Software Watermarking as a Proof of Identity: A Study of Zero Knowledge Proof Based Software Watermarking

Balaji Venkatachalam

Department of Computer Science,
Iowa State University, Ames, IA 50011, USA
balaji@cs.iastate.edu

Abstract. Software watermarking has been proposed as a way to prove ownership of software intellectual property in order to contain software piracy. In this paper, we propose a novel watermarking technique based on Zero Knowledge Proofs. The advantages are multi-fold. The watermark recognizer can now be distributed publicly. This helps in watermark being used as a proof for both authorship and authentication of the software. The watermark is shown as a mathematical proof which varies with every run instead of the watermark string as in the previous techniques. This watermarking scheme not only has a high degree of tamper resistance but also allows the protocol to point out the tampered subset of the embedded secret data. We present potential attacks on the protocol and discuss the strength of the watermarking scheme. We present empirical results based on our implementation.

1 Introduction

Software can be easily copied without permission. Protecting software against misuse and illegal copying is an important problem. The actual creator of the software can established his authorship by *software watermarking*. Software watermarking refers to the process of embedding secret data called *watermark* in a software application by the creator of the software so that the authorship of the software can be proven where the presence of the secret data is demonstrated by a watermark recognizer. Since the watermark is secret, only the true author knows its value.

An important consideration in watermarking is protecting the watermark from the adversary. The adversary might tamper with the program and modify or completely remove the watermark so that the watermark recognition would fail; in the existing watermarking schemes, by modifying even one bit of the watermark key, the recognizer would fail to recognize the correct secret data. The knowledge of the secret key to the adversary will render the whole scheme ineffective. These reasons constrain the watermark recognizer from being available in public.

In this paper, we propose a solution based on Zero Knowledge Proofs (ZKPs) that allows the watermark recognizer to be available publicly. In the ZKP based

M. Barni et al. (Eds.): IWDW 2005, LNCS 3710, pp. 299–312, 2005.

watermarking scheme, the presence of the watermark is conveyed to the end user or the adversary without revealing the actual keys by means of a protocol. In other words, the end-user (or an adversary) is convinced about the authenticity of the software but has gained *zero* knowledge about the secret keys.

Zero Knowledge Proofs introduced by Goldwasser, Micali, Rackoff [18], provide a solution in a situation where a *prover* wants to prove its knowledge of the truth of a statement to another party, called the *verifier*. However, the prover wants to convey the its knowledge of the proof without conveying the actual proof. The proof is provided by the *interaction* between the two parties, at the end of which the verifier is convinced of the prover's knowledge of the proof. However, the verifier has not gained any knowledge in the interaction. In our case, we show the watermark by a zero knowledge proof.

Let C be the actual creator of the software s. Let A be a malicious adversary who steals (claims authorship) s. C should prove its true *authorship* of s. If A sells a fake software s' as s to the end-user B, B would like to verify the *authenticity* of s. s can be a large program (for e.g., a text editor) or a smaller software component (for e.g., a spell checker for a text editor). For authentication, the end-users require a recognizer. Existing watermarking techniques provide proof of authorship only. We show the application of zero knowledge based watermarking system to solve both these problems.

We propose two different zero knowledge protocols to achieve these goals and discuss their relative merits and demerits.

Contributions: The main thesis of this paper is that zero knowledge based software watermarking has a number of advantages over the existing watermarking systems including novel applications to the problems of authentication and authorship. The novelty in this paper is the application of ZKP to software watermarking. Since the proof of the watermark is given in zero knowledge the software watermark recognizer can be made public, which is not possible in the existing models of software watermarking.

The goal of the paper is to show the natural connections between ZKPs and software watermarking. We show that the proof is independent of watermarking scheme used. We describe software watermarking using the ZKP of the "quadratic residue problem". We use this ZKP here for its ease of exposition. ZKPs of other languages and other cryptographic protocols can be applied to software watermarking (see discussion in section 8).

This scheme builds robustness to the authorship problem by tamper-proofing and detecting the exact bits that were tampered. In addition, it also provides a solution to the new problem of software authentication, due to the public distribution of the recognizer.

This paper is the first paper in applying cryptographic protocols to *software* watermarking to the best of our knowledge. This claim needs some clarification. There has been work to apply zero knowledge proofs to media watermarking and steganography [3,13,1]. However, the issues of media watermarking are different from software watermarking (for e.g. statistical profiling of static media objects, dynamic nature of software etc.). The current work was done

independently of these results; with the initial goal to improve robustness of the software watermarking schemes. The results we achieve (the computational difficulty for the adversary, public recognizer etc.) are different from the results of the media watermarking papers (more details in section 2).

Roadmap: Section 2 discusses existing watermarking schemes. In section 3, we describe the model and the terminology. We also discuss the problems of authorship and authenticity, and the attacks against these problems. Section 4 provides a specific zero knowledge protocol that we use for software watermarking. The application of ZKP to software watermarking – the two protocols and the related issues are described in sections 5 We discuss the advantages of our scheme in section 6. We describe the experimental results in section 7.2. Section 8 has a discussion on extensions to the scheme described.

2 Related Work

Watermarking media objects has been an active area of research since the 1990s, with a large amount of literature [12,20,6] and many different techniques for watermarking audio, video and images. *Software watermarking* [10] is a relatively recent research topic. Existing software watermarking schemes can be classified on the basis of watermark embedding and recognition techniques. *Static* watermarking [14,22] refers to embedding the watermark in the executable text or data segments of the software application such that it does not change the application semantically. In *dynamic* watermarking schemes [9], the watermark exploits the dynamic change of state of the software program. The instructions that generate the watermark are embedded in the software application. The watermark is generated at run time.

For example, in *dynamic graph-based watermarking* [9] the watermark string is represented as a graph. The instructions to generate this graph are embedded at various locations in the software application. At run time the embedded instructions execute along with the application code and generates the watermark graph on the heap. The encoding of the graph and the location of embedding are unknown to the adversary. During program execution, the time of watermark creation and heap structure vary in every run. The added stealth makes it difficult for the adversary to detect and tamper the watermark.

Sandmark[11] is a Java-based watermarking software that implements many static and dynamic watermarking schemes. Palsberg et al. [21] describe an implementation of the dynamic watermarking scheme [9] to show its practicality.

Media data is static, as opposed to, the dynamic change of state of software. The issues that arise in the software watermarking systems are different from the issues in media watermarking systems [10]. For example, statistical tests on media objects can detect some possible watermarking structures, whereas detecting software watermarks involve analyzing the heap space which is a hard problem [9]. Various papers [3,13,1,2] discuss the application of various zero knowledge proofs to watermark media. These papers deal with the statistical

distinction of an actual proof from a spurious proof, which is a natural problem in media watermarking. However, in the current paper, we are concerned with knowing the secret data and we discuss the computational difficulty to break the ZKP based watermarking system and its practical implications.

3 Models and Notation

In security systems it is essential that the threat model, the trust and power assumptions about the various agents, be made explicit to test the limitations of the system [19]. In this section, we will describe the model and definitions used in this paper.

A *watermark* can be viewed as secret data stored in an application. The secret data could be a string, a number, or other secret data that is hidden in some format in the application. We will call the secret data the *watermark value* and the secret format of the watermark representation its *encoding*. The creator of the software *embeds* the watermark into the software to obtain the *watermarked software*.

For example, in graph encoding of a dynamic watermark the secret data or the watermark value is a number, which is encoded in the form of a graph. The structure of the graph represents the watermark value in some radix. This method of representing the watermark string as a graph is called graph-based watermark encoding. The instructions that generate this watermark graph are embedded in the application program in an appropriate way to obtain a watermarked software.

The existence of the watermark is demonstrated by a watermark *recognizer*. When the watermark is *dynamic*, the recognizer executes the watermarked application; it observes the program execution trace to identify or recognize the embedded watermark. In this paper we propose to enhance the recognizer to facilitate an interactive session for the ZKP.

Prover and Verifier: To describe ZKPs, we need the notion of a prover and a verifier. The *prover* is an agent or entity who claims the *knowledge of the proof of a statement* and tries to prove it. The *verifier* is an agent or entity who tries to learn the proof from the verifier. At the end of the interaction, called a *protocol*, a prover convinces the verifier about his knowledge of the proof (but not any additional knowledge). If the prover does not know the proof, he is called a *cheating* prover. The protocol is designed so that the verifier would not accept the proof of a cheating prover. A *cheating* verifier, is the one who tries to gain knowledge from the prover through the protocol executions.

3.1 Threat Model

The end user of the software is the potential adversary in the software piracy world. In many cases, the end user/adversary also has supervisory privileges on the host machine where the software is executed. That is, we consider an *all powerful* adversary who can observe and modify the software that is available.

The adversary has access only to the watermarked application but does not know the encoding scheme or the location of the watermark within the application. As mentioned before, decoding a dynamic watermark is hard for an adversary. The adversary tampers with the application for two reasons – to learn the watermark value, or to modify or completely remove the watermark so that the watermark recognition would fail. Availability of the watermark recognizer publicly brings in extra difficulty of hiding the watermark, because it allows the adversary to observe the execution and tamper the recognizer. This helps the adversary to gain significant knowledge about the watermark.

The adversary tampering and removing the watermark are called *distortive and subtractive* attacks. The adversary can also add an additional watermark, which is called an *additive* attack. Adding an extra watermark does not interfere with the recognizer in recognizing the existing watermark. Therefore we do not consider additive attacks in this paper. Dispute resolving which arise due to additive attacks are not considered in this work.

3.2 Watermarking Problems and Their Attacks

In this subsection, we describe the main problems of software copy protection addressed in this paper and the attacks on those watermarking systems. Separating the concerns of the problems and the corresponding attacks is helpful, because the attacks on one system (say, authentication) are completely ineffective on the other watermarking system (say, authorship, and vice versa). For the rest of this subsection, let C be the actual creator of the software s, let A be the malicious adversary, and B the end user.

Proof of Authenticity: C creates a software s. The malicious adversary A sells a fake product s' as the original software s. B on buying a software from A, would like to ensure its *authenticity* – the creator of the software is indeed A. If the watermark recognizer of s is available publicly, B can use C's recognizer to test the authenticity. A fake software s' should not be able to cheat and should fail the protocol.

Proof of Authorship: The malicious adversary A sells the original software of C as his own. Now C claims the true authorship, and proves this by showing the presence of his secret data. If A is to prevent the proof of C's claims, he should remove the secret data in s.

In the authentication problem, we need a public recognizer. The authorship problem does not need a public recognizer but the watermark recognition is done in the presence of a trusted third party. The attack for the proof of authorship is the removal of watermark in the original software, which was not relevant in the authentication problem. The adversary needs to replicate the watermark in s' to provide a proof of authenticity.

4 ZKP of Quadratic Residue Problem

The ZKP of quadratic residue problem [18,15] is described as follows. Consider a number n that is a product of 2 large primes. The size of n, $|n| = b$ bits, is a security parameter. Initially, the prover chooses k-random numbers, $s_1, s_2, \ldots, s_k$ in Z_n^* (Z_n^* is the *multiplicative field* relative to n, that is, $Z_n^* = \{x | 1 \leq x \leq n \wedge gcd(x, n) = 1\}$). The numbers $v_1, v_2, \ldots, v_k$ are chosen such that $v_i = \frac{1}{s_i^2}$ (That is, v_i is the inverse of the square of s_i in the field Z_n^*).

At the start of the protocol, the modulus, n,the number of residues k, and the inverses, $v_1, \ldots, v_k$, are known to both the parties. However, $s_1, \ldots, s_k$ are known only to the prover. This protocol is pictorially represented in Figure 1.

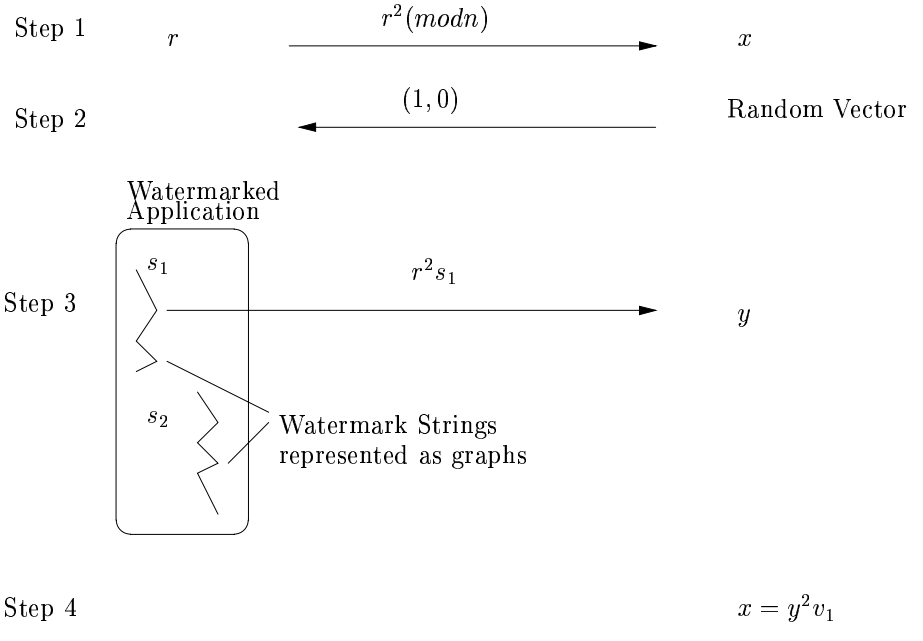

Fig. 1. Pictorial representation of Protocol 1 with a dynamic graph based encoding

Protocol:

1. Prover picks a random number $r \in [0, n)$ and sends $x = r^2 (\mathrm{mod}\, n)$ to verifier
2. Verifier sends a random bit vector $(e_1, e_2, \ldots, e_k)$ to prover, (that is $e_i \in \{0, 1\}$)
3. Prover sends to verifier
$$y = r \prod_{e_i=1} s_i (\mathrm{mod}\, n)$$
4. Verifier computes that
$$z = y^2 \prod_{e_i=1} v_i (\mathrm{mod}\, n)$$
and checks that $z = x$.

Note that, the proof varies in every run based on the values of r and e_i. The verifier does not know the value of r in an execution and the prover does not have control over the random bits, e_i. It is hard for the verifier to compute the values of r and the residues. The computation needed for the verifier – the multiplication in step 4 – is easy.

5 ZKP for Watermarking

In this section, we present two zero knowledge protocols to show the application of ZKPs to software watermarking. The first protocol is based on the natural connection between software authentication and smart card authentication. This protocol is expository and helps in understanding the modeling issues and the threat model better. This also helps us to understand the advantages of ZKP over traditional watermarking systems. Protocol 2 intends to resolve the issues of Protocol 1. While these are not the most optimal protocol or the best solution for all software applications, but this ZKP was chosen for the ease of exposition.

Fiat and Shamir [15] show the application of ZKPs to smart card authentication. The smart-card is the prover which proves its authenticity to the end-user, who is the verifier, and the smart-card reader supports the interaction. Analogously, for software watermarking, the software is the prover (who has access to secret data), the watermark recognizer helps in the interaction, and the end-user is the verifier. This correspondence leads to Protocol 1.

5.1 Protocol 1:

The quadratic residues $s_1, \ldots, s_k$ are stored in the software application as the secret key (watermark). The creator of the software embeds these in the application using any of the dynamic watermarking schemes. The numbers n and $v_1, \ldots, v_k$ are embedded in the public recognizer. (They can be generated by a random function [15]). The recognizer provides the interface between the prover (the application) and the verifier (the user). During the watermark recognition phase, the recognizer performs the computation needed for the protocol. The recognizer performs the following operations for each of the steps of the protocol – it receives the random bits e_i from the user, it reads s_i and computes y etc.

Advantages: It is easy to see that Protocol 1 solves the authorship problem (when the recognizer is not available in public). The relative merits of this scheme over the existing watermarking techniques are discussed in the next section. When the recognizer is distributed publicly to solve the authentication problem we have the following issues.

Attacks on Protocol 1: The adversary can fix the random string in round 1 and observe the change of state of the recognizer. These attacks are considered in the context of smart-cards – *Reseting attack* and *Concurrent Reset – CR1, CR2 attacks* [8] respectively. The solutions proposed in Bellare et al. [8] to overcome these attacks can be used in the context of watermarking. In general, any attack on the protocol is similar to attack on a general ZKP and the solutions to those attacks apply to watermarking too.

Drawback of Protocol 1: Although Protocol 1 provides the most natural mapping for a prover and verifier among the candidate set of software creator, application software, and the end user, as we discussed before, the recognizer being available publicly helps the adversary. In addition to the attacks on the protocol the adversary can observe changes and tamper with the recognizer. The adversary can potentially learn the s_i values when there will be no other secrets left. To prevent these attacks on the recognizer, we can use an *obfuscator* [10,5].

Barak, Goldreich et al. [7] show the impossibility of obfuscating programs. However, their impossibility result is for a generic obfuscator which obfuscates all programs and does not leak even one bit of information. Obfuscation of a smaller program (the recognizer) is more difficult (than a very large program) owing to the smaller space of combinatorial richness. On the other hand, there is some recent work towards efficient [5] and hardware-based obfuscators [4] which are very hard to crack. Obfuscation will be easier on a platform like *Palladium*. We do not know the practicality of "breaking" an obfuscator or the use of obfuscation to protect the recognizer.

5.2 Protocol 2

We now present an alternate protocol, to address the drawback of Protocol 1. The basic premise in the design of Protocol 2 is that a compromise of the inverses (v_i), is less harmful than a compromise of the residues (s_i). The following protocol allows for the authentication to happen over the Internet. The end user authenticates the software using the (server of the) creator of the software.

The creator picks numbers, n, $s_1, \ldots, s_k$ and $v_1, \ldots, v_k$ as before. The creator embeds the v_i values in the application (instead of the s_i as before), and the protocol is run with the creator of the software as the prover and the software as the verifier and the recognizer helping in the interaction between the prover and verifier. The public recognizer, which can be obtained from the creator, neither contains the residues nor the inverses. The protocol is as before – the software creator sends x, the recognizer receives the random vector from the end-user etc.

Advantages: Protocol 2 augments the existing watermarking schemes. Hence it is easy to see that Protocol 2 solves the authorship problem. Protocol 2 also solves the authentication problem as the user can verify z which changes with every run. Since r is chosen by the creator and the adversary cannot control the random source, the Resettable, CR1 & CR2 attacks are *not possible*. The adversary can observe the data in the network stream between the recognizer and the creator. But this provides only x and y which do not help the adversary to learn the secrets.

If the recognizer is not obfuscated, the adversary can see the change of state of the recognizer and the location in the application where the watermark is stored. (Recall that removal of watermark is not a solution for the authentication problem.) At best, the adversary will only learn the values of v_is from all this information. For the adversary computing the s_i values is hard.

If an adversary sells a fake software s' as the original software s, the end-user can verify the authenticity of s' by using the software creator's publicly available recognizer. Therefore s' not only needs to know all the data-structures but it should also replicate them in s' (also ensuring that, the data-structures of s' are not (mis)recognized as a watermark).

The creator of the software can pick different values $(v_1, \ldots, v_k, s_1, \ldots, s_k)$ for different copies of the software. We assume that the watermark recognizer which is available from the creator's web-page through a secure channel, is parameterized for every copy (say, using the registration key). Since the creator of the software is involved in the watermarking protocol, it can detect any suspicious protocol requests, (for e.g. too many requests from the same client, too many requests with the same v_i values etc.) and detect an attack.

6 Advantages

In this section, we discuss the advantages of using Protocol 2 for software watermarking. It is easy to see that most of the advantages for the proof of authorship hold for Protocol 1 too.

Proof of Authorship: When the recognizer is not distributed publicly using ZKPs for watermarking has a few advantages to solve the problem of authorship. In the current watermarking schemes, there is only one query for the watermark which displays the watermark string. Tampering even one bit of the watermark will destroy the watermark and the true creator cannot prove his claim of authorship.

With the zero knowledge protocol, the claim for authorship is proved (and substantiated) in multiple ways (1) When one or a few bits are tampered, only some of the residues are affected, and the watermarking protocol would work for the queries involving other residues. Suppose the residue v_i is tampered, v_i is used (in Step 4) only when $e_i = 1$. Therefore, the tampering does not affect the correctness of the proof for all random vectors generated in Step 2 where $e_i = 0$. There are 2^k possible distinct vectors that can be generated in Step 2. When f bits are tampered, in the worst case, f different residues v_i are tampered. These tampered residues do not affect the validity of the proof for all vectors for which the corresponding bit $e_i = 0$. The remaining $(k - f)$ bits can take on any values. We have $2^{(k-f)}$ vectors in which the tampered residues would be unused. The fraction of valid proof vectors as a fraction of all the 2^k vectors then is $\frac{2^{k-f}}{2^k}$ which is $1/2^f$. Note that, we are counting only the number of queries and not the number of different proofs shown (which are larger due to the choice of random numbers). (2) Based on the successful queries, the creator can exactly point to the bits that are tampered and the other bits that are tampered. (3) The creator has knowledge of the numbers s_i which are not easily computable by the adversary. The knowledge of the secret keys, hardness of computing the values and the mathematical proof for the bits tampered strongly support the claim for authorship.

Collberg and Thomborson watermarking scheme [9,21], is not resilient to tampering. Even when one bit of the string is tampered, the watermark recognizer will fail to retrieve the correct watermark. When the same key is replicated k times, there is only one query that returns same the watermark string. Once the secret value is known the scheme becomes ineffective [19]. Moreover, there is no (mathematical) proof for the third-party to believe the claim.

In terms of hardness of learning the secret watermark values, we achieve the best of both worlds – the stealth of the dynamic watermarks (difficulty of locating and decoding the watermark) and the computational difficulty of learning the secret values from the proof.

Proof of Authentication: One of the main advantages of using zero knowledge proofs for watermarking is the public distribution of the recognizer. In the existing watermarking schemes, the embedded key is shown during the watermark recognition process. This is because the watermarking process is symmetric. This hinders the distribution of watermark recognizer. Knowledge of the embedded key allows the adversary to create a "dummy recognizer", which always outputs the same key during all runs for watermark recognition. The adversary can also claim the knowledge of the key to the arbitrator and therefore claim to be the actual creator of the software. The original creator cannot prove his authorship as the creator does not have extra knowledge over the adversary.

With the ZKP watermarking scheme, however, the creator of the software can safely distribute/share the watermark recognizer to/with the end users. This is possible since the recognition protocol does not reveal the values $s_1, \ldots, s_k$. The information revealed are x and y (which change in every run). This still makes it computationally difficult to derive s_is. This is similar to the case of smart-cards where both smart-card and the smart-card verifier are publicly available. The computational difficulty of reversing the proof makes it hard for the adversary to break this scheme. As mentioned in the previous section, since the creator is involved in the verification process it is difficult for the adversary to cheat. As the watermark recognizer is available to the user, the watermark can be used as a proof of authenticity of the software.

Computation and Storage costs: We now show that the computation and the extra storage space needed for the protocol are not very high. The total number of bits that need to be encoded depends on the number of stored values, k, and the size of each value, b. We cab choose these values to exceed an acceptable vale of robustness for the watermarking scheme.

Let us consider $k = 20$, and n and the residues to be $512-$bit numbers. The total embedded key size is about $20 \times 512 - \text{bits} \approx 1.3KB$. The average number of multiplications needed per recognition is 10. The communication complexity (the number of bits used during the interaction) is about $1000-$bits per proof – 512 bits for each of x and y and the random vector $(e_1, \ldots, e_{20})$. This protocol computation is much cheaper than RSA where many more multiplications need to be performed by each party[15].

If the watermark string is about 50 characters long, the number of stored bits in a traditional dynamic watermarking method such as Sandmark is about

400 bytes. The space overhead compared to Sandmark is a factor of about 2-3 times. If only one 512-bit number (9-character string) is embedded, the space overhead is about 10 times. The computation overhead is about 10 extra multiplications. Note that even for a medium sized application, the 1.3KB watermark storage space and the computation is dwarfed by the application's own requirements.

7 Experimental Evaluation

7.1 Design and Implementation

The security parameters, n – the product of two large primes, k – and the number of quadratic residues can be chosen according to the degree of security needed for the protocol. In the Collberg-Thomborson model, only one number is stored, and it is stored as a graph. However, we need to encode and embed the residues of numbers. In ZKP based watermarking scheme, in addition to recognizing the watermark (as done in the earlier schemes), the recognizer acts as the interface between the prover and the verifier.

We extend Sandmark [11] class hierarchy and methods. We reuse the graph-based encoding methods and represent each quadratic residue as a separate graph (each with a different encoding radix). Since each residue is stored as a different graph knowing one graph to obtain one residue does not yield any information about the encoding of the other graphs. Each of these graphs is split into a number of parts so that, even if some subgraph of one of the graphs is identified, the entire watermark is not revealed. In fact even the graph to which this particular subgraph belongs to, is not known.

7.2 Empirical Results

In this section, we show that the experimental results follow the theoretical expectations. We find that the amount of heap space used and byte-codes size of the program increase linearly with the number of bits of the quadratic residue.

Building a recognizer and the time to embed the residues in the program does not impact the normal program execution in this model, because it is a one time cost (similar to compilation) incurred by the creator of the software.

The time to recognize the watermark depends on the program flow and the watermark data structure identification instructions. We observe that, once all the watermark data structures are recognized, the time to compute the product (for Step 3 of the protocol) is negligible. The degree of tamper resistance would also be as expected theoretically. The two factors that would affect in a program and which could possibly provide clues to an adversary are the heap space usage and the increase in the static program size.

To benchmark the results, we ran the watermarking process on a simple test program that does not allocate any memory for any object other than the watermark on the heap. We calculate the increase in the amount of heap space as

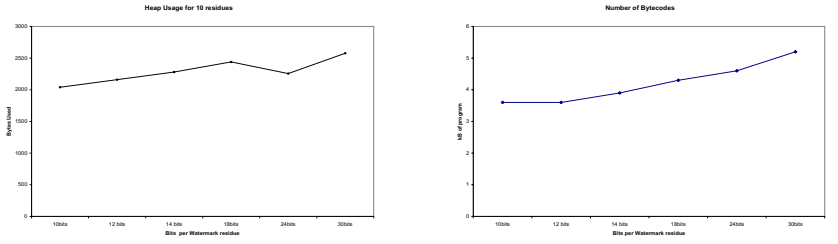

Fig. 2. Experimental Results: **(a)** Heap Space Overhead **(b)** Program Size Overhead

the number of bits of the quadratic residues increases. To obtain the worst case usage, we set all bits (e_i) to 1, and the resulting heap space sizes are averaged over experiments. This would give us the exact memory requirements of the watermark. This is a reasonable assumption because the amount of heap space used by the program is only for the watermark. The increase in memory usage due to the watermark is additive and independent of the program, and therefore we should observe similar heap overhead for other programs as well. Using larger programs for benchmarking would not give us consistent values across runs as the heap usage may vary due to program flow, garbage collection (by the virtual machine) and other program idiosyncrasies. Similarly we measure the increase in program size.

We embedded 10 residues. We repeat the experiment for residues of various sizes as shown in the graphs in Figures 2. (a) and 2. (b). We observe that the increase in program size is linear in the number of bits used to represent the residues (Figure 2. (a)). The heap space overhead also grows linearly with the number of bits in the residues as seen in Figure 2. (b).

Since we are using existing watermark encoding techniques, it is a reasonable conclusion that the overhead of space and heap usage are linear in the number of residues.

8 Discussion

As we mentioned before, we used ZKP of the quadratic residue problem mainly for the ease of exposition. More robust protocols can be used for software watermarking. For e.g. $v = \frac{a}{s^2} + b (\text{mod } n)$ for some $a, b \in Z_n^*$ would be harder function to break than the one described [16]. ZKP of other languages,interactive proofs, one-way functions and public-key schemes can also be used. For instance, the ZKP of *Graph non-Isomorphism* problem [17]. This is well suited for software watermarking as graphs are good, well-obfuscatable watermark representations.

Other watermark or birthmark encoding and embedding methods can also be used for watermarking.

The zero knowledge proof is independent of the encoding and embedding. As mentioned above various encoding schemes and cryptographic protocols can be used for watermarking. The choice of encoding scheme and the cryptographic protocol is an interesting design issue. For example, for a small application, a

graph based representation may be a bad choice. This is since the graph based representations are generally large, and might reveal more than they hide! An encoding and protocol to suit the application attributes can be chosen.

As noted in section 5, any improvement in obfuscation technology will help in securing the watermark. In the absence of obfuscation an extremely powerful adversary (who can read, modify, and tamper the entire watermark), *no* watermarking scheme is completely secure! However, our watermarking scheme has a higher degree of tamper resistance and prevents the creation of new identities (new pairs of values for $s_1, \ldots, s_k$ and the corresponding $v_1, \ldots, v_k$) easily.

9 Conclusions

We have presented a new watermarking paradigm for software watermarking based on zero knowledge proof systems (ZKPs). The proof of the watermark is provided in zero-knowledge, and therefore the recognizer can be distributed publicly. We show two protocols for problems of authenticity and authorship and discuss attacks on protocols. Both the protocols work best when obfuscated. However, when used without obfuscation, it is computationally hard to obtain all the secrets from the values learnt in Protocol 2.

With our watermarking scheme we obtain best of both worlds – stealth of dynamic watermarks and computational hardness of ZKPs, which makes it hard for the adversary to learn the watermark keys. One of the main advantages in the proof of authorship is in tamper detection and resistance. Moreover, a mathematical proof of which bits are tampered can be provided.

This watermarking environment was implemented as an extension to Sandmark. The memory usage of the heap and watermarking code size overheads are characterized experimentally. Both the heap space and byte-code size overhead are linear in the number of bits to represent the embedded secret data as predicted theoretically.

This is the first application of a cryptographic protocol for software watermarking to the best of our knowledge.

Acknowledgments. This work was supported by AFRL/OUSD under contract number F33615-02-C-1238 through Software Protection Initiative program. Thanks to Akhilesh Tyagi for some useful discussions, suggestions and help in editing some early drafts of the manuscript. Thanks also to Curt Clifton for some helpful feedback on the manuscript and to the anonymous reviewers for their critical comments which have helped in improving the presentation.

References

1. André Adelsbach, Stefan Katzenbeisser, Ahmad-Reza Sadeghi, "Watermark detection with zero-knowledge disclosure", *Multimedia Systems*, 9(3), pp. 266-278 2003.
2. André Adelsbach, Ahmad-Reza Sadeghi, "Zero-Knowledge Watermark Detection and Proof of Ownership", *Information Hiding 2001*, pp. 273-288, 2001.

3. André Adelsbach, Stefan Katzenbeisser and Ahmad-Reza Sadeghi, "Cryptography Meets Watermarking: Detecting Watermarks with Minimal- or Zero-Knowledge Disclosure", *XI European Signal Processing Conference*, Volume I, pp. 446-449.
4. M. Gomathisankaran, A. Tyagi, "3D Obfuscation Architecture", *under submission*, 2005.
5. D. Aucsmith, "Tamper Resistant Software: An Implementation", *Information Hiding 1996*, pp. 317-333, 1996.
6. Mauro Barni, Franco Bartolini, "Watermarking systems engineering : enabling digital assets security and other applications", *Marcel Dekker*, 2004.
7. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, Ke Yang, "On the (Im)possibility of Obfuscating Programs", *CRYPTO 2001*, pp. 1-18, 2001.
8. Mihir Bellare, Marc Fischlin, Shafi Goldwasser, Silvio Micali, "Identification Protocols Secure against Reset Attacks", *EUROCRYPT 2001*, pp, 495–511, 2001.
9. Christian Collberg, Clark Thomborson, "Software watermarking: Models & dynamic embeddings", *POPL'99*, 1999.
10. Christian S. Collberg, Clark Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection", *IEEE Transactions on Software Engineering*, 28:8, pp. 735-746, August 2002.
11. Christian Collberg, Ginger Myles, Andrew Huntwork, "SandMark - A Tool for Software Protection Research", *IEEE Security and Privacy*, Vol. 1, Num. 4, July/August 2003.
12. Ingemar Cox, Matthew Miller, Jeffrey Bloom, "Digital watermarking", *Morgan Kaufmann*, 2002.
13. Scott Craver, "Zero Knowledge Watermark Detection", *Information Hiding 1999*, pp. 101-116, 1999.
14. R. L. Davidson and N. Myhrvold,, "Method and system for generating and auditing a signature for a computer program". *US Patent 5,559,884*, Sept. 1996.
15. A. Fiat, A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", *CRYPTO '86*, LNCS 263, pp. 186-194.
16. A. Fiat, U. Fiege A. Shamir, "Zero-Knowledge Proofs of Identity", *STOC 87*, 1987.
17. Oded Goldreich, "Foundations of Cryptography: Basic Tools", *Cambridge University Press*, 2001.
18. S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal of Computing*, 18(1), pp. 186-208, February 1989.
19. Stefan Katzenbeisser, "On the Integration of Watermarks and Cryptography", *Digital Watermarking 2003*, pp. 50-60, 2003.
20. Stefan Katzenbeisser, Fabien A.P. Petitcolas(editors), "Information hiding techniques for steganography and digital watermarking", *Artech House*, 2000.
21. Palsberg, J., Krishnaswamy, S., Minseok, K., Ma, D., Shao, Q., and Zhang, Y, "Experience with software watermarking", ACSAC '00, pp. 308–316, 2000.
22. R. Venkatesan, V. Vazirani, S. Sinha, "A Graph Theoretic Approach to Software Watermarking", *Information Hiding Workshop 2001*, LNCS vol. 2137, 2001.