An *ID*-Based Watermarking Scheme for Java Programs

Zheng Yuan[1,2], Qiaoyan Wen[1], Wenling Wu[3], and Qing Zhang[1]

[1] School of Science, Beijing University of Posts and Telecommunications, Beijing, 100876, P.R.China
[2] Beijing Electronic Science and Technology Institute, Beijing, 100036, P.R.China
sxzyyz5318@163.com
[3] State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing, 100080, P.R.China

Abstract. In this paper, we propose an identities(*ID*) based watermarking scheme for Java programs. In our scheme, the watermark is generated by participants' identities, embedded via the watermarked opaque predicates, and verified using zero-knowledge proof. We also present a construction of a family of opaque predicates by Legendre symbol, which is resilient, cheap, and stealthy. The order of the watermark is encoded and embedded into the watermarked opaque predicates, and the watermarked opaque predicates are treated as threads of a Java program. Thus, the embedded watermark is dynamic and secure against all usual types of watermarks algorithms attacks and watermarks protocols attacks, and also secure against static and dynamic attacks.

Keywords: Java programs, watermarked opaque predicates, *ID*, aggregate signature, watermarking scheme.

1 Introduction

As the international network is becoming faster and more widespread, there has been an increased need for digital rights management, especially the protection of intellectual property rights over softwares becomes a paramount issue. Watermarking schemes which had been developed to assert authorship or ownership control over digital works such as images, video or audio, are being extended to cover software objects. However, most software watermarking schemes fail because it is easy to tamper or delete software watermarks, keeping the overall semantics of the program constant.

A software watermark should fulfill the following criteria:

1. The watermark should be **robust**, i.e., it should be resistant to watermarks algorithms attacks, such as software transforms which preserve the meaning of the program.
2. The watermark should be **invisible**, i.e., it should only be detectable by using special forensic software.

3. The watermark should be **secure** [13], i.e., it should be resistant to malicious attacks, such as watermarks protocols attacks , statical attacks, dynamic attacks. Additional, the purchaser also should have no way of removing the watermark, even if provided with the forensic software.
4. The watermark should have **authentication**, i.e., it should contain information which validate original creators, ownership and purchasers, etc.

Here we give a brief overview of previous work done in the field of software watermarking. The first formal software watermarking approach was described in paper [1], in which a watermark is embedded into a program by rearranging the order in which basic blocks of the program were arranged. A method of encoding the watermark inside a dummy method in the form of opcodes is described in paper [2], but using simply applying semantics-preserving transforms can easily damage or remove watermarks in paper [1, 2] without changing the semantics of the programs. Paper [3] firstly described a dynamical watermarks scheme, in which the watermark is constructed in the form of a graph dynamically during the program execution, the scheme is against static-time analysis, but it is not very difficult to be recognized which parts of the software belong to the original codes and which belong to the dynamic watermark generating code. In paper [4], a final dynamic experimental watermarking method relies on multi-threading to encode watermarks, and all encoded watermarks are in the choice of execution of basic blocks depending on the input pattern of the program, but for every encoded watermarks bit, the size of the software increases by nearly 1 KB, so this is a low capacity scheme.

Opaque predicates were first presented in paper [8] as a technique to aid in code obfuscation. Later opaque predicates were incorporated in Java programs watermarking technique proposed in paper [10]. Informally, the inserted opaque predicates make it difficult for an adversary to analyze the control-flow of the programs. This makes it more difficult to identify what certain portions of the programs are superfluous, but the opaque predicate library must remain secret, if an adversary knows even a few of the predicates he may be able to identify and remove them from the programs. Regrettably, the method [10] hasn't considered cryptographically secure, so it shouldn't stand against copy attacks and ambiguity attacks [5].

In this paper, we propose an identities (ID_i) of participants based watermarking scheme. In our scheme, the identities (ID_i) and a Java program J are signed using extended aggregate signatures scheme, then the concatenation of signatures is regarded as the watermark information ω, thus our watermarking scheme can stand against copy attacks, passive ambiguity attacks, and others of watermarked protocols attacks. We also construct a family of watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ using Legendre symbol, and the bits of ω and some appended information are encoded inside these predicates. Our watermarked opaque predicates have better resilience to resist static and dynamic attacks. Because the watermarked opaque predicates are treated as threads of J , they are difficult to be distinguished from original codes. Additionally, our watermarking scheme has a lower data bloating.

The remainder of our paper are structured as follows. Section **2** describes the basic of knowledge, including construction of a family of watermarked opaque predicates and watermark determining. Section **3** introduces our watermarking scheme, including encoding watermark algorithms, embedding watermark algorithms and detecting watermark algorithms. Section **4** analyzes the security of our watermarking scheme for Java programs. Finally, conclusions are presented in section **5**.

2 The Basic of Knowledge

2.1 Notations

Denote by J a Java program, which is available for manipulation in the current state, denote by ω the watermark information which is encoded and embedded into the program J, and denote by K embedding watermark key. Let $\boldsymbol{E}$ be a watermark embedding function, i.e. $\boldsymbol{E}$: $(J, \omega, K) \rightarrow J_\omega$, and let $\boldsymbol{D}$ be a watermark detecting function, i.e. $\boldsymbol{D}$: $(J_\omega, K) \rightarrow \tilde{\omega}$, if $Ver_{\tilde{\omega}=\omega} = True$, J_ω is accepted as legal, otherwise it is accepted as illegal.

For $i \in (1, 2, ..., m)$, let ID_i be the identity of the *i-th* participant, let s_i be the signature of ID_i. Write $< ID_1, ID_2, ..., ID_r >$ for an ordered sequence of r elements $ID_1, ID_2, ...,$ and ID_r. If $L_r =< ID_1, ID_2, ..., ID_r >$ is some finite sequence, then $< L_r, \alpha >=< ID_1, ID_2, ..., ID_r, \alpha >$ is also a finite sequence.

2.2 Constructing Opaque Predicates

According to the interleaving semantics, n statements in a parallel programs can be executed in $n!$ different ways, so they are more difficult to analyze statically than their sequential counterparts. It is well-known that parallel regions are constructed by threads in Java, Java's threads have two very useful properties to be used for obfuscation: one is that their scheduling policy is not specified strictly by the language specification, hence it will depend on the implementation, another is that the actual scheduling of a thread will depend on asynchronous events generated by user interaction, network traffic, etc., so we will use these observations combining with watermarking technique to create highly resilient watermarked opaque predicates.

Definition 1 (Watermarked Opaque Constructs). *From paper* [8]*, A variable V is opaque at a point* o *in a program, if V has a property* p *at* o *which is known at encoding and embedding watermark time. We write this as* V_o^{p} *if* p *is clear from context.*

Definition 2 (Watermarked Opaque Predicate). *For* $j \in (1, 2, ..., n)$*, a watermarked predicate* $\boldsymbol{O}_j$ *is opaque at program point* o *if and only if its outcome is previously known to the obfuscator (or watermark embedder) at encoding and embedding watermark time, but which is difficult for the deobfuscator to deduce at encoding and embedding watermark time.*

If $\mathbf{O}_j$ always evaluates to *True* /*False* at point o, we write $(\boldsymbol{O}_j)_o{}^T/(\boldsymbol{O}_j)_o{}^F$; if $\mathbf{O}_j$ sometimes evaluates to *True* and sometimes evaluates to *False* at point o, we write $(\boldsymbol{O}_j)_o{}^?$.

Definition 3 (Core of Watermarked Opaque Predicates). *Let A be the information which is contained in the watermarked opaque predicate $\boldsymbol{O}_j$, If the watermark ω is encoded in A, we define A as "core" of $\boldsymbol{O}_j$.*

For $j \in (1, 2, ..., n)$, "core" perhaps in the constants of $\mathbf{O}_j$, or perhaps in the rank within $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$.

Definition 4 (Trapdoor Watermarked Opaque Predicate).
For $j \in (1, 2, ..., n)$, let k_j be the secret key of a opaque predicates O_j. A trapdoor opaque predicate $\boldsymbol{O}_j$ is one that is difficult to be determined at encoding and embedding watermark time if its secret key k_j is unknown, but it is easy to be determined by k_j.

Constructing a family of Watermarked Opaque Predicates. To resist the attacks, the number of opaque predicates O_j must be large enough. One way of making O_j larger is using parametrized predicates(see paper [10]). This paper utilizes Legendre symbol [11] to construct a family of watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$.

Let Legendre symbol of module a prime p be $(\frac{d}{p})$.

1. Let $p = 2^t x + 1, t \geqslant 3, 2 \nmid x$, choose $b \in Z$, compute $(\frac{d}{p})$, and the corresponding family of watermarked opaque predicates is:

$$(\frac{d}{p}) = 1. \tag{1}$$

   In eq.(1), if the bit of encoded watermark information is 1, we choose b to be satisfied with $(\frac{b}{p}) = -1$; else if the bit is 0, choose b to be satisfied with $(\frac{b}{p}) = 1$. Inversely, as also, these are considered as the "core" of our watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$.
2. Let $p = 4x + 3$, compute $(\frac{d}{p})$, and the corresponding family of watermarked opaque predicates is:

$$(\frac{d}{p}) = 1. \tag{2}$$

3. Let $p = 8x + 5$, compute $(\frac{d}{p})$, and the corresponding family of watermarked opaque predicates is:

$$(\frac{d}{p}) = 1. \tag{3}$$

A family of watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ is parametrized by a given prime p above. This parameter can be generated by picking random values of $x \in Z, t \in Z$, and then testing if the resulting p is prime. The value of the variable d and b should be determinable at run-time only. Individual predicate should be hard to be resolved.

The watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ are secret.

2.3 Determining the Watermark

In this paper, a Java program J and identities (ID_i) are signed by participants using extended aggregate signatures scheme in turn, then the concatenation of signatures is regarded as the watermark information ω. The participants include all programmers, copyright holder, and purchasers, etc.

Aggregate signature is firstly proposed in paper [6], paper [7] presented an extended aggregate signature. Our extended aggregate signature schemes is as approximately same as paper [7] except our bilinear mapping e being the mapping from elliptical cyclic $\mathbf{G}_1$ to finite field $\mathbf{G}_2$.

System Parameters. Suppose that a trusted central authority(CA) is responsible for generating the system parameters, as follows:

- Choose $\mathbf{G}_1$ as a elliptic curve additive group of prime order q, $P \in \mathbf{G}_1$ as a generator of $\mathbf{G}_1$, and $\mathbf{G}_2$ as a cyclic multiplicative group of the same order.
- Choose e: $\mathbf{G}_1 \times \mathbf{G}_1 \mapsto \mathbf{G}_2$ as a bilinear mapping.
- Choose two secure strong hash functions : h: $(0,1)^* \mapsto \mathbf{Z}_q$; H: $(0,1)^* \mapsto \mathbf{G}_1$.

Then CA declares parameters $< \mathbf{G}_1, \mathbf{G}_2, e, P, h, H >$.

For $i \in (1, 2, ..., m)$, each participant $\boldsymbol{ID}_i$ generates his(her) private key: $x_i \leftarrow h(ID_i), x_i \in \boldsymbol{Z}_q$. and his(her) public key: $y_i \leftarrow x_i P \in \mathbf{G}_1$.

Extended Aggregate Signature Schemes *S*. For $i \in (1, 2, ..., m)$, a Java program J is signed by every participants ID_i, the order of signatures is passed from ID_i to ID_{i+1}, the signature method is extended aggregate signature scheme, as follow

- Sign s_i: for $i \in (1, 2, ..., m)$, let $L_r =< ID_1, ID_2, ..., ID_r >$, $(1 \leq r < m)$, let $s_0 = (0, 0) \in \mathbf{G}_1$, and recursive functions s_i be:

$$s_i = (x_i H(J, L_r) + s_{i-1}) \in \mathbf{G}_1. \tag{4}$$

  The signature of ID_i on the Java program J is $< s_i, L_i >$.
- Verify: ID_{i+1} accepts the signature $< s_i, L_i >$ of ID_i as valid if and only if $e(s_i, P) = \prod_{r=1}^{i} e(H(J, L_r), y_r)$.

Value of Watermarks For $i \in (1, 2, ..., m)$, let $s_i = (s_{ix}, s_{iy})$ as watermarks information, i.e.

$$\omega \leftarrow < (s_{1x}, s_{1y}), (s_{2x}, s_{2y}), ..., (s_{mx}, s_{my}) > \tag{5}$$

3 Watermarking Scheme P_J

Select $g \in \boldsymbol{Z}$ randomly, and decompose the program J into $(g+1)$ branching segments with points $\{o_f\}_{f=1}^{f=g}$, these segments are $\{J_0, J_1, J_2, ..., J_g\}$.

Use the embedding watermark key K to insert ω into J by the embedding watermark function$\boldsymbol{E}$.

The secret parameters are K, the order of ω, the watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$, the opaque predicates's key k_j, and each participant's private key x_i, and the public parameters are each participant's public key y_i.

3.1 Encoding Watermark Algorithms

1. Choose a number of dual data randomly, such as, $(a_{1x}, b_{1y}), ..., (a_{lx}, b_{ly}) \in Z_q$.
2. Append these dual data behind ω:

$$\omega' \leftarrow < (s_{1x}, s_{1y}), ..., (s_{mx}, s_{my}), (a_{1x}, b_{1y}), ..., (a_{lx}, b_{ly}) > \tag{6}$$

3. Choose an invertible encode permutation σ, and rearrange ω' in a bits sequence by permutation σ :

$$\omega'_\sigma \leftarrow \sigma(\omega') \in (0, 1)^*. \tag{7}$$

4. Encode the bits of ω'_σ in "core" of our opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$.
   In eq.(1), if the bit of ω'_σ is 1, we choose b to be satisfied with $(\frac{b}{p}) = -1$; else if the bit of ω'_σ is 0, choose b to be satisfied with $(\frac{b}{p}) = 1$, this is regarded as encoding rule R_1.

Remark 1. Let the number of bits encodable with our predicates $\boldsymbol{O}_j$ be $N(\boldsymbol{O}_j)$, in any case, the bits length of ω'_σ is satisfied with

$$|\omega'_\sigma| \leq \sum_{f=1}^{g} N(\boldsymbol{O}_j)_f - g \log 2^g \tag{8}$$

(see paper [10]).

3.2 Embedding Watermark Algorithms

For $j \in (1, 2, ..., n), f \in (1, 2, ..., g), c \in (1, 2, ..., h)$, if a number of transformations T_c always evaluate a predicate $(\boldsymbol{O}_j)$ to *True* at point o_f, then append $\wedge(\boldsymbol{O}_j)_{o_f}^T$ to the branching condition. Otherwise, if the transformations always evaluate a predicate $(\boldsymbol{O}_j)$ to *False* at point o_f, then append $\vee(\boldsymbol{O}_j)_{o_f}^F$ to the branching condition, these would not change the final value of the branching condition. This can be regarded as embedding watermarking rule R_2.

Remark 2. The permutation σ, the rule R_1 and the rule R_2 are regarded as the embedding watermark key K, i.e. $K \leftarrow \sigma \otimes R_1 \otimes R_2$,

3.3 Detecting Watermark Algorithms

1. For $j \in (1, 2, ..., n), c \in (1, 2, ..., h)$, extract watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ from a number of transformations T_c using the watermarked opaque predicates's key k_j.
2. Decode and get the bits of $\tilde{\omega}'_\sigma$ from the predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ with R_2^{-1}.
3. Compute and choose (σ^{-1} is the inverse permutation of σ):

$$\tilde{\omega} \leftarrow \tilde{\omega}' \leftarrow \sigma^{-1}(\tilde{\omega}'_\sigma) \tag{9}$$

4. Verify: for $i \in (1, 2, ..., m)$, if each

$$e(s_i, P) = \prod_{r=1}^{i} e(H(J, L_r), y_r) \tag{10}$$

is true, verifier accepts J_ω as legal; otherwise, accepts J_ω as illegal.

4 Security

Theorem 1. *Our watermarking scheme P_J is secure against passive ambiguity attacks and copy attacks.*

Proof. The DHP in $\mathbf{G}_1$ is hard, so our extended aggregate signatures scheme S is secure against existential forgery on messages or sequence of identities. This assertion is shown in Theorem 1 of [7] and in Theorem 4.5 of [13].

Let the length of J be n, and y_i be the public key of ID_i. Named an adversary Alice. We treat passive ambiguity attacks and copy attacks separately.

CASE 1. Suppose that Alice can (t, ε)-break our scheme P_J using passive ambiguity attacks, here, ε is attack successfully probability and t is a polynomial time. We will deduce a contradiction.

Construct a signature forging algorithm $Forg1_n$ in the following manner:

1. For $i \in (1, 2, ..., m)$, Alice forges ID_i's signature s_i on a Java program J using y_i, runtime is t_s.
2. Alice generates an alleged watermark ω' by s_i, runtime is t_n. In fact, there isn't the watermark ω' in J
3. Alice attacks our scheme P_J using a passive ambiguity, runtime is $t(n)$. $(J, \omega', K) \leftarrow$Alice (J, y_i).
4. If Alice attacks successfully, then verifies: output $e(s_i, P) = \prod_{r=1}^{i} e(H(J, L_r), y_r)$, runtime is $O(n)$, else output FALL, end.
   Thus, there is an attack that $(t_s + t_n + t(n) + O(n), \varepsilon)$-breaks our extended aggregate signatures scheme S. This contradicts the assumption above.

CASE 2. Suppose that Alice can (t', ε')-break our scheme P_J using copy attacks. Then, We will also get a contradiction.

Construct another signature forging algorithm $Forg2_n$:

1. Runs system's parameter $(1)^n$, produce embedding watermark key K, runtime is t_k
2. Generates two alleged Java program, J_1 and J_2, each length is n, runtime is $2t_n$.
3. For $i \in (1, 2, ..., m)$, from the signature inquiry of $< ID_1, ..., ID_i >$ on J_1 gets watermark ω, runtime is t_ω.
4. Encodes the watermark ω $(\omega'_\sigma \leftarrow \omega)$, then embed it into J_1.
   $J'_1 \leftarrow Embed(J_1, \omega'_\sigma, K)$, runtime is $t_{\omega'_\sigma}$.
5. Alice attacks our scheme P_J using a copy: copies ω'_σ from J'_1, then embeds ω'_σ into J_2, $J'_2 \leftarrow Alice(J'_1, J_2, K)$, runtime is $t(n)$.
6. If Alice attacks successfully, then verifies: output $e(s_i, P) = \prod_{r=1}^{i} e(H(J, L_r), y_r)$, runtime is $O(n)$, else output FALL, end.
   Thus, there is another attack that $(t_k + 2t_n + t_\omega + t_{\omega'_\sigma} + t(n) + O(n), \varepsilon')$-breaks our extended aggregate signatures scheme S. This also contradicts the assumption above.

Hence, our watermarking scheme P_J is secure against passive ambiguity and copy attacks.

Theorem 2. *Our watermarked opaque predicates* $\{\mathbf{O}_j\}_{j=1}^{j=n}$ *are trapdoor watermarked opaque predicates.*

Proof. From paper [9, 10], most of the number-theoretical opaque predicates are complex, and complex number-theoretical opaque predicates are one-way. Without opaque predicates key $(k_j)_{j=1}^{j=n}$, three formulas (Eq.(1), Eq.(2), Eq.(3)) are very difficult to be distinguished from the program J. Intuitively, the complexity of $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ is superpolynomial. It is one-way.

On the other hand, using opaque predicates keys $(k_j)_{j=1}^{j=n}$, it is very easy to distinguish the formulas Eq.(1), Eq.(2), and Eq.(3). It is also easy to compute Legendre symbol of module a prime p.

Hence, our watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ are trapdoor.

Corollary 1. *Our embedding watermark algorithms are robust.*

Proof. It can be proved from theorem 2 above.
So, the "core" of predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ must be contained within the transformations $T_c(J_\omega)$ for any feasible composition of transformations T_c.

Corollary 2. *Our watermarking scheme* P_J *is more invisible.*

Proof. From theorem 2 above, compare with the opaque predicates proposed in paper [10], without opaque predicates keys $(k_j)_{j=1}^{j=n}$, three formulas (Eq.(1), Eq.(2), Eq.(3)) are also more difficult to be distinguished each other, and any of the predicates is easier to be hidden. Intuitively, our watermarking scheme P_J is more invisible.

Theorem 3. *Our watermarking scheme P_J is more secure under statical and dynamic attacks.*

Proof. Because our watermarked opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ are constructed based on the intractability of Legendre symbol static analysis problems, so our opaque predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ are resilient against static attacks.

On the other hand, our predicates $\{\boldsymbol{O}_j\}_{j=1}^{j=n}$ are designed in such a way that several predicates have to be cracked at the same time. Thus they stand against dynamic attacks. It is shown in paper [3, 8].

According to Corollary 2 again, our watermarking scheme P_J is more secure against statical and dynamic attacks.

Theorem 4. *Our watermarking scheme P_J is zero-knowledge proof of the watermark ω.*

Proof. It is obvious from Eq.(10). In Our watermarking scheme P_J, the verifying of the watermark ω uses zero-knowledge proof [6, 12], which is a non-interactive proof involving many provers and one verifier.

Corollary 3. *From above, our watermarking scheme P_J is secure against watermarks protocol attacks. The watermark ω can't be removed or modified by anyone even if any other participants and purchasers. The order of ω also can't be changed. The watermark is verified as correct if and only if the correct order of ω and all honest provers.*

Theorem 5. *Our ID-based watermarking scheme P_J is authenticated.*

Proof. It's evident. Whereas, most of former watermarking schemes hadn't considered authentication.

5 Conclusions

The construction of our watermarked opaque predicates is based on Legendre symbol, and the data structure of Legendre symbol is simple, so the data bloating is lower in our scheme than other schemes previously.

From Theorem 4 and Corollary 3, we can choose the order of the watermark ω according to the priority of the participants, the lowest prior participant signs at first and the highest prior participant signs at last.

Our watermarking scheme is dynamic and secure against all usual types of attacks, it is clear that our scheme might can be applied equally well to other languages programs.

Acknowledgement

The authors wish to thank the referees for their comments and suggestions that

helped to improved the correspondence. This work was supported by the National Natural Science Foundation of China (No.60373059), the National Research Foundation for the Doctoral Program of Higher Education of China (No.20040013007) and the Major Research Plan of the National Natural Science Foundation of China(Grant No. 90604023).

References

1. Davidson. R. L., Myhrvold. N.: Method and System for Generating and Auditing a Signature for a Computer Program. US Patent No. 5,559,884. Assignee: Microsoft Corporation(1996)
2. Monden. A., Iida. H., Matsumoto. K., Inoue. K. and Torii. K.: A Practical Method for Watermarking Java Programs. 24th Computer Software and Applications Conference (2000)
3. Collberg. C., Thomborson. C.: Software Watermarking: Models and Dynamic Embeddings. In Conference Record of POPL' 99: The 26th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages (Jan. 1999)
4. Nagra. J., Thomborson. C.: Threading Software Watermarks. In: 6th Intl Information Hiding Workshop (IH 2004), Lecture Notes in Computer Science, Vol.3200. Springer-Verlag, Berlin Heidelberg New York(2004)
5. Adelsbach. A., Katzenbeisser. S., Veith. H.: Watermarking Schemes Provably Secure Against Copy and Ambiguity Attacks. In: ACM Workshop on Digital Rights Management(DRM'03). ACM Press, Washington (2003) 111-119
6. Boneh. D., Gentry. C., Lynn. B., and Shacham. H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Eli Biham (eds.): EUROCRYPT. Lecture Notes in Computer Science, Vol. 2656. Springer-Verlag, Berlin Heidelberg New York (2003) 416-432
7. Saxena. A. and Soh. B.: Additive Zero-Knowledge and Applications: SPAM Prevention. http://eprint.iacr.org/ (2005)
8. Collberg. C., Thomborson. C., Low. D.: Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs. In Principles of Programming Languages 1998, POPL' 98, San Diego, CA.(1998). http://www.cs.arizona.edu/ collberg/Teaching/620/
9. Collberg. C.: CS 620 Security Through Obscurity. Course Notes from http://www.cs.arizona.edu/ collberg/Teaching/SoftwareSecurity.html(2002)
10. Arboit. G.: A Method for Watermarking Java Programs via Opaque Predicates. In: International Conference on Electronic Commerce Research (*ICECR*-5)(2002)
11. Chengdong Pan, Chengbiao Pan.: Elementary Numbers Theory. Beijing University Publishing Company, Beijing, China (1991)
12. Wenbo Mao.: Modern Cryptography: Theory and Practice. Publishing House of Electronic Industry, Beijing, China (2004)
13. Cox. I. J., Miller. M. L., Bloom. J. A.: Digital Watermarking. Publishing House of Electronic Industry, Beijing, China (2003)