates a heuristic set $\prec_C^k$ for each design ϕ_k during preprocessing and includes it in the message vector used in the OT protocol. The IP user then retrieves the corresponding heuristic set using a 1-out-of- N OT via

$$(\prec_C^i, \perp) \leftarrow \text{OT}(i, \{\prec_C^1, \prec_C^2, \dots, \prec_C^N\}).$$

By the security guarantee of OT, the user learns only $\prec_C^i$ and obtains no information about $\{\prec_C^j\}_{j \neq i}$, while the vendor learns nothing about the user's selection i . Furthermore, since $\prec_C$ only specifies a partial ordering of variables for branching decisions, it does not reveal structural information about the underlying circuit.

We formally define the `hw-ppSAT` decision step in Algorithm 1, which implements a hybrid method that combines DLIS (the best in `ppSAT.Solve`) with a control-flow-guided heuristic. In this approach, `hw-ppSAT` selects a control literal available from $\prec_C$ for the branching heuristic. If all control literals have been evaluated and the formula remains unsolved, the solver defaults to the DLIS strategy for the remaining non-control literal.

Algorithm 1 Decision Step in hw-ppSAT

Input: $L = \{\ell_1, \dots, \ell_{2n}\}$, $C = \{C_1, \dots, C_m\}$, $\prec_C$

Output: $d \in [1..n] \times \{0, 1\}$

```

1:  $d \leftarrow (\perp, \perp)$ ,  $d' \leftarrow (\perp, \perp)$   $\triangleright d$ : decision;  $d'$ : fallback
   decision
2: for  $i \in \prec_C$  do
3:    $s_P \leftarrow \perp$ ,  $s_N \leftarrow \perp$   $\triangleright$  Check literal polarity in clauses
4:   for  $j \leftarrow 1$  to  $m$  do
5:      $s_P \leftarrow s_P \vee C_j.\text{contain}(\ell_i)$ 
6:      $s_N \leftarrow s_N \vee C_j.\text{contain}(\neg \ell_i)$ 
7:   end for
8:    $d \leftarrow s_N ? (i, 0) : d$ 
9:    $d \leftarrow s_P ? (i, 1) : d$ 
10: end for
11: for  $i \in L \setminus \prec_C$  do  $\triangleright$  Fallback to standard DLIS heuristic
   for non-control signals
12:    $d' \leftarrow \text{DLIS}(C, i)$ 
13: end for
14:  $d \leftarrow (d \neq (\perp, \perp)) ? d : d'$   $\triangleright$  Use heuristic decision;
   otherwise, fallback

```

TABLE II: Soft IP metadata in each record.

Attribute	Description	Size
ID	Unique IP identifier	32B
Hash	Hash of source code	32B
From	Vendor's address	20B
To	User's address	20B
Reference	Integrated IDs (up to 5)	32B $\times$ 5
Total		264B

F. License Authorization

To ensure traceability and transparency throughout the lifecycle of a soft IP, its record is published to the distributed ledger. This record contains the IP's ownership information, transaction details, and references to its integrated components. The format of the records is detailed in Table II. However, these ledger-based mechanisms do not physically prevent redistribution outside the marketplace. Redistribution risks can be mitigated by integrating complementary protection techniques such as hardware locking [13], active metering [14], and watermarking or fingerprinting methods [16], [15], where the buyer's ownership information can serve as a unique identifier embedded into the distributed IP instance.

VI. DESIGN PRUNING

Encoding a design as a pair of binary matrices $(P, N) \in \{0, 1\}^{n \times m}$ for property verification can sometimes incur unnecessary cryptographic overhead, as not all variables and clauses derived from the design are relevant to the property. To reduce the matrices' size, design pruning is applied before oblivious IP selection. As described in Section V-B1, each design statement is translated into an SMT representation. Given a specific property, the IP vendor can perform a Cone-of-Influence reduction [44] through backtracking the transitive fan-in from the property signal and retaining only the property-

relevant parts of the design to minimize m and n . SAT-based model checking has proved that COI reduction is particularly effective for control-dominated designs where the property depends on a small subset of signals, but provides limited reduction for datapath-intensive circuits where dependencies rapidly propagate through the design [45], [46]. In Section VII-A, our experimental results further confirm this observation.

Moreover, pruning is performed offline on the vendor side based on the user-provided property signal. Although the semantics of the property remain private to the user, the signal itself must be revealed to the vendor for COI analysis. To mitigate this disclosure, users may first submit pruning requests for multiple candidate designs as decoys to hide the actual verification target and then proceed with verification for the selected design.

VII. EXPERIMENT

Testbed. We implement OT and 2PC components (oblivious design selection and secure verification) of the framework using the semi-honest 2PC library of the EMP-toolkit [47]. We build our static analysis frontend on top of PyVerilog [32] and apply Z3 [48] to generate Boolean formula for both the designs and the properties. The experiment is conducted on an AWS `z1d.3xlarge` with 12 vCPUs and 96 GB memory.

Benchmarks. Table III lists all benchmark descriptions and detailed assertions. We verify bounded *assert properties* and *cover properties* on ANSI-C [49], OpenCores [50] and Trusthub [51]. ANSI includes the corresponding assertions within the code, and Trusthub, where chip-level designs are annotated with known vulnerabilities that can be expressed as *cover property*. For example, we verify constraints on output signals and state transitions to ensure correct sequential behaviour in the finite state machines `b01` and `b02`. The `Rrobin` design serves as a round-robin arbiter, where mutual exclusion is enforced by asserting that two acknowledge outputs should never be high simultaneously. The `adbg` module, a JTAG-compliant TAP controller, includes a password check mechanism, and we verify that no invalid input can bypass this check to set `passchk = 1`. In the Trusthub benchmark, `RS232_T700` verifies whether an input sequence causes the transmitter to enter a Denial-of-Service (DoS) state. In `memctrl_T100`, we check whether writing `0x77` in the CSR register results in the flash sleep bit (`fs`) being erroneously set to 1. Lastly, `wb_conmax_T300` checks whether any crafted input pattern allows a master to alter its destination slave unexpectedly.

A. Performance Evaluation

We highlight two types of performance improvements in this evaluation: (1) time reductions due to design pruning, and (2) improvements from `hw-ppsSAT`. For 13 IPs and their corresponding 26 (pruned) design pairs, 7 out of 13 IPs benefit directly from design pruning, achieving noticeable reductions in both variable and clause counts. In addition, 17 out of 26 instances (13 design pairs) demonstrate improved verification time with our control-flow-guided heuristics, as shown in Figure 5. Compared to the total solving time,

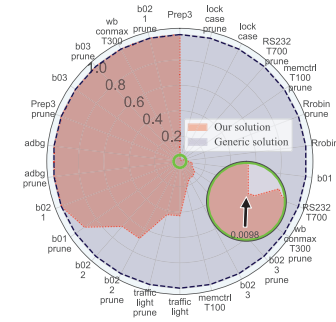


Fig. 5: Illustrates the performance ratio between (generic)ppsSAT and (our) `hw-ppsSAT` across various (pruned) benchmarks. Each segment represents a specific design, with the outer boundary (ratio = 1.0) indicating the baseline solving time of `ppsSAT`. The shaded region corresponds to `hw-ppsSAT`; a smaller area indicates a greater speedup. As shown, most designs experience a substantial reduction in solving time, with normalized ratios significantly below 1.0. In the best-case scenario, our method achieves a remarkable speedup with a ratio as low as 0.0098.

the overhead introduced by the IP selection and demasking phases is negligible—typically accounting for less than a few percent of the overall runtime. This indicates that the main performance bottleneck lies in the SAT solving stage, where our heuristics yield the most substantial speedup. Although the overall verification time can still be long for large and complex IPs, this cost is one-time and occurs only during the initial verification phase before the IP transaction.

Evaluation of Design Pruning We first evaluate the impact of pruning with `ppsSAT`. Table IV also reports the corresponding pruning-induced improvements for `hw-ppsSAT`. Since pruning is solver-agnostic and fundamentally reduces the size of the input matrices processed by the solver, its impact is similar for both solvers, and we therefore omit a separate discussion.

As shown by the highlighted entries in Table IV, `ppsSAT` achieves significantly faster performance on the pruned designs in 8 out of 13 benchmark pairs. This improvement is driven by the reduced formula size, which lowers both the overhead of oblivious design selection (OT time) and the 2PC workload (Solving time).

In particular, benchmarks such as `b01` and `Prep3` contain extraneous state machine logic that is not exercised by the property, allowing pruning to reduce the SAT solving time from 51853.60s to 2667.76s in `b01`, and from 20561.45s to 15176.60s in `Prep3`. The large IPs such as `memctrl_T100` and `wb_conmax_T300` with multiple control/data paths achieve orders-of-magnitude speedups. For `memctrl_T100`, both cases reach a timeout, while the timeout threshold is reduced from 14 million seconds to 24 thousand seconds. In the case of `RS232_T700`, although the design initially contains 1487 variables, only a subset related to the denial of service vulnerability is relevant, allowing a drastic formula reduction to just 225 variables and a run-time drop from 163885s to 31603.80s. In contrast, smaller designs such as `b02_2`, `b03`, `Rrobin`, and `lock_case` exhibit little to no performance change, as they are single-purpose IPs with

TABLE III: Design, Line of Code, Abstract Description and Assertion. We expect UNSAT for the negation of the assertion property conjoined with the design, and SAT for the counterexample derived from the conjunction of the cover property and the design.

Benchmark[Source]	LoC	Description	SVA Assertion
Rrobin [49]	59	round robin arbiter	assert (ack0==0 ack1==0)
b01 [49]	111	8-state Mealy FSM	assert (OVERFLW==1 -> ##2 (OVERFLW==0))
b02_1 [49]	81	7-state Moore FSM	assert (stato==D -> ##2 (stato==B))
b02_2 [49]	81	7-state Moore FSM	assert (U==1 -> ##1 U==0)
b02_3 [49]	81	7-state Moore FSM	assert (U==1 -> stato==B)
b03 [49]	119	4-request arbiter	assert (GRANT_O==0 GRANT_O==8 GRANT_O==4 GRANT_O==2 GRANT_O==1)
traffic_light [49]	52	3-state FSM with counter	cover (reset##5 (Light_Sign == YELLOW_LIGHT))
lock_case [50]	25	4-state Moore FSM	cover (unlock == 1)
Prep3 [50]	122	8-state Mealy FSM	assert (next_out == 8'h20 -> ##1 (next_out == 8'h11 next_out == 8'h30))
adbg [50]	531	JTAG controller	cover (next_passchk == 1)
memctrl_T100 [51]	1786	memory controller	cover (csr_r[2]==0 && fs==1)
RS232_T700 [51]	231	uart transmitter	cover (state_DataSend==7)
wb_conmax_T300 [51]	7626	general round robin arbiter	cover (slv_sel != wb_addr_i[31:28])

TABLE IV: We summarize the end-to-end execution time of BlindMarket for 13 IP designs in both their original and pruned forms. Var/Cls denotes the number of variables and clauses in the verification formula ψ . Design Size represents the size of the encoded design transferred during OT, measured in 128-bit blocks. OT (1/32) reports the execution time of 1-out-of-32 OT. Demask measures the time required to reconstruct the selected design within the 2PC (Equation 1). For the SAT solving stage, DLIS records the number of giant steps taken under the standard ppSAT DLIS heuristic, and ppSAT reports the corresponding solving time. Similarly, Ctrl denotes the number of giant steps taken under hw-ppSAT, while hw-ppSAT reports the solving time using the control-flow-guided heuristic. The results show that combining design pruning with hw-ppSAT achieves faster verification times on 12 of 13 benchmarks. In the table, yellow cells represent time improvements from design pruning, and green cells represent time improvements from heuristic guidance.

Design	Bound	Var	Cls	Design Size(blocks)	OT (1/32)(ms)	Demask(s)	DLIS	ppSAT(s)	Ctrl	hw-ppSAT(s)
Prep3	3	876	2617	35266	146.37	80.06	911	20561.45	911	20561.45
Prep3_prune	3	771	2392	32116	142.23	63.56	818	15176.60	818	15176.60
RS232_T700	5	1487	5200	124512	513.56	290.69	20000*	1638850	2000	163885
RS232_T700_prune	5	225	708	2784	12.33	5.99	20000*	31603.80	222	350.80
Rrobin	6	154	387	1512	4.97	2.03	11561	6704.32	521	302.13
Rrobin_prune	6	154	387	1512	4.25	2.04	11561	6675.69	501	289.29
adbg	4	2186	13844	498168	2115.35	1148.19	1100	356573.80	1100	356573.80
adbg_prune	4	2103	13756	467500	1877.73	1094.98	1102	346138.20	1102	346138.20
b01	4	256	994	3952	16.81	9.52	20000*	51853.60	1978	5128.32
b01_prune	4	211	762	3024	12.08	6.28	1686	2667.76	1539	2435.16
b02_1	3	163	565	2132	9.28	3.33	108	97.80	108	97.80
b02_1_prune	3	144	478	1784	9.41	2.47	97	65.73	97	65.73
b02_2	2	118	395	720	3.80	1.64	298	138.30	204	94.67
b02_2_prune	2	118	395	720	3.17	1.62	298	140.27	204	96.02
b02_3	3	160	570	2132	8.77	3.24	1153	1080.95	178	166.88
b02_3_prune	3	160	570	2132	8.37	3.39	1153	1055.94	178	163.02
b03	7	1289	5026	109670	436.25	223.30	240	15413.88	240	15413.88
b03_prune	7	1289	5026	109670	430.14	226.00	240	15234.12	240	15234.12
lock_case	4	176	450	1728	9.43	2.87	20000*	15844.80	196	155.28
lock_case_prune	4	176	450	1728	7.36	2.92	20000*	15661.34	196	153.48
memctrl_T100	1	5183	12943	1059194	4469.61	2450.10	20000*	14628460	4365	3192661.40
memctrl_T100_prune	1	268	458	2592	10.66	4.44	20000*	24927	236	294.14
traffic_light	5	590	2028	20190	91.33	41.60	2376	28052.48	1017	12007.31
traffic_light_prune	5	550	1948	19390	77.22	36.82	2142	22330.56	925	9643.22
wb_conmax_*	1	2781	5705	246972	1045.03	582.29	20000*	3264500	20000*	3264500
wb_conmax*_prune	1	296	421	1974	8.24	3.63	2745	3391.09	384	474.38

* timeout when reaching 20000 giant steps;

compact state machines that leave no room for structural reduction.

Evaluation of hw-ppSAT The integration of control-flow-guided heuristics in hw-ppSAT delivers a clear performance advantage across a majority of benchmarks. As seen in Table IV, 17 out of 26 instances exhibit faster solving times when switching from the default decision heuristic (ppSAT) to the control-aware version (hw-ppSAT). This speedup is further illustrated in Figure 5. This improvement is primarily driven by the reduced number of giant steps. For instance, in b01_prune, hw-ppSAT

reduces runtime from 2667.76s to 2435.16s by cutting 147 steps, while in RS232_T700_prune, the step count drops from 20000(timeout) to 222, resulting in a total runtime reduction from 31603.80s to 350.80s. Similarly, the traffic_light_prune benchmark improves from 22330.56s to 9643.22s, and memctrl_T100_prune from 24927s to 294.14s.

In 9 out of 26 instances, no runtime improvement is observed. Comparing the Ctrl and DLIS configurations shows that the proposed heuristic neither improves nor degrades performance in these cases. This behavior is inherent to SAT

solving: heuristics are only invoked when the solver must choose the next undecided literal. For these instances, the solver can resolve the formula through propagation without making heuristic-driven decisions, and thus the heuristic is never applied during the solving process.

B. Communication Overhead

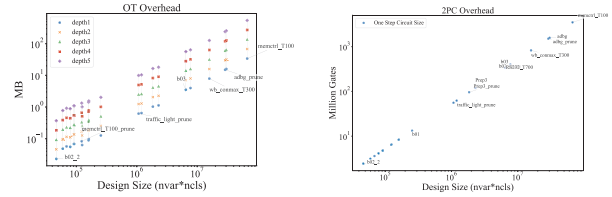
Figure 6a illustrates the communication overhead introduced by the 1-out-of- 2^{depth} OT protocol. The x-axis represents the design size measured as the product of the number of variables and clauses ($n_{\text{var}} \times n_{\text{cls}}$), while the y-axis reports the total OT cost in MB. Each point corresponds to a verification instance. As shown in Figure 6a, the OT communication overhead increases approximately linearly with the design size. As discussed in Section II-B, each clause in the CNF-encoded design is represented as a pair of binary vectors of length n . Consequently, the OT overhead scales with $n_{\text{var}} \times n_{\text{cls}}$.

Furthermore, the OT overhead grows approximately linearly with 2^{depth} . In a 1-out-of- 2^{depth} OT protocol, increasing depth enlarges the portfolio size $N = 2^{\text{depth}}$. As a result, more masked candidate messages must be communicated during the transfer. For the smallest benchmark, b02, with $\text{depth} = 1$, the OT communication overhead is approximately 0.022 MB, whereas for the largest benchmark, mem_ctrl, with $\text{depth} = 5$, the OT overhead reaches approximately 517 MB.

Figure 6b shows the cryptographic overhead of 2PC. This overhead primarily arises from hw-ppSAT, which adapts the DPLL algorithm as its core routine and compiles each *giant step* into the garbled circuit, where the communication overhead scales linearly with the number of AND gates (about 32 bytes per gate). Thus, the total communication cost is the per-step circuit cost multiplied by the number of *giant steps*. Similar to the OT protocol, the circuit size grows approximately linearly with the design size. Larger designs produce larger CNF encodings and require more evaluation during verification. For the largest design, mem_ctrl, the circuit size can reach around 3 billion gates per step, making secure verification impractical without optimization. After applying design pruning, we reduce the communication overhead to about 6 million gates per step. Furthermore, using the control-flow-guided heuristic, the number of giant steps is reduced from more than 20,000 to 236, making secure verification feasible.

C. Comparison with Non-private Baseline

We implement hw-ppSAT and ppSAT in Python and include the non-private baseline experiment in Figure 7. To ensure consistency with Table IV, the timeout is set to 20,000 giant steps, identical to that used in Table IV. Figure 7 shows that reaching 20,000 giant steps corresponds to approximately 100 seconds for the largest timed-out design. The results also indicate that solver runtime is largely dominated by the number of executed giant steps, as the baseline implementations follow the same DPLL core routine. The most time-consuming benchmark, memctrl_T100, finishes verification in 37.9 seconds, corresponding to an approximately 8.4×10^4 slowdown compared with the non-private baseline. In contrast,



(a) Communication overhead of the 1-out-of- 2^{depth} OT protocol versus design size ($n_{\text{var}} \times n_{\text{cls}}$). The overhead scales approximately linearly with design size and increases with portfolio size 2^{depth} . (b) 2PC communication overhead per giant step measured by the garbled circuit size. The circuit size grows approximately linearly with the design size ($n_{\text{var}} \times n_{\text{cls}}$)

Fig. 6

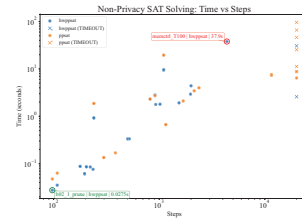


Fig. 7: Runtime of the non-private baseline implementation of hw-ppSAT without 2PC. The results indicate that solver runtime is largely dominated by the number of executed steps.

b02_1_prune finishes within 0.02 seconds, incurring an overhead of about 3×10^3 . As expected, the non-private verification implementation is significantly faster than the privacy-preserving one due to the additional cryptographic operations. Moreover, the slowdown is not uniform across benchmarks and primarily depends on the design size, as larger designs incur substantially higher overhead in GC-based 2PC computation.

VIII. CONCLUSION & DISCUSSION

This paper introduces BlindMarket, the first end-to-end framework for addressing privacy concerns in hardware formal verification and ownership traceability after IP trading. We propose a zero-trust model that eliminates the need for a trusted third party while enabling formal logic verification, a capability not addressed by prior industrial or academic work. By leveraging 2PC and OT, we extend the state-of-the-art privacy-preserving SAT solver ppSAT to the hardware domain (hw-ppSAT) and apply optimizations that make IP-level verification practical. A practical challenge of BlindMarket lies in its interactive protocol, where stages such as oblivious design selection and secure SAT solving require synchronized message exchanges between the IP vendor and the IP user, introducing additional latency compared with offline verification. Despite this overhead, the framework remains compatible with existing post-licensing IP protection solutions and can be integrated into current hardware supply-chain ecosystems.

IX. ACKNOWLEDGMENT

Portions of this work were supported by the National Science Foundation (2340949, 2419880, and 2304533).

REFERENCES

- [1] S. I. Association *et al.*, “Global semiconductor sales, units shipped reach all-time highs in 2021 as industry ramps up production amid shortage,” *Haettu*, vol. 16, p. 2022, 2022.
- [2] N. Fern, I. San, and K.-T. T. Cheng, “Detecting hardware trojans in unspecified functionality through solving satisfiability problems,” in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.
- [3] K. Ceesay-Seitz, F. Solt, and K. Razavi, “ μ cfi: Formal verification of microarchitectural control-flow integrity,” in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 213–227.
- [4] X. Guo, R. G. Dutta, J. He, and Y. Jin, “Pch framework for ip runtime security verification,” in *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2017, pp. 79–84.
- [5] R. Zhang, C. Deuschbein, P. Huang, and C. Sturton, “End-to-end automated exploit generation for validating the security of processor designs,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 815–827.
- [6] X. Meng, S. Kundu, A. K. Kanuparthi, and K. Basu, “Rtl-contest: Concolic testing on rtl for detecting security vulnerabilities,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 466–477, 2021.
- [7] Q. Zhang, L. Liu, Y. Yuan, Z. Zhang, J. He, Y. Gao, Y. Li, X. Guo, and Y. Zhao, “A gate-level information leakage detection framework of sequential circuit using z3,” *Electronics*, vol. 11, no. 24, p. 4216, 2022.
- [8] Reuters Staff, “Former russian asml employee set for court hearing in alleged ip theft case,” *Reuters*, 2024, accessed: 2025-04-02. [Online]. Available: <https://www.reuters.com/technology/former-russian-asml-employee-set-court-hearing-alleged-ip-theft-case-2024-12-09/>
- [9] R. Iyengar, “Tesla sues former employee for allegedly stealing autopilot source code for chinese rival,” *CNN Business*, 2019, accessed: 2025-04-02. [Online]. Available: <https://www.cnn.com/2019/03/22/tech/tesla-xiaopeng-motors-lawsuit/index.html>
- [10] D. Halbert, “Intellectual property theft and national security: Agendas and assumptions,” *The Information Society*, vol. 32, no. 4, pp. 256–268, 2016.
- [11] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP security and trust*. Springer, 2017.
- [12] S. Ray, “Samsung Bans ChatGPT Among Employees After Sensitive Code Leak,” May 2 2023, *Forbes*.
- [13] A. B. Kahng and *et al.*, “Constraint-based Watermarking Techniques for Design IP Protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1236–1252, 2001.
- [14] Y. Alkabani and F. Koushanfar, “Active Hardware Metering for Intellectual Property Protection and Security,” in *USENIX Security Symposium (USENIX Security ’07)*, vol. 20, 2007, pp. 1–20.
- [15] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, “Advances in Logic Locking: Past, Present, and Prospects,” *IACR Cryptol. ePrint Arch.*, 2022.
- [16] R. S. Chakraborty and S. Bhunia, “Harpoon: An obfuscation-based soc design methodology for hardware protection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.
- [17] “IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP),” *IEEE Std 1735-2014 (Incorporates IEEE Std 1735-2014/Cor 1-2015)*, pp. 1–90, 2015.
- [18] J. Speith, F. Schweins, M. Ender, M. Fyrbiak, A. May, and C. Paar, “How Not to Protect Your IP—An Industry-Wide Break of IEEE 1735 Implementations,” in *2022 IEEE Symposium on Security and Privacy (Oakland ’22)*. IEEE, 2022, pp. 1656–1671.
- [19] M. Gebeily, J. Pearson, and D. Gauthier-Villars, “How Israel’s bulky pager fooled Hezbollah,” <https://www.reuters.com/graphics/ISRAEL-PALESTINIANS/HEZBOLLAH-PAGERS/mopawkkwjpa/>. Accessed April 3rd, 2025.
- [20] N. Luo, S. Judson, T. Antonopoulos, R. Piskac, and X. Wang, “ppSAT: Towards Two-Party Private SAT Solving,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2983–3000.
- [21] A. Dobis, “Formal verification of hardware using mlir,” Master’s thesis, ETH Zurich, 2024.
- [22] S. Vijayaraghavan and M. Ramanathan, *A Practical Guide for SystemVerilog Assertions*. Springer Science Business Media, 2005.
- [23] A. C.-C. Yao, “How to generate and exchange secrets,” in *Annual Symposium on Foundations of Computer Science*. IEEE, 1986.
- [24] M. Hashemi, S. Roy, F. Ganji, and D. Forte, “Garbled EDA: Privacy Preserving Electronic Design Automation,” in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD ’22)*, 2022, pp. 1–9, to appear.
- [25] D. Mouris, C. Gouert, and N. G. Tsoutsos, “MPloC: Privacy-Preserving IP Verification using Logic Locking and Secure Multiparty Computation,” *Cryptology ePrint Archive*, 2023.
- [26] D. Mouris and N. Georgios Tsoutsos, “Pythia: Intellectual Property Verification in Zero-Knowledge,” in *2020 57th ACM/IEEE Design Automation Conference (DAC ’20)*, 2020, pp. 1–6.
- [27] A. Chhotaray, A. Nahiyan, T. Shrimpton, D. Forte, and M. Tehranipoor, “Standardizing Bad Cryptographic Practice: A Teardown of the IEEE Standard for Protecting Electronic-Design Intellectual Property,” in *ACM SIGSAC CCS ’17*, 2017, pp. 1533–1546.
- [28] M. N. Islam and S. Kundu, “Enabling ic traceability via blockchain pegged to embedded puf,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 3, pp. 1–23, 2019.
- [29] C. K. Chaudhary, U. Chatterjee, and D. Mukhopadhyay, “Auto-pufchain: an automated interaction tool for pufs and blockchain in electronic supply chain,” in *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2021, pp. 1–4.
- [30] S. Balla, Y. Zhao, and F. Koushanfar, “Ictoken: An nft for hardware ip protection,” *arXiv preprint arXiv:2412.06726*, 2024.
- [31] Q. Wang, R. Li, Q. Wang, and S. Chen, “Non-fungible token (nft): Overview, evaluation, opportunities and challenges,” *arXiv preprint arXiv:2105.07447*, 2021.
- [32] S. Takamada-Yamazaki, “Pyverilog: A Python-based Hardware Design Processing Toolkit for Verilog HDL,” in *Applied Reconfigurable Computing: 11th International Symposium, ARC 2015, Bochum, Germany, April 13-17, 2015, Proceedings 11*. Springer, 2015, pp. 451–460.
- [33] C. Wolf, “Yosys open synthesis suite,” 2016.
- [34] C. Barrett, A. Stump, C. Tinelli *et al.*, “The smt-lib standard: Version 2.0,” in *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, vol. 13, 2010, p. 14.
- [35] C. W. Barrett, D. L. Dill, and J. R. Levitt, “A decision procedure for bit-vector arithmetic,” in *Proceedings of the 35th Annual Design Automation Conference*, 1998, pp. 522–527.
- [36] G. S. Tseitin, “On the complexity of derivation in propositional calculus,” *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pp. 466–483, 1983.
- [37] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, *Compilers principles, techniques & tools*. pearson Education, 2007.
- [38] J. Marques-Silva, I. Lynce, and S. Malik, “Conflict-driven clause learning sat solvers,” *Handbook of satisfiability*, pp. 131–153, 2009.
- [39] M. J. Heule, O. Kullmann, S. Wieringa, and A. Biere, “Cube and conquer: Guiding cdel sat solvers by lookaheads,” in *Haifa Verification Conference*. Springer, 2011, pp. 50–65.
- [40] C.-A. Wu, T.-H. Lin, C.-C. Lee, and C.-Y. Huang, “Qutesat: a robust circuit-based sat solver for complex circuit structure,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.
- [41] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, “A circuit-based sat solver for logic synthesis,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–6.
- [42] J. Chen and F. He, “Control flow-guided smt solving for program verification,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 351–361.
- [43] O. Shtrichman, “Tuning sat checkers for bounded model checking,” in *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*. Springer, 2000, pp. 480–494.
- [44] S. Berezin, S. Campos, and E. M. Clarke, “Compositional reasoning in model checking,” in *International Symposium on Compositionality*. Springer, 1997, pp. 81–102.
- [45] K. L. McMillan, “Applying sat methods in unbounded symbolic model checking,” in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 250–264.
- [46] E. M. Clarke, “Model checking,” in *International conference on foundations of software technology and theoretical computer science*. Springer, 1997, pp. 54–56.
- [47] X. Wang, A. J. Malozemoff, and J. Katz, “Emp-toolkit: Efficient multiparty computation toolkit,” 2016.

- [48] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
- [49] R. Mukherjee, M. Tautschnig, and D. Kroening, “v2c – A Verilog to C Translator Tool,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS ’16)*. Springer, 2016, pp. 580–586.
- [50] Various Contributors, “OpenCores.org,” 2021.
- [51] M. T. Hassan Salmani, “Trusthub,” <https://www.trust-hub.org/>.