Fair Watermarking Using Combinatorial Isolation Lemmas

Jennifer L. Wong, *Student Member, IEEE*, Rupak Majumdar, and Miodrag Potkonjak, *Member, IEEE*

***Abstract*—Watermarking is one of the most effective mechanisms for intellectual property protection (IPP) of hardware and software artifacts. Numerous watermarking-based IPP techniques have been proposed that satisfy a spectrum of IPP desiderata, including full preservation of functionality, low timing, area and power overhead, transparency to the synthesis and compilation process, and resilience against attacks. Two objectives that are very important, but, until now have not yet been properly addressed, are credibility and fairness. We present a new watermarking technique that specifically targets credibility and fairness. Using a combinatorial result by Valiant and Vazirani, we demonstrate how these two desiderata can be achieved during the watermarking of a satisfiability (SAT) instance. The effectiveness of the technique is demonstrated on both specially created examples, where the number of solutions is known, as well as on common computer-aided design and operational research SAT benchmark instances.**

***Index Terms*—Boolean satisfiability (SAT), intellectual property protection (IPP), watermarking.**

I. Introduction

THE REUSE of intellectual property (IP) such as IC cores and software libraries is widely considered to be the most economically efficient way to close the increasing gap between the ability of designers to develop integrated circuits and the potential of silicon. One of the prerequisites for hardware and software IP reuse is the development of intellectual property protection (IPP) techniques. Several approaches for IPP have been proposed, including watermarking, fingerprinting of silicon dies, and forensic engineering. In watermarking, the designer embeds extra information into a design for the purpose of identification or proof of authorship. Watermarking is one of the most effective IPP techniques due to its flexibility, strong proof of authorship, and very low overhead in terms of speed, area, and power. In the last several years, a number of watermarking-based IPP techniques have been developed at all levels of the design process, including system synthesis, behavioral synthesis, logic synthesis, and physical design. The key observation on which all watermarking-based IPP techniques are based is the fact that many synthesis problems are associated with computationally intractable or difficult optimization problems which often have a high number of different solutions with identical or very similar quality. The key idea of watermarking-based techniques is to use this fact by incorporating an encrypted signature of the designer into the design specification as additional constraints and therefore ensuring that the completed design satisfies both the initial specification as well as the new constraints. Thus, the final design depends uniquely on the encrypted signature that only the author of the design knows. While watermarking techniques perform well in practice, relatively little is known about their theoretical underpinnings. Two issues, in particular, deserve more sound and effective treatment: the calculation of *credibility of ownership* (informally, the strength of the proof of authorship) and *fairness* (informally, a measure of the difficulty of finding a solution in the watermarked design). We focus our research on watermarking for the Boolean satisfiability (SAT) problem due to its wide use in optimization and design tasks in computer-aided design (CAD), artificial intelligence, and operations research.

It is important to emphasize that neither credibility nor fairness can be defined in an unambiguous way that is eficiently measureable in practice. This is because both credibility and fairness are related to finding multiple solutions to a computationally intractable (NP-complete) problem. Even from a conceptual point of view, it is not clear if a definition based on the number of solutions or one based on the required effort (e.g., time) to find a solution is more adequate. Even if one prefers one of these two options, the unavoidable logistic problem is that one cannot provide a unique operational definition for either. In the case of the number of solutions, since the problem is computationally intractable, it is impossible to count the number of solutions for standard benchmarks. While time or memory space required to find a solution can be measured, both greatly depend on the particular algorithm and particular system used to solve the particular instance.

Intuitively, we propose to measure the strength of the proof that a considered solution is indeed produced after the addition of watermarking constraints by measuring the likelihood of the solution being selected at random. Therefore, one can define credibility as the ratio of the number of solutions that satisfy the watermarking constraints and the number of solutions of the original problem. In this case, a lower ratio indicates higher credibility. Therefore, we can conclude that ultimate credibility is achieved if after the addition of a watermark, only a single solution exists for an instance.

Credibility can also be defined with respect to the effort required to find a particular solution. One can claim that credibility is high if after the addition of watermarking constraints, all or at least a majority of state-of-the-art solvers will produce with high likelihood a watermarked solution and that in the absence of the watermark they will produce with high likelihood a different solution. Note that many watermarking techniques can provide very high credibility if the length of the message is increased.

Manuscript received December 31, 2002; revised August 29, 2003 and February 22, 2004. This paper was recommended by Associate Editor J. H. Kukula.

The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90095-1596 (e-mail: jwong@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2004.836730

0278-0070/04$20.00 © 2004 IEEE

From a practical point of view, a good aim for addressing credibility would be to evaluate the technique according to its ability to provide a variety of tradeoffs: strength of watermarking proof versus quality of solutions, or in the case of SAT, time required to find a solution.

Fairness can be defined as a function that measures the difficulty of finding a solution after a variety of watermarks of a specific length are embedded. Again, the definition of difficulty can be established in multiple ways. One approach is to define fairness as the number of solutions after the watermark is embedded. Another approach is to measure how difficult it is to find a solution once the signature is embedded, which can be quantified using the runtime on common solvers. Note that one can define a variety of statistics that combine the difficulty of specific instances.

Recently, Qu *et al.* [1] introduced the first watermarking technique which embeds instance-specific constraints to ensure fairness. However, that technique is empirical and does not provide any theoretical guarantees. We provide here a new watermarking technique that addresses these issues. Our new method is based on a combinatorial result by Valiant and Vazirani [2] that randomly reduces a SAT instance to an instance which has exactly one satisfying assignment (with high probability). The Valiant–Vazirani approach successively adds constraints to the original formula to produce a series of formulas that have a monotonically decreasing number of solution. By utilizing a binary search, one can effectively pursue the maximal length of the signature, which still does not make the formula unsatisfiable. Another important property is that if additional constraints are added at random, there is very high probability that all signatures will terminate with unique solutions at very similar watermark lengths.

II. Related Work

The related work can be traced in three different research directions: the SAT problem, IPP using watermarking, and algorithmic techniques for producing instances with unique solutions.

A. SAT Problem

The Boolean SAT problem asks, for a given input Boolean expression E, if there is some assignment to the variables in E such that E evaluates to true. (See the equation at the bottom of the page.)

SAT is, of course, NP-complete [3]. SAT has numerous applications both in very large scale integrated (VLSI) CAD and other domains, such as artificial intelligence, operations research, and combinational optimization. SAT has been successfully used for automated test pattern generation [4], deterministic test pattern generation, delay fault testing, logic synthesis and verification, model checking, field programmable gate array (FPGA) routing [5], and timing analysis. Furthermore, SAT has also been used to solve covering problems [6], physical design problems, integer and 0–1 linear programming problems [7], [8], and finding prime implicants of Boolean functions. An excellent survey on SAT in electronic design automation is [9].

While theoretically intractable, several heuristic techniques have been developed to solve the SAT problem efficiently on real-world instances. These include backtrack search [10], local search [11], algebraic manipulation, continuous formulation, and recursive learning [10].

B. Watermarking

Ultrahigh system-on-chip integration depends on the availability of third party hardware and software components. Therefore, to facilitate viable business models, there is a strong need for a variety of IPP techniques. A number of IPP techniques such as watermarking [1], [12]–[18], fingerprinting [19], [20], software obfuscation, and forensic engineering [21], [22] have been proposed. We focus our review of related work on watermarking due to its direct relevance to this work. There are two conceptually different domains where watermarking is applied: for static artifacts (e.g., graphics and multimedia) and functional artifacts (e.g., software and integrated circuits design).

Watermarks for static artifacts use imperfections in human perception to add signatures to the object. On the other hand, watermarks for functional artifacts must fully preserve their functional specifications. Here, the majority of approaches embed the watermark into the design by superimposing additional constraints. Functional artifacts can be specified and therefore watermarked at several levels of abstraction such as system level designs, FPGA designs [23], at the behavioral and logic synthesis levels, and physical designs [24], [13]. Techniques have also been developed for watermarking of DSP algorithms, sequential circuits, sequential functions [25], [26], and analog designs [27].

The basis for our watermarking technique is a paper by Valiant and Vazirani [2]. They proved that the number of solutions of a SAT problem, which can vary from zero to exponentially many, does not impact its inherent intractability. Their main technical construction is a lemma for reducing the number of solutions of an arbitrary SAT instance (with high probability) by adding random additional constraints.

There are two main advantages of the new technique over previously published techniques. First, the new technique provides a proof of credibility by enabling the designer to impose a number of constraints in such a way that there exist only unique solutions. These solutions correspond to the signature. Second, the technique provides strong probabilistic proof that the fairness property is enforced during the watermarking process.

Problem : Boolean Satisfiability
Instance : A set U of variables and a collection C of clauses over U.
Question : Is there a satisfying truth assignment for C?

III. Preliminaries

In this section, we briefly survey the constraint-based watermarking methodology. Fig. 1 illustrates the generic watermarking technique. There are two inputs, the initial instance of the optimization problem (which corresponds to optimization synthesis or a compilation problem) and the owner's signature. The signature is translated into a set of additional constraints using the proposed watermarking technique, which should satisfy tests for randomness. Once the additional constraints are defined, they are combined with the original instance specification creating the overconstrained specification of the problem instance. The overconstrained instance is then solved using any solver for the problem. The solution obtained from the solver is a watermarked solution, since the solution satisfies both the initial instance specification and the additional constraints added from the signature bitstream. The last step of the process flow is to evaluate the effectiveness of the watermark.

In order to generate a bitstream for watermark encoding which is representative of the encoder's signature, we introduce the following process shown in Fig. 2. By using the pretty good privacy (PGP) encoding scheme, a sufficiently randomized bitstream is created resulting in watermarking constraints which are difficult to imitate and detect.

The figure illustrates the encoding and verification process for watermarking. The PGP software package is used to encode the watermark signature with the users private key. A seedfile is generated from PGP software that is used to create a signature-related bit-stream. The bit-stream, which is the output of the RC4 stream cipher, is a cryptographically strong pseudorandom bit-stream. This bit-stream is the basic signature which is used by the watermarking techniques.

To verify a signature, one must show that both the signature is present in the SAT solution and that the signature corresponds to the textfile and the PGP public key of the supposed owner. Demonstrating that the signature exists in the SAT solution is achieved by demonstrating that the solution satisfies all the constraints claimed by the author. One can show that the signature corresponds to the textfile and the owner's public key by running PGP.

The crucial observation is that during the creation of additional constraints, suitable for intentional watermarking is that one has to establish a well-defined ordering for the components of the instance. In the case of the SAT problem, the components are variables and clauses. This can be done in two ways, either by using industry-imposed standards or by using properties of the designs. For example, for the second option, for the SAT problem, we can use orderings according to both clauses and variables. In the case of variables, for example, we can use rank order rules such as the number of appearances of variables in all clauses, the number of complemented forms of each variable, and the number of occurrences of variables and uncomplemented variables in clauses of odd length. Once this well-defined ordering is available, we can assign a specific rank to each variable and to each clause. After this step, additional constraints can be added in a unique way to the instance and one can establish a unique relationship

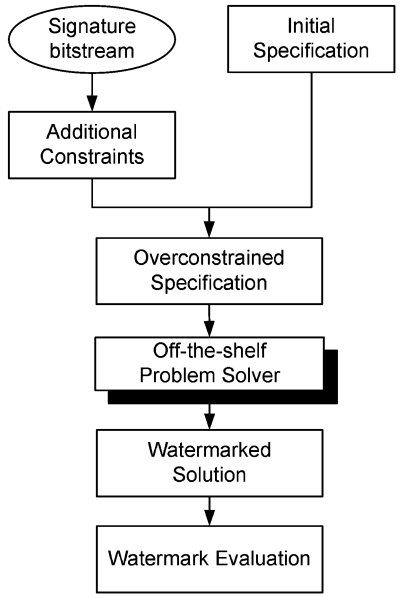

Fig. 1. Watermarking-Based IPP process flow.

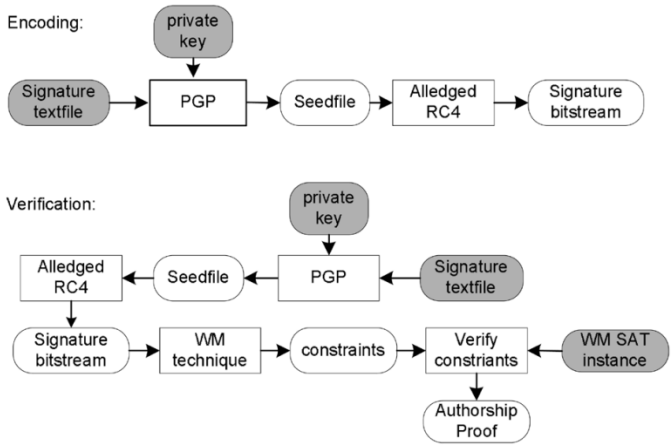

Fig. 2. Procedure for translation of arbitrary signature to infinite random string.

between each bit in the user's signature and each new constraint added to the instance. Furthermore, by following the ordering, one can conduct reverse engineering of the original signature, which is one of the key objectives for establishing proof of authorship.

The output of the process is a watermarked design which can be analyzed according to standard watermarking desiderata, which include high credibility, high resiliency against attacks, low overhead, complete transparency to the standard problem solving tools, and partial protection and fairness. The essence of constraint-based watermarking is to restrict the users solution to the part of solution space which is characterized by the signature constraints. The key essential assumption is that there are numerous solutions of high and very similar quality. In the case of decision problems, such as SAT, the addition of extra constraints should not change the positive answer to the initial instance of the problem to a negative answer after the addition of the watermarking constraints.

IV. Creating Unique and Fair Solutions to SAT

In this section, we present the mathematical foundation for the new watermarking technique. We start by presenting the Valiant–Vazirani Isolation Lemma and an illustration of its application on a small example.

The method is based on a combinatorial result of [2] that isolates a solution of a conjunctive normal form (CNF formula) by randomized reduction. Given an instance f of SAT, the method successively conjoins constraints to f to obtain a series of formulas $f_1, f_2, \ldots, f_n$ that will have a decreasing number of solutions. If f is satisfiable, we can prove that with probability at least 1/4, one of the formulas will have a unique solution. If we choose one of the formulas at random, then the probability that it has a unique solution is at least 1/4. This probability can be boosted as usual. On the other hand, if f is not satisfiable, then each of the formulas will be unsatisfiable.

The watermarking method will consequently construct, given an instance of SAT, a formula with a unique satisfying assignment, and produce the unique assignment as the solution. The construction will ensure that this assignment satisfies the original formula f. However, the probability that a random algorithm picks exactly this satisfying assignment is low. We now outline the construction. The treatment is from [2]. We omit the proofs of correctness.

We shall select constraints at random from some suitable set. Ideally, we would like to eliminate each solution independently with a certain probability. This is not possible with only a polynomial number of random choices. However, the use of GF[2] inner products with polynomially few vectors over $\mathrm{GF}[2]^n$ suffices for our purposes. Let f be a CNF formula over the variables $x_1, x_2, \ldots, x_n$. We shall view truth assignments to the variables $x_1, x_2, \ldots, x_n$ as n-dimensional $\{0,1\}$ vectors over the vector space $\mathrm{GF}[2]^n$. The satisfying assignments of f form a set of vectors from this space. For $u, v \in \mathrm{GF}[2]^n$, let $u \cdot v$ denote the inner product over GF[2] of u and v.

Lemma 1: If f is any CNF formula in $x_1, x_2, \ldots, x_n$ and $w_1, \ldots, w_k \in \{0,1\}^n$, then one can construct in linear time a formula f'_k whose satisfying assignments v satisfy f and the equations $v \cdot w_1 = v \cdot w_2 = \cdots = v \cdot w_k = 0$. Furthermore, one can construct a polynomial-size CNF formula f_k in variables $x_1, \ldots, x_n, y_1, \ldots, y_m$ for some m, such that there is a bijection between solutions of f_k and f'_k defined by equality on the values of $x_1, \ldots, x_n$.

Proof: We show the lemma for $k = 1$. The general case follows easily. The formula f'_1 is

$$f \wedge (x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_j} \oplus 1)$$

where $\oplus$ denotes the exclusive-or function, and $i_1, \ldots, i_j$ are the indices of the x_i that have value 1 in w_1. The function f_1 is the CNF equivalent of f'_1

$$f \wedge (y_1 \Leftrightarrow x_{i_1} \oplus x_{i_2}) \cdots \left(y_{j-1} \Leftrightarrow y_{j-2} \oplus x_{i_j}\right) (y_{j-1} \oplus 1).$$

```
/* Precondition: f is of the form x_1 ⊕ x_2 ⊕ ... x_k ⊕ 1 */
convertToCNF(formula f){
    let {y_1, y_2, ..., y_k} be new variables;
    /* let a ⇔ b ⊕ c denote the CNF formula
    (ā ∨ b̄ ∨ c̄)(ā ∨ b ∨ c)(a ∨ b ∨ c̄)(a ∨ b̄ ∨ c) */
    return ((y_1 ⇔ x_1 ⊕ x_2)(y_2 ⇔ y_1 ⊕ x_3) ...
            (y_{k-1} ⇔ y_{k-2} ⊕ x_k)(y_{k-1} ⊕ 1)); }
formula addOneConstraint(formula f){
    pick n bits for w from signature bitstream;
    let {i_1, ..., i_k} be positions of the 1 entries in w;
    return f ∧ convertToCNF(x_{i_1} ⊕ x_{i_2} ⊕ ··· ⊕ x_{i_k} ⊕ 1); }
/* Precondition: f is a CNF formula over variables
    {x_1, ..., x_n} */
formula
generateConstrainedFormula(formula f){
    get t from {1, ..., n} from signature bitstream;
    for (i = 1 to t) do
        f = addOneConstraint(f);
        od }
```

Fig. 3. Pseudocode for watermarking using combinatorial isolation lemmas.

The intuition behind the construction is the following surprising fact. Let S be a subset of $\{0,1\}^n$. Define the sets

$$S_1 = \{v \,|\, v \in S, v \cdot w = 0\}$$
$$S'_1 = \{v \,|\, v \in S, v \cdot w = 1\}.$$

Then, if w is chosen randomly, any S will be partitioned in this way into two roughly equal halves with high probability. In our construction, S is the set of satisfying assignments of f, we choose $w_1, \ldots, w_k$ at random, and constructing f_k, we obtain a formula with roughly $2^{-k}|S|$ satisfying assignments. Note that we do not know $|S|$, other than that it lies between 0 and 2^n. Therefore, we need to "guess" the size of $|S|$. This is where the random choice of k comes in: with probability $(1/n)$, we make the right guess.

The overall construction is simply the following. Given a CNF formula f, choose an integer k at random from $\{1, \ldots, n\}$, randomly choose vectors $w_1, \ldots, w_k$, and output f_k. We now give the technical result that formalizes the above intuition.

Lemma 2: Let $S \subseteq \{0,1\}^n$. Suppose $w_1, \ldots, w_k$ are chosen at random. For each $i \leq n$, let $S_i = \{v \,|\, v \in S, v \cdot w_1 = \cdots = v \cdot w_i = 0\}$, and let $P_n(S)$ be the probability that, for some $i \leq n, |S_i| = 1$. Then

1) $P_n(S) \geq (1/4)$;
2) if $w_1, \ldots, w_n$ are chosen to be linearly independent in addition, then $P_n(S) \geq (1/2)$.

Proof: This is the main construction of [2].

Fig. 3 shows the algorithm to produce the final formula (conjoined with the additional constraints). The function addOneConstraint adds one more constraint to the current formula, thus making the number of solutions drop to roughly half the original number (with high probability). The function generateConstrainedFormula() is a loop that calls addOneConstraint k times. Note that every call effectively reduces the number of satisfying assignments by half. The function convertToCNF takes a formula of the form $x_1 \oplus \cdots \oplus x_k \oplus 1$ and converts it to a CNF formula (with new variables).

Example: As an example of the application of method, consider the CNF formula

$$f = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_3 \vee x_4)(x_2 \vee \overline{x}_3 \vee x_4)$$

over the variables $\{x_1, x_2, x_3, x_4\}$. This expression has ten satisfying assignments, namely $\{0000, 0001, 0011, 0110, 0111, 1001, 1011, 1101, 1110, 1111\}$. The watermarking process begins with $\mathrm{generateConstrainedFormula}$. In this step, we select a number t from our bit-stream which is between 1 and n, where n is the number of variables in the original instance specification. In our case, $n = 4$. Suppose that our bitstream generates $t = 2$. Therefore, $\mathrm{addOneConstraint}(f)$ is called twice.

In the first invocation, we select a watermark string w_1 from the bitstream of length equal to the number of variables in the original instance. Suppose the first watermark w_1 is 0101. We associate each bit of the watermark with a single variable in the original instance. Therefore, x_1 is associated to 0, x_2 to 1, x_3 to 0, and x_4 to 1. In this case, there are two 1's in the watermark that are associated with x_2 and x_4. The following formula f_1 is then created and passed to the $\mathrm{convertToCNF}$ function:

$$f_1 = (x_2 \oplus x_4 \oplus 1) = (y_1 \Leftrightarrow x_2 \oplus x_4)(y_1 \oplus 1).$$

In $\mathrm{convertToCNF}$, the formula f_1 is converted to CNF form. In this case, the first term $(y_1 \Leftrightarrow x_2 \oplus x_4)$ translates into four new CNF clauses $(\overline{y_1} \vee \overline{x_2} \vee \overline{x_4})(\overline{y_1} \vee x_2 \vee x_4)(y_1 \vee \overline{x_2} \vee x_4)(y_1 \vee x_2 \vee \overline{x_4})$. The final term $(y_1 \oplus 1)$ translates solely to a single clause, $(\overline{y_1})$. These clauses are appended to the original formula f, creating f', and returned to $\mathrm{generateConstraindFormula}$

$$\begin{aligned} f' = {} & (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_3 \vee x_4)(x_2 \vee \overline{x}_3 \vee x_4) \\ & (\overline{y}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ & (\overline{y}_1 \vee x_2 \vee x_4)(y_1 \vee \overline{x}_2 \vee x_4)(y_1 \vee x_2 \vee \overline{x}_4)(\overline{y}_1). \end{aligned}$$

Formula f' now has four satisfying assignments $\{0000, 0111, 1101, 1111\}$. In this first step, the number of solutions was reduced from ten to four, approximately half. In the second iteration, $\mathrm{addOneConstraint}$ is called with f'. Note that only the original variables (x_i) are used to watermark the instance throughout the watermarking process despite the fact that new variables (y_i) are created. A second watermark string from the signature bitstream is selected, $w_2 = 0011$. In this case, the 1's correspond to variables x_3 and x_4. The constraint below f_2 is created and encoded to CNF form

$$f_2 = (x_3 \oplus x_4 \oplus 1) = (y_2 \Leftrightarrow x_3 \oplus x_4)(y_2 \oplus 1).$$

The formula f'' is the overconstrained instance after both w_1 and w_2 have been encoded into the original formula f. This formula has only three satisfying assignments $\{0000, 0111, 1111\}$. The assignment 1101 was eliminated by the constraint $(y_2 \vee x_3 \vee \overline{x_4})$. Note that now the instance has the following form:

$$\begin{aligned} f'' = {} & (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_3 \vee x_4) \\ & (x_2 \vee \overline{x}_3 \vee x_4)(\overline{y}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ & (\overline{y}_1 \vee x_2 \vee x_4)(y_1 \vee \overline{x}_2 \vee x_4)(y_1 \vee x_2 \vee \overline{x}_4)(\overline{y}_1) \\ & (\overline{y}_2 \vee \overline{x}_3 \vee \overline{x}_4)(\overline{y}_2 \vee x_3 \vee x_4)(y_2 \vee \overline{x}_3 \vee x_4) \\ & (y_2 \vee x_3 \vee \overline{x}_4)(\overline{y}_2). \end{aligned}$$

The beauty of the technique is that the final algorithm is extremely simple (it involves only some random choices), yet it yields optimal results with high probability. A discussion of potential applications of the new watermarking technique to other optimization problems can be found in [28].

Input: k number of variables.
m number of solutions.
Output: SAT instance and solutions.

Algorithm:
createSATInstance() {
1. *generateSolutions(k, m);*
2. *orderVariables();*
3. *eliminateNonSolutions();*
4. *Addition of Confusion();* }
5. *generateSolutions(k, m)* {
6. Random number generation of m solutions using linear hash function; }
7. *orderVariables()* {
8. Order pairs of variables according to their constancy. }
9. *eliminateSolutions()* {
10. Create clauses which prevent non-solutions from being solutions. }

Fig. 4. Algorithm for the creation of a SAT instance with a specific number of solutions.

V. Software Experimental Environment

We now present the software environment which is used for experimental evaluation of the new watermarking technique. Specifically, we have developed three programs: 1) a procedure that generates an instance of the SAT problem with a user specified number of solutions for a requested number of variables; 2) a program for a branch and bound-based exhaustive enumeration of the solutions of a SAT instance; and 3) a program for watermarking SAT instances using the combinatorial isolation lemmas. In addition, we also used several public domain SAT solvers.

At the intuitive level, the program for creating an instance of SAT with a known number of solutions consists of four phases. First, we use a random-number generator and linear hash function-based algorithm for avoidance of collision to generate n different assignments of variables which will constitute solutions to the instance of the created SAT problem. In the second phase, we order the variables according to the diversity of their mutual literal appearance. Note that two variable x_i and x_j can appear in a total of four combinations together: $x_i x_j, \overline{x}_i x_j, x_i \overline{x}_j, \overline{x}_i \overline{x}_j$. We order the variable pairs which appear in the fewest number of combinations first in order to cut the solution space as rapidly as possible. Next, we add clauses to eliminate all nonvalid solutions. Finally, in the last phase we alter some of the clauses and add additional clauses to better hide the structure of the instance. Fig. 4 shows pseudocode for the creation of the SAT instance.

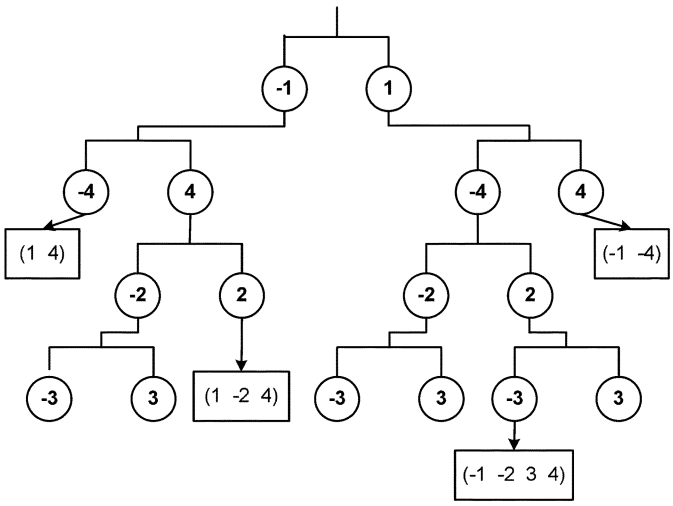

Fig. 5. Branch and bound tree for the creation of a SAT instance with four variable and five solutions.

To clarify the process, consider the following example. Our goal is to create an SAT instance defined using four variables x_1, x_2, x_3, x_4 and five solutions. Using the random-number generator and linear hash function we generate the following assignments of variables, which are the solutions to the instance:

$$(\overline{x}_1, \overline{x}_2, x_3, x_4)(\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4)(x_1, \overline{x}_2, \overline{x}_3, \overline{x}_4)$$
$$(x_1, \overline{x}_2, x_3, \overline{x}_4)(x_1, x_2, x_3, \overline{x}_4).$$

Now, we order the variables according to the least number of pair combinations. As a result of pair x_1 and x_4 appearing together in only two forms $\overline{x}_1 x_4$ and $x_1 \overline{x}_4$, we order these two variables first. We then compare the rest of the variables to the previous pair. The resulting ordering is x_1, x_4, x_2, x_3. We begin adding clauses by building a branch and bound binary search tree as shown in Fig. 5. We begin by adding variable x_1. By examining our solutions, we see that x_1 appears in both forms x_1 and $\overline{x}_1$, therefore, we cannot terminate any branches. We continue for x_4. We see that no solutions have the form $\overline{x}_1, \overline{x}_4$ or x_1, x_4. We terminate these branches and create clauses that eliminate all solutions of these forms. In this case, we add clauses $(x_1 \vee x_4)$ and $(\overline{x}_1 \vee \overline{x}_4)$. We continue this process until all branches which lead to nonvalid solutions have been terminated by the creation of appropriate clauses. The created instance is as follows:

$$f = (\overline{x}_1 \vee \overline{x}_4)(x_1 \vee x_4)(x_1 \vee \overline{x}_2 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee x_4).$$

The last step is to add additional clauses and to alter the clauses to hide the structure of the instance and increase the complexity of the instance.

The second program also uses the branch and bound technique to enumerate all solutions of a given instance. We implemented the algorithm shown in Fig. 3 for watermarking SAT instances. The watermarked instances were tested on the following public domain SAT solvers WalkSAT [29], zChaff [30], and Rel_SAT [31].

TABLE I
DIMACS INSTANCES WITH ORIGINAL RUNTIME ON WALKSAT AND ZCHAFF SOLVERS

Instance	# Vars	# Clauses	Orig. WalkSAT (sec)	Orig. zChaff (sec)
par8-1-c.cnf	64	254	0.1	0.01
jnh1.cnf	100	850	0.7	0.01
uf225-097.cnf	225	960	0.1	4.86
par16-3-c.cnf	334	1332	9.6	4.38
f600.cnf	600	2550	1.3	-
hanoi4.cnf	718	4934	12.2	2.67
ii8c2.cnf	950	6689	0.1	0.05
f1000.cnf	1000	4250	0.4	-
par16-1.cnf	1015	3310	7.9	1.46
par32-2-c.cnf	1303	5206	12.7	-
ii16a1.cnf	1650	19368	0.2	-
ii16b1.cnf	1728	24792	0.4	57.51
g125.17.cnf	2125	66272	63.3	-
par32-1.cnf	3176	10277	10.9	-

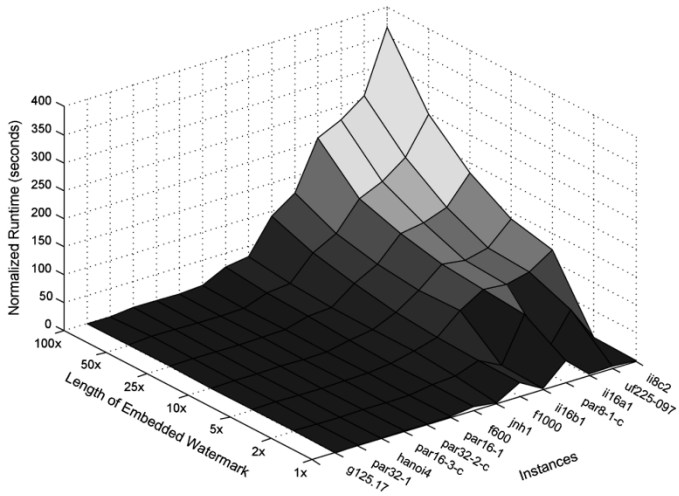

Fig. 6. Credibility is demonstrated by continuous tradeoff between the strength of watermark (length) and required runtime (walkSAT) on DIMACS examples.

VI. EXPERIMENTAL RESULTS

In this section, we present simulation results that evaluate the effectiveness of our watermarking technique. We first present experimental results related to credibility. Then, we present experimental results for fairness. Both techniques are analyzed using measures based on the number of solutions as well as required runtime.

Since SAT is a decision problem, there is no overhead in terms of impact on the quality of the solution. All solutions are of equal quality. The overhead in terms of runtime can be directly recorded from the increase in runtime of the solvers. The proposed watermarking technique is completely transparent to all available SAT solvers. SAT instances are intrinsically nonpartitionable and, therefore, it is inappropriate to discuss partial protection in this case. Finally, it is important to emphasize that the main protection of any integrated circuit watermarking technique against an arbitrary attack is not that the attacker can not reuse a part of the solution to find another solution, but in the inherent structure of the design process, where alteration of the design at the higher levels of synthesis process inevitably

TABLE II
CREDIBILITY IS DEMONSTRATED BY CONTINUOUS TRADEOFF BETWEEN THE STRENGTH OF WATERMARK (LENGTH) AND THE REQUIRED RUNTIME (ZCHAFF) ON DIMACS EXAMPLES

	par8-1-c	jnh1.cnf	uf225-097	par16-3-c	hanoi4	ii8c2	par16-1	ii16b1	ii16a1
1x	0.10	1.70	0.42	2.35	1.68	1.62	0.80	1.14	0.96
2x	0.73	1.60	1.25	1.18	1.35	1.82	1.21	1.54	0.93
5x	0.10	1.70	0.42	2.35	1.68	1.62	0.80	1.14	0.88
10x	9.60	7.78	1.54	-	-	2.38	-	1.12	1.39
25x	-	-	108.15	-	-	2611.47	-	32.20	-
50x	-	-	-	-	0.04	-	-	-	-
100x	-	416.00	-	299.86	-	-	-	-	-

has as a side effect, requirements for more comprehensive alterations of the solution at lower levels of abstractions. Therefore, the attacker is forced to spend significant time and effort to find a suitable new solution; essentially he/she must redo the entire design.

There are two main conceptual difficulties in evaluating the proposed watermarking technique. The first is that the technique is used to watermark instances of an NP-complete problem. Therefore, it is unlikely that one can guarantee to find a solution, and even less likely to find all solutions. The second problem is that the exact performance of an experimentally evaluated watermarking technique is most likely correlated, at least to some unknown extent, to a particular solver used to solve the instances of the SAT problem.

In order to resolve these two conceptual problems and to quantify the effectiveness of the proposed watermarking approach, we conducted two sets of experiments. The goal of the first set was to establish how well the technique performs on standard benchmarks. Note that for this type of instances, the number of solutions is unknown. The set consists of popular DIMACS examples shown in Table I. In order to enhance the diversity, we selected examples that significantly differ in terms of the number variables, number of clauses, and the ratio of number of clauses to number of variables. The last criteria, ratio of number of clauses to number of variables, is often a good measure of the difficulty to solve the instance. We present the name of the instance along with the number of variables and number of clauses for a subset of the DIMACS instances. The last two columns indicates the runtime, in seconds, to solve the initial instance using the WalkSAT solver and the zChaff solver. The second set of experiments was performed on a set of instances that were constructed in such a way that the number of solutions is known, as described in the previous section.

A. Credibility

We first evaluated runtime-based credibility using DIMACS instances. In Fig. 6, we present the average normalized runtime for each DIMACS instance after 100 watermarks of each signature length are embedded. The length of the watermark signatures are 1, 2, 5, 10, 25, 50, and 100 times the number of variables in each instance. On the two horizontal axes we present the DIMACS instances and the length of the embedded watermarks. On the vertical axis, the normalized runtime using the WalkSAT solver is shown in seconds. Note that the new watermarking technique provides a continuous smooth tradeoff

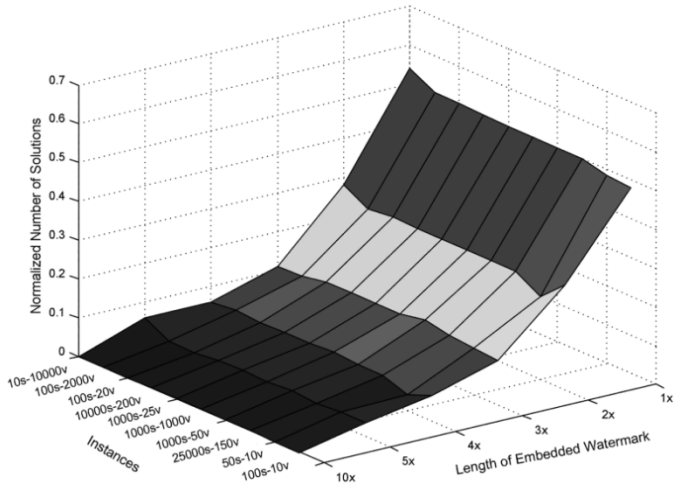

Fig. 7. Experimental results for credibility on created SAT instance with known number of solutions.

TABLE III
EXPERIMENTAL RESULTS FOR CREDIBILITY. AVERAGE RUNTIME FOR CREATED INSTANCES WHERE THE NUMBER OF SOLUTIONS IS KNOWN

	1x	2x	3x	4x	5x	10x
Average	0.502	0.251	0.122	0.061	0.035	6.2e-4

between the length of watermark and runtime and therefore it is well suited to be used as a high-credibility IPP technique.

Additionally, we performed an identical analysis using the zChaff solver. The normalized runtime for instances which could be solved by the zChaff solver after embedding the various length watermarks are shown in Table II. Instances which are not listed in this table, could not be solved by zChaff under the standard time limit.

In order to evaluate credibility tradeoffs using a measure that is based on the number of solutions, we tested the approach on instances with the known number of solutions. These instances were created using the approach presented in Section V. The number of variables for each instance varied from 10 to 10 000 and the number of solutions ranged from 10 to 25 000. One hundred watermarks of lengths 1, 2, 3, 4, 5, and 10 times the number of variables in each instance were embedded using the proposed technique. After the addition of the watermark, the number of solutions still valid were enumerated. In Fig. 7, we present the results. The created instances are labeled as the number of solutions and the number of variables (i.e., the instance 1000 s–1000 v has 1000 solutions and 1000 variables) and are denoted on one horizontal axis. The other horizontal

axis shows the length of the watermark. The average normalized number of solutions remaining after the embedding of the watermark of a given length is shown on the vertical axis.

We present the normalized average number of solutions remaining after the addition of the watermark of a given length for each instance in Table III. The normalization is conducted against the initial number of solutions before watermarking. It is easy to see that in Fig. 7 and Table III, the number of solutions scales almost exactly as predicted by the combinatorial isolation lemma.

B. Fairness

To evaluate the fairness of the proposed technique, we evaluated both runtime and solution count-based measures. Experiments were performed on the set of 14 DIMACS instances. For each instance, we embedded 100 watermarks of 5, 10, 25, 50, and 100 times the number of variables in the instance. In more than 90% of the cases the variance is less than 0.1, and in 95% of cases the worst-case difference is less than 0.1. The overall low variance clearly indicates high fairness. Comprehensive results are available in [28].

Evaluation of the technique using instances with the known number of solutions was performed on examples created with a specified number of variables and solutions. For this purpose we used the same instances evaluated in the experimental results for credibility validation. In all cases, the maximum variance between the number of solutions remaining after the addition of the watermark of given length is 0.009. Again, the small variance in the number of solutions indicates that the technique is reduces the number of solutions to the instances in a fair manner. A comprehensive table of the results are available in [28].

VII. Conclusion

We have presented a new scheme for watermarking SAT solutions using a combinatorial isolation lemma [2]. The watermarking scheme has good theoretical properties (it provides both credibility and fairness with high probability). Moreover, we have demonstrated that there is a close match between the theoretical predictions and experimental results on standard SAT benchmarks and tools.

References

[1] G. Qu, J. Wong, and M. Potkonjak, "Fair watermarking techniques," in *Proc. Asia South Pacific Design Automation Conf.*, 2000, pp. 55–60.

[2] L. Valiant and V. Vazirani, "NP is as easy as detecting unique solutions," *Theor. Comput. Sci.*, vol. 47, pp. 85–93, 1986.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[4] J. Marques-Silva and K. Sakallah, "Robust search algorithms for test pattern generation," in *Fault-Tolerant Computing Symposium (FTCS)*, 1997, pp. 1–10.

[5] G. Nam, K. Sakallah, and R. Rutenbar, "Satisability based FPGA routing," in *International Conference on VLSI Design*, 1999, pp. 574–577.

[6] V. Manquinho and J. Marques-Silva, "On using satisfiability-based pruning techniques in covering algorithms," *Design Automation and Test in Europe*, pp. 356–363, 2000.

[7] P. Barth, "A Davis-Putnam based enumeration algorithm for linear pseudo Boolean optimization," Max-Planck-Institut fur Informatik, Tech. Rep., 1995.

[8] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Generic ILP versus specialized 0–1 ILP: An Update," in *Proc. Int. Conf. Computer-Aided Design*, 2002, pp. 450–457.

[9] J. Marques-Silva and K. Sakallah, "Boolean satisfiability in electronic design automation," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 675–680.

[10] ——, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, pp. 506–521, May 1999.

[11] B. Selman, H. Levesque, and D. Mitchell, "A new method for solving hard satisability problems," in *AAAI*, 1992, pp. 440–446.

[12] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design IP protection," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1236–1252, Oct. 2001.

[13] D. Kirovski, Y.-Y. Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection by watermarking combinational logic synthesis solutions," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 194–198.

[14] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 190–193.

[15] J. L. Wong, G. Qu, and M. Potkonjak, "Optimization-intensive watermarking techniques for decision problems," *IEEE Trans. Computer-Aided Design*, vol. 23, pp. 50–59, Jan. 2004.

[16] G. Wolfe, J. L. Wong, and M. Potkonjak, "Watermarking graph partitioning solutions," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 1196–1204, Oct. 2002.

[17] D. Kirovski and M. Potkonjak, "Local watermarks: Methodology and application to behavioral synthesis," *IEEE Trans. Computer-Aided Design*, vol. 22, pp. 1277–1284, Sept. 2003.

[18] G. Qu, "Publicaly detectable techniques for the protection of virtual components," in *Proc. Design Automation Conf.*, 2001, pp. 474–479.

[19] G. Qu and M. Potkonjak, "Fingerprinting intellectual property using constraint-addition," in *Proc. Design Automation Conf.*, 2000, pp. 587–592.

[20] A. Caldwell, H. Choi, A. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J. Wong, "Effective iterative techniques for fingerprinting design IP," *IEEE Trans. Computer-Aided Design*, vol. 23, pp. 208–215, Feb. 2004.

[21] J. L. Wong and M. Potkonjak, "Relative generic computational forensic techniques," presented at the Inform. Hiding Workshop, 2004.

[22] J. L. Wong, D. Kirovski, and M. Potkonjak, "Forensic engineering techniques for VLSI CAD tools," *IEEE Trans. Computer-Aided Design*, vol. 23, pp. 581–586, June 2004.

[23] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Fingerprinting techniques for field-programmable gate array intellectual property protection," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1253–1261, Oct. 2001.

[24] A. Kahng, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust IPP watermarking methodologies for physical design," in *Proc. Design Automation Conf.*, 1998, pp. 782–787.

[25] R. Chapman and T. Durrani, "IP protection of DSP algorithms for system on chip implementation," *IEEE Trans. Signal Process.*, vol. 48, pp. 854–861, Mar. 2000.

[26] A. Oliveira, "Techniques for the creation of digital watermarks in sequential circuit designs," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1101–1117, Sept. 2001.

[27] R. Newbould, D. Irby, J. Carothers, and J. Rodriguez, "Watermarking IC's for IP protection," *IEE Electron. Lett.*, vol. 38, no. 6, pp. 272–274, 2002.

[28] J. L. Wong, R. Majumdar, and M. Potkonjak, "Fair watermarking using combinatorial isolation lemmas," Univ. California, Los Angeles, CA, Rep. 040017, 2004.

[29] B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Proc. Cliques, Coloring, Satisfiability: 2nd DIMACS Implement. Challenge*, 1993, pp. 521–532.

[30] L. Zhang and S. Malik, "Conflict driven learning in a quantified Boolean satisfiability solver," in *Proc. Int. Conf. Computer-Aided Design*, 2002.

[31] R. Bayardo and R. Schrag, "Using CSP look-back techniques to solve exceptionally hard SAT instances," *Principles Practice Constraint Program.*, pp. 46–60, 1996.

Jennifer L. Wong (S'99) received the B.S. degree in in computer science and engineering and the M.S. degree in computer science in 2000 and 2002, respectively, from the University of California, Los Angeles, where she is currently pursuing the Ph.D. degree.

Her research interests include intellectual property protection, optimization for embedded systems, and mobility in ad-hoc sensor networks.

Rupak Majumdar received the B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1998 and the Ph.D. degree in computer science from the University of California, Berkeley, in 2003.

Since 2003, he has been an Assistant Professor in the Department of Computer Science, University of California, Los Angeles. His research interests are in the verification and control of reactive, real-time, hybrid, and probabilistic systems, software verification and programming languages, game theoretic problems in verification, logic, and automata theory.

Prof. Majumdar received the President's Gold Medal from the Indian Institute of Technology and the Leon O. Chua Award from the University of California, Berkeley.

Miodrag Potkonjak (S'86–M'92) received the Ph.D. degree in electrical engineering and computer science from University of California, Berkeley, in 1991.

In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with Computer Science Department at the University of California, Los Angeles (UCLA). His watermarking-based Intellectual Property Protection research formed a basis for the Virtual Socket Initiative Alliance standard. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received the NSF CAREER Award, the OKAWA Foundation Award, the UCLA TRW SEAS Excellence in Teaching Award, and a number of best paper awards.