Algorithmic Information Theory for Obfuscation Security

Rabih Mohsen[1] and Alexandre Miranda Pinto[2,3]
[1]*Department of Computing, Imperial College London, London, U.K.*
[2]*Information Security Group, Royal Holloway University of London, Egham, U.K.*
[3] *Instituto Universitário da Maia, Maia, Portugal*

Keywords: Code Obfuscation, Kolmogorov Complexity, Intellectual Property Protection.

Abstract: The main problem in designing effective code obfuscation is to guarantee security. State of the art obfuscation techniques rely on an unproven concept of security, and therefore are not regarded as provably secure. In this paper, we undertake a theoretical investigation of code obfuscation security based on Kolmogorov complexity and algorithmic mutual information. We introduce a new definition of code obfuscation that requires the algorithmic mutual information between a code and its obfuscated version to be minimal, allowing for controlled amount of information to be leaked to an adversary. We argue that our definition avoids the impossibility results of Barak et al. and is more advantageous then obfuscation indistinguishability definition in the sense it is more intuitive, and is algorithmic rather than probabilistic.

1 INTRODUCTION

Malicious reverse engineering represents a great risk to software confidentiality, especially for software that runs on a malicious host controlled by a software pirate intent on stealing intellectual artifacts. Thwarting vicious reverse engineering, using software protection techniques such as code obfuscation, is vital to defend programs against malicious host attacks. An obfuscating transformation attempts to transform code so that it becomes unintelligible to human and automated program analysis tools, while preserving their functionality. It is a low-cost technique that does not affect portability and has promise for defending mobile programs against malicious host attacks.

One of the major challenges of code obfuscation is the lack of a rigorous theoretical background. The absence of a theoretical basis makes it difficult to formally analyse and certify the effectiveness of these techniques against malicious host attacks. In particular, it is hard to compare different obfuscating transformations with respect to their resilience to attacks. Often, obfuscation transformation techniques are proposed with no provable properties presented.

Software developers strive to produce structured and easy to comprehend code, their motivation is to simplify maintenance. Code obfuscation, on the other hand, transforms code to a less structured and intelligible version. It produces more complex code that looks patternless, with little regularity and is difficult to understand. We argue that irregularities and noise makes the obfuscated code difficult to comprehend. Kolmogorov complexity (Li and Vitányi, 2008) is a well known theory that can measure regularities and randomness. The size of the shortest program that describes a program tends to be bigger as more noise and irregularities are present in a program. Kolmogorov complexity is the basic concept of algorithmic information theory, that in many respects adapts the viewpoint of well-established Information Theory to focus on individual instances rather than random distributions. In general, algorithmic information theory replaces the notion of probability by that of intrinsic randomness of a string.

Kolmogorov complexity is uncomputable; however it can be approximated in practice by lossless compression (Kieffer and Yang, 1996) (Li and Vitányi, 2008), which helps to intuitively understand this notion and makes this theory relevant for real world applications. Our aim in this paper is to provide a theoretical framework for code obfuscation in the context of algorithmic information theory: to quantitatively capture the security of code obfuscation, to discuss its achievability and to investigate its limitations and resilience against an adversary.

We introduce the notion of unintelligibility to define confusion in code obfuscation and argue this is not good enough. We then propose our notion of se-

Mohsen R. and Miranda Pinto A..
Algorithmic Information Theory for Obfuscation Security.
DOI: 10.5220/0005548200760087
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 76-87
ISBN: 978-989-758-117-5

curity and compare both definitions. We argue that our model of security is fundamentally different from the virtual black-box model of Barak et al., and that because of this their impossibility result does not apply. Then, we show that under reasonable conditions we can have secure obfuscation. Finally, we study the security of two main approaches to obfuscated code in software, *encoding* and *hiding*, at the subprogram level.

To the best of our knowledge, this paper is the first to propose algorithmic information theory as a theoretical basis for code obfuscation and its threat model; we believe our work is the first step to derive consistent metrics that measure the protection quality of code obfuscation. The framework can be applied to most code obfuscation techniques and is not limited to any obfuscation method or language paradigm.

Paper structure: In Section 2, we provide an overview of related work. Section 3 provides the preliminaries and the required notations. In Sections 4, we discuss the intuition behind our approach, propose the formal definition of code obfuscation and present some results for security code obfuscations against passive attackers. In section 5, we study the security of two main approaches to code obfuscation at the subprogram level. Section 6 concludes with the proposed future work.

2 RELATED WORK

Collberg et al. (Collberg et al., 1997) were the first to define obfuscation in terms of a semantic-preserving transformation. Barak et al. (Barak et al., 2001) introduced a formal definition of *perfect* obfuscation in an attempt to achieve well-defined security, which is based on the black-box paradigm. Intuitively, a program obfuscator $\mathcal{O}$ is called *perfect* if it transforms any program P into a virtual black box $\mathcal{O}(P)$ so that anything that can be also efficiently computed from $\mathcal{O}(P)$, can be efficiently computed given just oracle access to P. However, they also proved that the black-box definition cannot be met by showing the existence of set of functions that are impossible to obfuscate. On the other hand, a recent study conducted by Garg et al (Garg et al., 2013) provided positive results, using indistinguishability obfuscation, for which there are no known impossibility results. However, as argued by (Goldwasser and Rothblum, 2007) there is a disadvantage of indistinguishability obfuscation: it does not give an intuitive guarantee about the security of code obfuscation.

3 PRELIMINARIES

We use the U as the shorthand for a universal Turing machine, x for a finite-length binary string and $|x|$ for its length. We use ϵ for a negligible value, and Λ for an empty string. We use the notation $O(1)$ for a constant, $p(n)$ for a polynomial function with input $n \in \mathbb{N}$. $\|$ is used to denote the concatenation between two programs or strings.

$\mathcal{P}$ is a set of binary programs and $\mathcal{P}'$ is a set of binary obfuscated programs, $\mathcal{L} = \{\lambda_n : \lambda_n \in \{0,1\}^+, n \in \mathbb{N}\}$ is a set of (secret) security parameters used in the obfuscation process [1]. $\mathbb{A} = \{\mathcal{A}_n\}_{n\in\mathbb{N}}$ represents a set of adversaries (deobfuscators) where an adversary $\mathcal{A} \subseteq \mathbb{A}$ uses a set of deobfuscation techniques (e.g. reverse engineering); the term adversary is used interchangeably with deobfuscator.

We consider only the prefix version of Kolmogorov complexity (prefix algorithmic complexity) which is denoted by $K(.)$. *Complexity* and *Kolmogorov complexity* terms are sometimes used interchangeably; for more details on prefix algorithmic complexity and algorithmic information theory, we refer the reader to (Li and Vitányi, 2008). The necessary parts of this theory are briefly presented in the following.

Definition 1. Let $U(P)$ denote the output of U when presented with a program $P \in \{0,1\}^+$.

1. The *Kolmogorov complexity* $K(x)$ of a binary string x with respect to U is defined as:
$$K(x) = \min\{|P| : U(P) = x\}.$$
2. The Conditional Kolmogorov Complexity relative to y is defined as:
$$K(x\,|\,y) = \min\{|P| : U(P,y) = x\}.$$

Definition 2. Mutual algorithmic information of two binary programs x and y is given by
$$I_K(x;y) = K(y) - K(y\,|\,x).$$

Theorem 1 (chain rule (Gács, 1974))**.** *For all x, y* $\in \mathbb{N}$

1. $K(x;y) = K(x)+K(y\,|\,x)$ *up to an additive term* $O(\log K(x,y))$.
2. $K(x)-K(x\,|\,y) = K(y)-K(y\,|\,x)$ *i.e.* $I_K(x;y) = I_K(y;x)$, *up to an additive term* $O(\log K(x,y))$.

Logarithmic factors like the ones needed in the previous theorem are pervasive in the theory of Kolmogorov complexity. As commonly is done in the literature, we mostly omit them in our results, making a note in the theorem statements that they are there. This also "hides" smaller constant terms, and for this reason we regularly omit them in the derivations.

[1] The security parameter may include the obfuscation key, the obfuscation transformation algorithm or any necessary information that the obfuscation function can use.

Definition 3. The Mutual algorithmic information of two binary strings x and y conditioned to a binary string z is defined as: $I(x;y|z) = K(y|z) - K(y|x,z)$

Theorem 2 ((Li and Vitányi, 2008))**.** *There is a constant c such that for all x and y*

$$K(x) \leq |x| + 2\log|x| + c \quad \textit{and} \quad K(x\,|\,y) \leq K(x) + c.$$

Theorem 3 ((Shen, 1982; Taveneaux, 2011))**.** *Given a recursive computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$, for all x the algorithmic information content of $f(x)$ is bounded by: $K(f(x)) \leq K(x) + O(1)$.*

Theorem 4 ((Shen et al., 2014))**.** *Given a recursive computable function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$, for any strings x, y the algorithmic information content of $f(x,y)$ is bounded by:*

$$K(f(x,y)|y) \leq K(x|y) + O(1).$$

If we are aware that x belongs to a subset S of binary strings, then we can consider its complexity $K(x|S)$. Also, we can measure the level of randomness (irregularities) using randomness deficiency.

Definition 4. (Li and Vitányi, 2008) The randomness deficiency of x with respect to a finite set S containing x is defined as $\delta(x|S) = \log \#S - K(x|S)$.

Lemma 5. *(Li and Vitányi, 2008) Let S be an enumerable binary set of programs such that $x \in S$, Then,*

$$K(x) \leq K(S) + \log \#S + O(1)$$

4 CODE OBFUSCATION USING KOLMOGOROV COMPLEXITY

4.1 Motivation

The main purpose of code obfuscation is to confuse an adversary, making the task of reverse engineering extremely difficult. Code obfuscation introduces noise and dummy instructions that produce irregularities in the targeted obfuscated code. We believe that these make the obfuscated code difficult to comprehend. Classical complexity metrics have a limited power for measuring and quantifying irregularities in obfuscated code, because most of these metrics are designed to measure certain aspects of code attributes such as finding bugs and code maintenance. Human comprehension is a key in this case; an adversary has to understand the obfuscated code in order to recover the original. Measuring code obfuscation has to take into consideration this human factor. Although measuring code comprehension is very subjective, there were some successful attempts to measure human cognitive reasoning and cognitive science based on Kolmogorov complexity (Gauvrit et al., 2011).

Code regularity (and irregularity) can be quantified, as was suggested in (Lathrop, 1997) and (Jbara and Feitelson, 2014), using Kolmogorov complexity and compression. Code regularity means a certain structure is repeated many times, and thus can be recognized. Conversely, irregularities in code can be explained as the code exhibiting different types of structure over the code's body. Regularities in programs were introduced by Jbara et al. in (Jbara and Feitelson, 2014) as a potential measure for code comprehension; they experimentally showed using compression that long regular functions are less complex than the conventional classical metrics such as LOC (Line of Code) and McCabe (Cyclomatic complexity) could estimate.

The main intuition behind our approach is based on the following argument: if an adversary fails to capture some patterns (regularities) in an obfuscated code, then the adversary will have difficulty comprehending that code: it cannot provide a valid and brief, i.e., simple description. On the other hand, if these regularities are simple to explain, then describing them becomes easier, and consequently the code will not be difficult to understand.

We demonstrate our motivation using the example in Fig. 1. We obfuscate the program in Fig. 1-(a) that calculates the sum of the first n positive integers, by adding opaque predicates[2] with bogus code and data encoding. If we apply Cyclomatic complexity (McCabe (McCabe, 1976)), a classical complexity measure, to Fig. 1-(b) the result will be 6. Cyclomatic complexity is based on control flow graph (CFG), and is computed by: $E - N + 2$, where E is the number of edges and N is the number of nodes in CFG. Fig. 1-(b) contains $N = 8$ nodes, $E = 13$ edges then the Cyclomatic complexity is $(13 - 8 + 2) = 7$. We can see some regularity here: there is one opaque predicate repeated three times. Furthermore, the variable y is repeated three times in the same place of the `If`-branch. We conjecture that we can find the short description of the program in Fig. 1-(b), due to presence of regularity by using lossless compression.

We take another obfuscated version in Fig. 1-(c) (of the same program); this code is obfuscated by adding three different opaque predicates. The patterns are less in this version comparing to Fig. 1-(b); the shortest program that describes Fig. 1-(c) is likely to be very similar to the code itself, where the Cyclo-

[2] An opaque predicate is an algebraic expression which always evaluates to same value (true or false) regardless of the input.

```
while(i<n){
 i=i+1;
 x=x+i;}
```

(a) Sum code

```
while(i<n){
  i=i+1;
if (7*y*y-1==x*x){ //false
  y=x*i;
else
  x=x+4*i;}
if (7*y*y-1==x*x){
  y=x*i;
else
  x=x-2*i;}
if (7*y*y-1==x*x){
  y=x*i;
else
  x=x-i;}}
```

(b) One opaque predicate

```
while(i<n){
 i=i+1;
 if (7*y*y-1==x*x){ //false
  y=x*(i+1);
 else
  x=x+4*i;}
 if (x*x-34*y*y==-1){ //false
  y=x*i;
 else
  x=x-2*i;}
 if ((x*x+x)mod 2==0){ //true
  x=x-i;
 else
  y=x*(i-1);}}
```

(c) Three opaque predicate

Figure 1: Obfuscation example: (a) is the original code for the sum of n integers; (b) is an obfuscated version of (a) with one opaque predicate and data encoding which has some patterns and regularities; (c) is another obfuscated version of (a) with three opaque predicate and data encoding, which has less patterns and regularities comparing to (b).

matic complexity is the same 7, and it does not account for the changes that occurred in the code. Assuming the opaque predicates of Fig. 1-(c) are equally difficult to break, attacking this code requires at least twice more effort than the code in Fig. 1-(b), as we need to figure out the value of two more opaque predicates. Furthermore, Fig. 1-(b) can be compressed at higher rate than Fig. 1-(c); again, this is due to the inherent regularity in Fig. 1-(b).

We argue that an obfuscated program which is secure and confuses an adversary will exhibit a high level of irregularity in its source code and thus require a longer description to characterize all its features. This can be captured by the notion of Kolmogorov Complexity, which quantifies the amount of information in an object. An obfuscated program will have more non-essential information, and thus higher complexity, than a non-obfuscated one. Thus, we can use Kolmogorov Complexity to quantify the level of confusion in obfuscated programs due to the obfuscation process.

4.2 Applying Kolmogorov Complexity to Code Obfuscation

In this section, we present a novel approach for code obfuscation based on notions from algorithmic information theory. We start with an intuitive definition that is inspired by practical uses of obfuscation. The rationale behind this definition is that an obfuscated program must be more difficult to understand than the original program. This uses the separate notion of c-unintelligibility:

Definition 5. A program P' is said to be c-unintelligible with respect to another program P if it is c times more complex than P, i.e. the added complexity is c times the original one, and thus more difficult to understand. Formally:

$$K(P') \geq (c+1)K(P),$$

for some constant $c > 0$.

Definition 6. A *c-Obfuscator* $\mathcal{O} : \mathcal{P} \times \mathcal{L} \rightarrow \mathcal{P}'$ is a mapping from programs with security parameters $\mathcal{L}$ to their obfuscated versions such that $\forall P \in \mathcal{P}, \lambda \in \mathcal{L}$. $\mathcal{O}(P,\lambda) \neq P$ and satisfies the following properties:

- **Functionality**: $O(P,\lambda)$ and P compute the same function.
- **Polynomial Slowdown**: the size and running time of $\mathcal{O}(P,\lambda)$ are at most polynomially larger than the size and running time of P, i.e. for polynomial function p. $|\mathcal{O}(P,\lambda)| \leq p(|P|)$, and if P halts in k steps on an input i, then $O(P,\lambda)$ halts within $p(k)$ steps on i.
- **Unintelligibility**: $\mathcal{O}(P,\lambda)$ is c-unintelligible with respect to P.

It is interesting to ask to what extent unintelligibility is related to the quality of the obfuscation parameter λ. Is a large λ *necessary* for high unintelligibility? Is it *sufficient*?

We answer the first question in the positive by showing that c-unintelligibility sets a lower bound on the size of λ.

Lemma 6. *Consider a program P and an obfuscated version $P' = \mathcal{O}(P,\lambda)$ such that P' is c-unintelligible with respect to P. Then, $|\lambda| \geq cK(P) - O(1)$.*

Proof. By assumption, $K(\mathcal{O}(P,\lambda)) \geq (c+1)K(P)$. To compute P', we only need P, λ and the obfuscator program $\mathcal{O}$ and so we can upper bound $K(P')$:

$$\begin{aligned} K(P) + K(\lambda|P) + K(\mathcal{O}) &\geq K(\mathcal{O}(P,\lambda)) \\ &\geq (c+1)K(P) \\ \Rightarrow K(\lambda|P) &\geq cK(P) - K(\mathcal{O}). \end{aligned}$$

Assuming the obfuscator program is simple, that is, $K(\mathcal{O}) = O(1)$, we have by basic properties of Kolmogorov complexity: $|\lambda| \geq K(\lambda) \geq K(\lambda|P) \geq cK(P) - O(1)$.

□

To answer the second question, we show a counter-example. So far, we have not addressed the nature of $\mathcal{O}$ and how well it uses its obfuscation parameter. It could well be the case that $\mathcal{O}$ only uses some bits of λ to modify P. In an extreme case, it can ignore λ altogether and simply return $\mathcal{O}(P,\lambda) = P$. The result satisfies the first two properties of an obfuscator, but can be considered unintelligible only in the degenerate case for $c = 0$ and surely we would not call the resulting code obfuscated. Another extreme case is when $\lambda = P$. Now, we would have at most $K(\mathcal{O}(P,\lambda)) \leq K(P) + K(\mathcal{O}) + O(1)$ which again would lead to a very small c. These two cases, although extreme, serve only to show that the quality of an obfuscator depends not only on λ but also on the obfuscation algorithm itself, $\mathcal{O}$. This is addressed later in Theorem 12.

Definition 6 is perhaps the first natural definition one can find, but it has one shortcoming. Merely requiring the obfuscated program to be complex overall does not mean that it is complex in all its parts, and in particular, that it hides the original program. To illustrate this point, consider the following example.

Example 1. *Consider an obfuscated program $P' = \mathcal{O}(P,\lambda) = P \parallel \lambda$, which is a simple concatenation of P and λ. Define $n = |P'|$. We know $K(P \parallel \lambda) \simeq K(P,\lambda)$ within logarithmic precision (see (Li and Vitányi, 2008) page 663). Then, by applying the chain rule of Theorem 1, $K(P') = K(P \parallel \lambda) \simeq K(P) + K(\lambda|P) + O(\log n)$. For large λ independent of P, this might signify a large unintelligibility, but the original program can be extracted directly from the obfuscated version requiring only $O(\log n)$ to indicate where P ends and λ starts.*

This leads us to our second definition, where we require not that the obfuscated program be more complex than the original but rather, that it reveal almost no information about the original. This is captured by the notion of algorithmic mutual information and can be stated formally as:

Definition 7. Consider a program P and its obfuscated program $P' = \mathcal{O}(P,\lambda)$. We say P' is a γ-secure obfuscation of P if the mutual information between P and P' is at most γ, that is:

$$I_K(P;\mathcal{O}(P,\lambda)) \leq \gamma.$$

We say P' is a secure obfuscation of P if is γ-secure and γ is negligible.

It is common to consider, in the literature about Kolmogorov Complexity, that logarithmic terms are negligible. Thus, if both P and P' have lengths of order n, we migth consider that P' would be a secure obfuscation for $\gamma = \log n$. This intuition, however, is bound to fail in practice.

Programs are typically redundant, written in very well-defined and formal languages, with common structures, design patterns, and even many helpful comments. It is expected that the complexity of a non-obfuscated program be low, compared to its length. Consider then the case that for a given P and $n = |P|$ we have $K(P) = O(\log n)$. Consider a scenario like that of Example 1, where the obfuscated reveals the original program and so the mutual information between both programs is maximum, i.e.,

$$I_K(P;\mathcal{O}(P,\lambda)) = K(P) - O(\log n) = O(\log n).$$

Even though this obfuscation can not be considered secure, the resulting mutual information is so small that Definition 7 would declare it secure. We have two ways out of this:

- we do not consider programs with $K(P) = O(\log n)$, since this is the error margin of the important properties of Kolmogorov complexity, and at this level we can not achieve significant results;
- or we consider a relative definition of security, requiring that the mutual information be only at most a negligible fraction of the information in P.

The second option leads us to the following definition:

Definition 8. Consider a program P and its obfuscated program $P' = \mathcal{O}(P,\lambda)$. We say P' is a ϵ-secure obfuscation of P if the mutual information between P and P' is at most $\epsilon K(P)$, that is:

$$I_K(P;\mathcal{O}(P,\lambda)) \leq \epsilon K(P),$$

for $0 \leq \epsilon \leq 1$.

We say P' is a secure obfuscation of P if is ϵ-secure and ϵ is negligible in some appropriate sense.

The next proposition provides an example of how to test the security of an obfuscated code against an adversary.

Proposition 1. *Consider a clear program P of length n with $K(P) \geq n - O(\log n)$ and A an adversary that extracts at least $m \leq n$* consecutive *bits of P from $P' = \mathcal{O}(P,\lambda)$ then:*

1. $K(P|P') \leq n - m + O(\log n)$.
2. $I_K(P;P') \geq m - O(\log n)$

Proof. We prove this proposition by building the following algorithm:

Algorithm:

- Build A of length $O(1)$.
- Compute $A(P')$, we obtain $m \leq n$ bits of P. Denote these by string ω.
- Now, build a program β such that $P = \beta(\omega)$ which computes two blocks of bits: those that come before and those that come after ω.
  To produce P, β needs at most to produce a pair of two strings (s_1, s_2) with combined length $n - m$. To describe the pair, we need at most $O(\log n)$ bits saying where to divide s_1 from s_2. Thus, $K(\beta) \leq n - m + O(\log n)$.

By construction, $K(P|\mathcal{O}(P,\lambda)) + O(1) \leq K(\beta) + O(1) \leq n - m + O(\log n)$. Using the assumption about $K(P)$, it is straight forward to compute

$$I_K(P;P') \geq m - O(\log n)$$

□

On the other hand, the adversary can fully obtain P, with only a logarithmic error, if it knows λ the security parameter, as the next theorem shows.

Theorem 7. *Let $P' = \mathcal{O}(P,\lambda)$, for an adversary A who knows λ:*

1. $K(P|P',\lambda) = O(\log n)$.
2. $I_K(P,P'|\lambda) = K(P|\lambda) - O(\log n)$.

where n is the maximum length of P, P' and λ.

Proof. If λ contains all the knowledge that A requires to obtain P given P', then the shortest program for a universal Turing machine U that describes P given A, P' and λ, is negligible i.e. empty string. It is sufficient for U to describe P from A, P' and λ with no need of any extra programs i.e. $U(\Lambda, A(P'), \lambda) = P$ where Λ is an empty string. $O(\log n)$ is needed as an overhead cost required by U to combine A, P' and λ and to locate them on U tape. The advantage of A given λ is obtained as follows:

$$\begin{aligned} I_K(P;P'|\lambda) &= K(P|\lambda) - K(P|P',\lambda) \\ &= K(P|\lambda) - O(\log n) \end{aligned}$$

□

These results are not surprising. Intuitively, an adversary can easily recover the original code from the obfuscated version once the security parameter that is used for obfuscation is known.

4.3 On the Impossibility of Obfuscation

There exist other definitions of obfuscation in the literature. Of particular importance to us is the work of (Barak et al., 2012), due to its famous impossibility result. As the authors argue in that paper, the black-box model they propose for obfuscation is too strong to be satisfiable in practice.

The black-box model considers a program[3] P obfuscated if any property that can be learned from P can also be obtained by a simulator with only oracle access to P. This essentially states that P does not give any particular information about that property, since it is possible to learn it without having access to P. Notice that this model does not compare an obfuscated program with an original one, but rather with its functionality.

This is different from the definitions that we have proposed so far. Our definitions can be used to capture this purpose, namely, measuring how much information a program P gives about the function it computes, which we denote by $[\![P]\!]$.

It suffices to note that every function F has a minimal program for it, say, Q. Then, its Kolmogorov complexity is the size of Q and for every other program P computing F we have

$$K(F) = |Q| \leq |P|.$$

For every such program it must be that $K(F) \leq K(P) + O(1)$, otherwise we could build a program R of size $K(P)$ that produced P and then ran in succession R and $U(R)$.[4] This composition of programs is itself a program with complexity $K(P) + O(1)$ which would be smaller than the assumed minimal program F, i.e., a contradiction. Therefore, our definition can be changed to compare the obfuscated program not with any simpler program but with *the simplest* program computing the same function.

Definition 9. Consider a program P'. We say P' is ϵ-securely obfuscated if

$$I_K([\![P']\!];P') \leq \epsilon K([\![P']\!]),$$

for $0 \leq \epsilon \leq 1$.

We say P' is a secure obfuscated program if is ϵ-secure and ϵ is negligible.

We believe our definitions of obfuscation differ from the simulation black-box model in important ways, and that because of this they avoid the impossibility result of (Barak et al., 2012).

[3] Or rather, a circuit or a Turing machine representation thereof.

[4] That is, we run R to produce P then we execute the result of this first execution, that is P itself.

Our definition is a less stringent form of obfuscation rather than a weak form of black box obfuscation. We assume the functionality of an obfuscated program is almost completely known and available to an adversary, and only require hiding the implementation rather than the functionality itself. This approach to obfuscation is very practical and pragmatic, especially for software protection obfuscation, as usually the customer likes to know exactly what a product does, although s/he might not care about how it was implemented.

Our definition for security takes a program P (clear code), which supposedly is an easy and smart implementation of functionality F, and compares it with P', which is a different and supposedly unintelligible implementation of F, such that the original P can not be perceived or obtained from P'. The defenders' aim is not to prevent an adversary from understanding or finding F, but to prevent her/him from finding their own implementations P.

This intuition best matches the idea of *best possible obfuscation* which was advanced by Goldwasser and Rothblum (Goldwasser and Rothblum, 2007). According to (Goldwasser and Rothblum, 2007) an obfuscator $\mathcal{O}$ is considered as best possible if it transforms a program P so that any property that can be computed from the obfuscated program P', can also be computed from any equivalent program of the same functionality. However, despite the close intuitive correspondence, our definition also differs from best possible obfuscation, in the sense that it relies on some form of black-box simulation. It was proved in (Goldwasser and Rothblum, 2007) that best possible obfuscation has a strong relation with *indistinguishability obfuscation* (Barak et al., 2001) (Garg et al., 2013), if $\mathcal{O}$ is an efficient obfuscation i.e. run in polynomial time.

In contrast, the black box model definition requires that all properties of a given obfuscated program P must be hidden from any adversary that ignores its source code but has access to at most a polynomial number of points of $[\![P]\!]$.

1. We can see that in our case, the adversary knows more about the functionality. Since the functionality is mostly public, this would be equivalent to giving the simulator in the black-box model access to this extra knowledge, reducing the advantage of the adversary and possibly making some functions obfuscatable.
2. On the other hand, our definition allows the leakage of a small, but non-zero, amount of information. Compare with the black-box model where a single-bit property that is non-trivially computed by an adversary, renders a function unobfuscatable. Our definition requires the adversary to be able to compute more than a few bits in order for obfuscation to be broken.
3. Our definition considers only deterministic adversaries, again making adversaries less powerful and reducing their advantage.

We can try to model the implications for our definition of a successful black-box adversary against obfuscation. Consider an adversary $\mathcal{A}$ attacking a predicate π by accessing an algorithm A , such that $A(P') = 1$ if and only P' satisfies π. In this case, $\mathcal{A}$ is able to compute 1 bit of information about P', and we want to measure how much this helps $\mathcal{A}$ in describing some simpler program P that implements the same function $[\![P']\!]$.

Since the adversary $\mathcal{A}$ knows A, s/he can enumerate the set S of all programs that satisfy A. Then, for some program $P \in S$, we would have $K(P|A) \leq K(S|A) + \log|S|$.

Note that the set S may be infinite, and so enumeration is the best that can be done: we enumerate all relevant programs and run A with each of them, noting the result. We could try to avoid this infinity problem by noticing we are only interested in programs simpler than the original, and thus satisfying $K(P) \leq K(P') \leq |P'|$. This does not give absolute guarantees, since in general there are programs with $|P| > |P'|$ and $K(P) \leq K(P')$, but our hope is that these are few and far between as they increase in length. Thus, even if we make this assumption and disregard a whole class of possibly relevant programs, we still have in the order of 2^n specimens. If A were a deterministic algorithm, we would have $K(S|A) = O(1)$, and if S has less than a half of all possible programs, then indeed we would find that $K(P|A) \leq K(S|A) + \log|S|/2 \leq n - 1$.

However, A is randomized, and in order to accurately produce S, for each program q, A must be run with a set of random coins r such that $A(q,r) \Leftrightarrow q \in S$. One way to describe S from A would need a polynomial number of bits for each program in S. Now, we no longer have the comfort of a negligible $K(S|A)$ term, and we can no longer be sure that knowing this property would in any way reduce the complexity of our target program.

We can still try to go around this problem, by allowing our enumerator to list not only all programs but also all possible random strings, and choosing the majority vote for each program. The length, and therefore the number, of possible random strings is bound by the running time of the program, which in turn is bound by a function of its length. Therefore, if we limit the length of our programs we limit the number

of random strings to search[5].

This would eliminate the necessity of considering the extra information due to the random coins, but on the other hand the running time would increase exponentially. Any successful adversary would be very different to the PPT adversaries of (Barak et al., 2001), and although our definition has been made, for ease of exposition, with unbounded Kolmogorov complexity (for unbounded adversaries), it is easy to change it to consider polynomially-bounded adversaries by using an alternative definition of mutual information:

$$I^*_K(P;P') = K(P) - K^{t()}(P|P'),$$

where $t()$ is a polynomial on the length of P' that acts as a hard limit to the number of steps the universal machine can be run for.

This notion of information can be negative in some cases, but clearly limits the ability of any adversary trying to find P from P' in a consistent way with the black-box model. With this definition of obfuscation, the above reasoning would lead to examples where non-obfuscatability by the black-box model does not prevent obfuscatability in the algorithmic information-theoretic model.

4.4 Security and Unintelligibility

Our first attempt to characterize obfuscation was based on *unintelligibility*, and then we evolved to a notion of security based on *mutual information*. The first notion seemed more immediately intuitive, traditional obfuscation techniques seem to rely only on making the code as complex as possible in an attempt to hide its meaning. But precisely this notion of "hiding meaning" is nothing more than reducing the information that the obfuscated program leaks about the original one, and so we believe the second approach to be the correct one. However, we can ask the natural question: is there any relation between these two concepts? Does high unintelligibility imply high security, or vice-versa?

We give a partial answer to this question. In certain situations, high unintelligibility will imply high security, as stated in the following theorem.

Theorem 8. *Consider a program P of length m and its obfuscated version $P' = \mathcal{O}(P,\lambda)$ of length $n > m$ (where n is at most polynomially larger than m), satisfying c-unintelligibility for $c \geq \frac{n}{K(P)}$. Assuming that the obfuscation security parameter λ satisfies $K(P'|P) \geq K(\lambda|P) - \alpha$, then up to $O(\log n)$:*

$$I(P;\mathcal{O}(P,\lambda)) \leq \alpha - O(1)$$

[5]Conversely, if we really want to enumerate all programs, then we also have to enumerate an infinite number of random strings.

```
\\ variable that holds authentication
    password
string user-Input = input();
string secure-password = ...;
if secure-password ==  user-Input {
 grant-access();
else
 deny-access();}
```

P : simple password checker

```
string O@ = x0();
string $F=...;
if O@== $F{
 x1();
else
 x2(); }
```

P' : obfuscating P

Figure 2: An example for obfuscating a program using renaming technique.

Proof. By Theorem 3, $K(P') = K(\mathcal{O}(P,\lambda)) \leq K(P,\lambda) + K(\mathcal{O}) = K(P,\lambda)$[6] and by assumption $K(P') \geq (c+1)K(P)$. Then,

$$K(P,\lambda) \geq (c+1)K(P)$$
$$K(P,\lambda) - K(P) \geq cK(P)$$
$$K(\lambda|P) \geq cK(P)$$

Now, for mutual information, we have

$$\begin{aligned} I(P;P') &= K(P') - K(P'|P) \\ &\leq K(P') - K(\lambda|P) + \alpha \text{ (by assumption)} \\ &\leq n - cK(P) + \alpha \\ &\leq n - n + \alpha \text{ (by assumption)} \\ &= \alpha \end{aligned}$$

□

Intuitively, if we consider an optimal obfuscation key (it has all the information needed to produce P' from P, but not much more than that), we can say that if P' is c-unintelligible for large enough c, then P' is a secure obfuscation of P.

The above theorem shows when high c-unintelligibility implies security of code obfuscation. It turns out that the reverse implication does not exist, as the following theorem illustrates.

Claim 9. *There are obfuscated functions $\mathcal{O}(P,\lambda)$ that are arbtrarily secure and yet do not satisfy c-unintelligibility for $c \geq 0$.*

[6]The $O(1)$ term is absorbed by the logarithmic additive term that we are not notating

Proof. Consider first the case of program P of Fig. 2, that simply checks a password for access, and its obfuscated version P', which was computed using layout obfuscation (Collberg et al., 1997): variable and function renaming and comment deleting. The obfuscated variable and function names are independent of the original ones and so the information that P' contains about P is limited to the unchanging structure of the code: assignment, test and if branch.

The complexity of the original code can be broken in several independent parts: the comment, the structure, the variable and function names, and therefore we can write $K(P) = n_c + n_s + n_v$. The only thing that P' can give information about is the structure part, since all the other information was irrevocably destroyed in the process: there is no data remaining that bears any relation to the lost comment or the lost function names. Therefore, $I_K(P;P') \leq n_s = \frac{n_s}{n_c+n_s+n_v} K(P)$. We can make the fraction $\frac{n_s}{n_c+n_s+n_v}$ as small as necessary by inserting large comments, long names and representing structure as compactly as possible, for example, keeping names in a dictionary block and indicating their use by pointers to this.

However the complexity of P' is less than that of P, since there is less essential information to describe: the same structure, no comments and the function names could be described by a simple algorithm. Therefore, we have that c-unintelligibility can not be satisfied for any non-negative value of c. This shows that high ϵ-security does not imply high unintelligibility. □

The next theorem shows how we can obtain security if the obfuscated code is complex enough and the obfuscation key is independent of the original program.

Theorem 10. *Let P be a program of length n and λ an independent and random obfuscation key, satisfying $K(\lambda) \geq n - \alpha$ and $K(P,\lambda) \geq K(P) + K(\lambda) - \beta$, where $\alpha, \beta \in \mathbb{N}$. Suppose the obfuscation $\mathcal{O}(P,\lambda)$ satisfies $K(O(P,\lambda)|P) \geq K(\lambda|P) - O(1)$. Then up to a logarithmic factor:*

$$I_K(P;\mathcal{O}(P,\lambda)) \leq \alpha + \beta$$

Proof.

$$\begin{aligned} I_K(P;\mathcal{O}(P,\lambda)) &= K(\mathcal{O}(P,\lambda)) - K(\mathcal{O}(P,\lambda)|P) \\ &\leq K(\mathcal{O}(P,\lambda)) - K(\lambda|P) \end{aligned}$$

Applying Theorem 1

$$\begin{aligned} &\leq K(\mathcal{O}(P,\lambda)) - K(\lambda,P) + K(P) \\ &\leq n - K(P) - n + \alpha + \beta + K(P) \\ &\leq \alpha + \beta \end{aligned}$$

□

The first two assumptions are natural: picking λ at random and independently of P will satisfy high complexity and low mutual information with high probability, so we can simply assume those properties at the start. The third assumption, however, is not immediately obvious: it describes a situation where the obfuscation key is *optimal*, in the sense that it contains just the amount of information to go from P to P', within a small logarithmic term. The following lemma shows how λ must have a minimum complexity to allow the derivation of P to P'.

Lemma 11. *Consider $P' = \mathcal{O}(P,\lambda)$ is the result of obfuscating a program P. Then, $K(\lambda|P) \geq K(P'|P) - O(1)$*

Proof. Given the obfuscator $\mathcal{O}$ and any λ, construct the function $q_\lambda(\cdot) = \mathcal{O}(\cdot,\lambda)$. Let Q_λ be a program that implements it. Then, clearly, $U(Q_\lambda,P) = P'$ and so $|Q_\lambda| \geq K(P'|P)$. To describe Q_λ, we only need to specify λ and instructions to invoke $\mathcal{O}$ with the proper arguments, but since P is already known, we can use it to find a shorter description for λ. This gives $K(P'|P) \leq K(\lambda|P) + O(1)$. □

An optimal obfuscation key λ is then the one that uses as little information as possible.

Obfuscation techniques can use randomness or not. In Claim 9, we showed one case where names could be obfuscated in a deterministic way, without any randomness. However, we could equally have used instead highly random names, with the same effect in security but increasing unintelligibility as well. We can as well consider that the set of obfuscation techniques is finite and describe each of them by a unique number. This way, we can characterize a single application of obfuscation by a key composed of the technique's index and the randomness needed.

We now proceed to show that it is possible to achieve obfuscation security according to our definitions, but restricted to a passive adversary, that is, one that does not realize transformations over the intercepted code. The intuition is to use obfuscation techniques that behave as much as possible as secure encryption functions, namely, using random keys that are independent from the code and large enough that they obscure almost all original information. The crucial difference that enables security is that because an obfuscation technique preserves functionality, we do not need to decrypt the obfuscated code and so don't need to hide the key.

First, we prove the effect of obfuscating an elementary piece of code, by application of a single obfuscation technique. Then, we reason about the case of a full program, composed of several of these independent blocks.

Theorem 12. *Let p represent a program block, $\mathcal{O}$ an obfuscation technique and $\lambda \in \mathcal{L}$ an obfuscation key with fixed length n. Let $p' = \mathcal{O}(p, \lambda)$ be the obfuscated block. Assume $\mathcal{O}$ produces an output with length $\ell \leq n + \gamma$ and is "nearly-injective" in the following sense: for every p, any subset of $\mathcal{L}$ of keys with the same behaviour for p has cardinality at most polynomial in n. That is, for all p and $\lambda_0 \in \mathcal{L}$, $|\{\lambda : \mathcal{O}(p, \lambda) = \mathcal{O}(p, \lambda_0)\}| \leq n^k$, for some positive integer k.*

Then, if the key is random, $K(\lambda) \geq n - \alpha$ and independent from p, $K(\lambda|p) \geq K(\lambda) - \beta$, the obfuscated code p' is $(\alpha + \beta + \gamma)$-secure up to a logarithmic term.

Proof. By symmetry of information, we can write $K(p|p') = K(p'|p) + K(p) - K(p')$. Since $p' = \mathcal{O}(p, \lambda)$, it's easy to see that $K(p'|p) \leq K(\mathcal{O}) + K(\lambda|p) = K(\lambda|p)$, since $K(\mathcal{O})$ is a constant independent of p, λ or p'. As well, we can show the reverse inequality. To produce λ from p, we can first produce p' from p (with a program that takes at least $K(p'|p)$ bits) and then build the set $S_{p,p'} = \{\lambda : |\lambda| \leq n, \mathcal{O}(p, \lambda) = p'\}$ of all compatible λ, by a program whose length is $O(1)$. Finally, we just have to give the index of λ in this set, and so $K(\lambda|p) \leq K(p'|p) + \log \# S_{p,p'}$. Then, by assumptions on λ, $K(p'|p) \geq K(\lambda|p) - \log \# S_{p,p'} \geq n - \alpha - \beta - \log \# S_{p,p'}$. By assumption on the output of $\mathcal{O}$, $K(p') \leq |\mathcal{O}(p, \lambda)| \leq n + \gamma$. This gives $K(p|p') \geq K(p) + n - \alpha - \beta - O(\log n) - K(p') \geq K(p) - \alpha - \beta - \gamma - O(\log n)$. □

The randomness and independence conditions for the keys are natural. The other two conditions may seem harder to justify, but they ensure that $\mathcal{O}$ effectively mixes p with the randomness provided by λ: the limit on the size of subsets of $\mathcal{L}$ implies a lower bound on the number of possible obfuscations for p (the more the better); on the other hand, the limit on the length of the output of $\mathcal{O}$ forces the information contained in p to be scrambled with λ, since it can take only a few more bits than those required to describe λ itself. The extreme case is similar to One-Time Pad encryption[7], when both the output and λ have the same size n, and $\mathcal{O}$ is injective for each p: there are 2^n keys, as well as possible obfuscations for each p. Furthermore, because the obfuscated code has the same length of λ, the exact same obfuscated strings are possible for each p, maximizing security.

In general, a program is composed of several of these minimal blocks in sequence. The above proof shows when the obfuscated block p' gives no information about its original block, say p_0. As well, p' can not give any more information about any other block p_1, as there is no causal relation between p_1 and p'. At best, there is some information in p_1 about p_0, but then the information given by p' about p_1 should be at most that given by about p_0. Therefore, we conclude that if all the sub-blocks in a program are securely obfuscated, then the whole program is. The above theorem, then, shows that secure obfuscation is possible under very reasonable assumptions.

[7]Which is proved to be an unconditionally secure symmetric cypher

5 INDIVIDUAL SECURITY OF CODE OBFUSCATION

Studying the security of individual instances of obfuscated code provides more granularity. Even if the obfuscated program is considered secure according to our definition, it may have parts which can provide some information about other obfuscated parts, which reduce the security of the obfuscated code. It could be that a program is obfuscated but that some module is not: some part of the obfuscated code stays still in its original form. We can demonstrate this relation by providing some boundaries on the complexity of subprograms in the same program.

We can view (obfuscated) programs as finite, and therefore recursively enumerable, sets of subprograms (blocks or modules) such that $P' = \{p'_n\}_{n \in \mathbb{N}}$. Given an obfuscated program P', it may consist of obfuscated and unobfuscated modules, that is: $\exists p'_j, p'_i \in P'$, where p'_i is an obfuscated module and p'_j is an unobfuscated module.

Theorem 10 demonstrates the effect of security parameters λ on the whole obfuscation process. The following results show the effect of each individual security parameter on each obfuscated subprogram. Choosing a good λ (with a good source of randomness) requires a minimum amount of information to be shared with obfuscated code and the clear code. It is important to study the relation between the security parameter and the original (clear) code. In the following theorem, we use a simple way to check the independence between λ and P on the subprogram level.

Theorem 13. *Let P' be a set of obfuscated subprograms and P a set of clear subprograms, such that each subprogram p' of P' has length at most n, and is the obfuscation of a corresponding block in P: $\exists p_i \in P, \kappa_i \in \lambda$. $p' = \mathcal{O}(p_i, \kappa_i)$. If $K(\kappa_i, p_i) \geq K(p_i) + K(\kappa_i) - \alpha$, then (up to a logarithmic term):*

$$I_K(\kappa_i; p_i) \leq \alpha$$

Proof. Since $K(p_i, \kappa_i) \leq K(\kappa_i|p_i) + K(p_i) +$

$O(\log n)$, we have by assumption

$$K(p_i) + K(\kappa_i) - \alpha \le K(\kappa_i|p_i) + K(p_i) + O(\log n)$$
$$K(\kappa_i) - \alpha \le K(\kappa_i|p_i) + O(\log n)$$
$$K(\kappa_i) - K(\kappa_i|p_i) \le \alpha + O(\log n)$$
$$I_K(\kappa_i, p_i) \le \alpha + O(\log n)$$

□

The following two theorems address the security of two different forms of code obfuscation: obfuscation-as-encoding and obfuscation-as-hiding. In the obfuscation-as-encoding technique, the original program is transformed in such a way it changes the structure of original code, but preserving the functionality, for example Data transformation techniques such as array splitting, splitting variable, Restructure Arrays and Merge Scalar Variables (Collberg et al., 1997). The encoding process is considered as the security parameter that dictates how the obfuscation should be performed and where it should take place. Encoding differs from encryption, if somebody knows the encoding process, then the original code can be recovered. In encoding obfuscation, the clear program is not presented in the obfuscated code; what still exists, but hidden, is the encoding process. Reversing the encoded program (obfuscated) requires finding and understanding the encoding process. For instance we used a simple encoding in Fig. 1, `x=x+i` is encoded as `x=x+4*i;x=x-2*i;x=x-i`; the encoding process converts `i` to `4*i;-2*i;-i`, to figure out `x=x+i` , we have to find and combine `4*i;-2*i;-i`, which is the security parameter in this case.

The next theorem addresses the security of obfuscation code when we apply encoding to P (as a set of subprograms), here, the encoding process is presented as a part of security parameter.

Theorem 14 (Encoding). *Let P' be a collection of obfuscated subprograms p'_i using $\kappa_i \in \lambda$, each of length at most n. Then,*

$$I_K(\kappa_i; p'_i) \le \delta_{\kappa_i} - O(1),$$

for $\delta_{\kappa_i} = \delta(\kappa_i|P')$.

Proof. Since P' is a collection of sub-programs, we can assume that it contains all the information in p'_i as well as that of all other sub-programs. Then,

$K(\kappa_i|P') = K(\kappa_i|p'_1, \ldots, p'_i, \ldots, p'_n) \le K(\kappa_i|p'_i)$ and so

$$\begin{aligned} I_K(\kappa_i; p'_i) &= K(\kappa_i) - K(\kappa_i|p'_i) \\ &\le K(\kappa_i) - K(\kappa_i|P') \\ &\le K(\kappa_i) - (\log \#P' - \delta_{\kappa_i}) \text{ by Definition 4} \end{aligned}$$

Because each subprogram has length at most n, P' can contain at most 2^n distinct programs. Assuming that each appears at most a constant number of times, we have that $\#P' = O(2^n)$ and $\log \#P' = n + O(1)$. Then,

$$\begin{aligned} I_K(\kappa_i; p'_i) &\le n - n + \delta_{\kappa_i} - O(1) \\ &\le \delta_{\kappa_i} - O(1) \end{aligned}$$

□

In hiding obfuscation techniques the original subprogram still exists in the obfuscated program (set of obfuscated subprograms), the security of such techniques depends on the degree of hiding in the set obfuscated subprograms. An example of such technique is the control flow obfuscations such as Insert Dead basic-blocks, Loop Unrolling, Opaque Predicate and Flatten Control Flow (Collberg et al., 1997). Normally these techniques are combined and used with encoding obfuscation techniques, in order to make the code more resilient to reverse engineering techniques.

For code obfuscation, an opaque predicate is used as a guard predicate that cannot statistically be computed without running the code; however the original code still exists too in the obfuscated code, but protected by the predicate. In Fig. 1 we used opaque predicates with simple data encoding technique. Consider the following obfuscated code of Fig. 1-(a), where the encoding has been removed. Obviously, `x=x+i` is still in the code, but is hidden under the protection of opaque predicate.

```
while(i<n){
  i=i+1
if (7*y*y-1==x*x){ //false
  y=x*i;
else
  x=x+i;}}
```

opaque predicate with no encoding

Theorem 15 (Hiding). *Let P' be a collection of obfuscated subprograms p'_i, each of length at most n. Then,*

$$I_K(p_i; p'_i) \le \delta_{p_i} - O(1),$$

for $\delta_{p_i} = \delta(p_i|P')$.

Proof. The proof is very similar to Theorem 14. The block p_i is hidden in P' but in its original form, due to the obfuscation process. Since P' is a collection of sub-programs, we can assume that it contains all the information in p'_i as well as that of all other subprograms. Then,

$K(p_i|P') = K(p_i|p'_1, \ldots, p'_i, \ldots, p'_n) \le K(p_i|p'_i)$

$$
\begin{aligned}
I_K(p_i; p_i') &= K(p_i) - K(p_i|p_i') \\
&\leq K(p_i) - K(p_i|P') \\
&\leq K(p_i) - (\log \#P' - \delta_{p_i}) \text{ by Definition 4}
\end{aligned}
$$

Similarly to the proof of Theorem 14, $\#P' = O(2^n)$ and $\log \#P' = n + O(1)$. Then,

$$
\begin{aligned}
I_K(p_i; p_i') &\leq n - n + \delta_{p_i} - O(1) \\
&\leq \delta_{p_i} - O(1)
\end{aligned}
$$

□

6 CONCLUSION AND FUTURE WORK

In this paper, we provide a theoretical investigation of code obfuscation. We defined code obfuscation using Kolmogorov complexity and algorithmic mutual information. Our definition allows for a small amount of secret information to be revealed to an adversary, and it gives an intuitive guarantee about the security conditions that have to be met for secure obfuscation. We argued our definition is more lenient than the virtual black-box model of Barak et al. and that for that reason the impossibility result does not apply. In contrast, we showed that under reasonable circumstances we can have secure obfuscation according to our definition.

To the best of our knowledge, this paper is the first to propose algorithmic information theory as a theoretical basis for code obfuscation. We believe that our new definition for code obfuscation provides the first step toward establishing quantitative metrics for certifying code obfuscation techniques. Currently, we are working toward deriving new metrics based on our model, aiming to validate and apply these metrics, empirically, to real obfuscated programs using state of the art obfuscation techniques.

There are still some questions we want to address in future work. For example, it is still not clear whether the complexity of security parameter (key) has always a positive effect on the the security of obfuscated programs based on algorithmic mutual information definition. Furthermore, algorithmic mutual information has a parallel counterpart based on classical information theory such as Shannon mutual information, it would be interesting to explore the relation between our definition and Shannon mutual information in the context of code obfuscation security. We are also planning to study and characterize the security of particular techniques and to analyze more carefully the scenario of active adversaries.

ACKNOWLEDGEMENTS

This research is partially funded by EPSRC-DTA (Mohsen). We would like to thank Steffen van Bakel and Emil Lupu for their comments.

REFERENCES

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., and Yang, K. (2012). On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48.

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. P., and Yang, K. (2001). On the (im)possibility of obfuscating programs. *IACR Cryptology ePrint Archive*, 2001:69.

Collberg, C., Thomborson, C., and Low, D. (1997). A Taxonomy of Obfuscating Transformations.

Gács, P. (1974). On the symmetry of algorithmic information. *Soviet Math. Dokl*, 15:1477–1480.

Garg, S., Raykova, M., Gentry, C., Sahai, A., Halevi, S., and Waters, B. (2013). Candidate indistinguishability obfuscation and functional encryption for all circuits. In *In FOCS*.

Gauvrit, N., Zenil, H., and Delahaye, J. (2011). Assessing cognitive randomness: A kolmogorov complexity approach. *CoRR*, abs/1106.3059.

Goldwasser, S. and Rothblum, G. N. (2007). On best-possible obfuscation. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 194–213, Berlin, Heidelberg. Springer-Verlag.

Jbara, A. and Feitelson, D. G. (2014). On the effect of code regularity on comprehension. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 189–200, New York, NY, USA. ACM.

Kieffer, J. C. and Yang, E. H. (1996). Sequential codes, lossless compression of individual sequences, and Kolmogorov complexity. *IEEE Trans. on Information Theory*, 42(1):29–39.

Lathrop, J. I. (1997). Compression depth and the behavior of cellular automata. *Complex Systems*.

Li, M. and Vitányi, P. M. (2008). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 edition.

McCabe, T. J. (1976). A complexity measure. *IEEE Trans. Software Eng.*, 2(4):308–320.

Shen, A. (1982). Axiomatic description of the entropy notion for finite objects. *VIII All-USSR Conference (Logika i metodologija nauki),Vilnjus*, pages 104 – 105. The paper in Russian.

Shen, A., Uspensky, V., and Vereshchagin, N. (2014). *Kolmogorov complexity and algorithmic randomness*. MCCME Publishing house.

Taveneaux, A. (2011). Towards an axiomatic system for kolmogorov complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:14.