Zero-Knowledge and Code Obfuscation

Satoshi Hada

Tokyo Research Laboratory, IBM Research,
1623-14, Shimotsuruma, Yamato, Kanagawa 242-8502, Japan.
satoshih@jp.ibm.com

Abstract. In this paper, we investigate the gap between auxiliary-input zero-knowledge (AIZK) and blackbox-simulation zero-knowledge (BSZK). It is an interesting open problem whether or not there exists a protocol which achieves AIZK, but not BSZK. We show that the existence of such a protocol is closely related to the existence of secure code obfuscators. A code obfuscator is used to convert a code into an equivalent one that is difficult to reverse-engineer. This paper provides security definitions of code obfuscation. By their definitions, it is easy to see that the existence of the gap implies the existence of a cheating verifier such that it is impossible to obfuscate any code of it. Intuitively, this means that it is possible to reverse-engineer any code of such a cheating verifier. Furthermore, we consider the actual behavior of such a cheating verifier. In order to do so, we focus on two special cases in which the gap exists: (1) there exists a *constant round public-coin* AIZK interactive argument for a language outside of $\mathcal{BPP}$. (2) there exists a *3-round secret-coin* AIZK interactive argument for a language outside of $\mathcal{BPP}$. In the former case, we show that it is impossible to securely obfuscate a code of a cheating verifier behaving as a pseudorandom function. A similar result is shown also in the latter case. Our results imply that any construction of constant round public-coin or 3-round secret-coin AIZK arguments for non-trivial languages essentially requires a computational assumption with a reverse-engineering property.

Keywords: Zero-knowledge, code obfuscation, reverse-engineering, interactive proof, interactive argument.

1 Introduction

In this paper, we investigate the gap between two definitions of zero-knowledge (ZK): auxiliary-input zero-knowledge and blackbox-simulation zero-knowledge. We will show that the gap is closely related to code obfuscation.

1.1 Zero-Knowledge

ZK is one of the most important notions in modern cryptography. The original definition of ZK, which we call GMR-ZK, is given in [GMR85] as follows: For

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 443–457, 2000.
© Springer-Verlag Berlin Heidelberg 2000

any probabilistic polynomial-time (PPT) cheating verifier, there exists a PPT machine (called the *simulator*) which simulates the interaction, i.e., produces a probability distribution which is computationally indistinguishable from the distribution of the real interaction between the prover and the cheating verifier. This definition is not suitable for cryptographic applications since it is not closed under sequential composition [GoKr96]. In cryptographic applications, the cheating verifier may have some additional a-priori information. However, GMR-ZK did not deal with this stronger verifier. In order to overcome this problem, Goldreich and Oren introduced an alternative definition called auxiliary-input zero-knowledge (AIZK) [GoOr94]. AIZK is a stronger formulation in which the simulation requirement is extended to deal with stronger (non-uniform) verifiers, namely, verifiers with some additional a-priori information. They showed that AIZK is closed under sequential composition.

The above two definitions only require that, for each cheating verifier, there exists a simulator. That is, in both definitions, the simulator is allowed to examine the internal state of the cheating verifier. On the other hand, blackbox-simulation zero-knowledge (BSZK) requires that the existence of a single universal simulator which uses any non-uniform cheating verifier as a blackbox to simulate the interaction. That is, the simulator is only allowed to simply observe the input/output behavior of the cheating verifier. BSZK is most restrictive among three definitions. Nevertheless, almost all known ZK protocols are BSZK [1]. It is an interesting open problem whether there exists a protocol which achieves AIZK, but not BSZK.

1.2 Code Obfuscation

Given a code, how can we make it hard to reverse-engineer it ? This is one of major open problems concerning computer practice. Code obfuscation is the most viable method for preventing reverse-engineering. There are many heuristic and ad-hoc obfuscation techniques for particular programming languages such as C, C++, Java and so on [CTL97]. However, to the best of our knowledge, no theoretical treatment have been provided so far. In this paper, we provide the definitions of secure code obfuscators and show that the existence of secure obfuscators for some code is closely related to the gap between AIZK and BSZK.

Take pseudorandom function ensembles (PRFEs) for example [GGM86]. PRFEs are function ensembles that can not be distinguished from truly random functions by any efficient procedure (any adversary) which can get the value of the function at arguments of its choice, provided its seed is chosen randomly. However, the pseudorandomness is guaranteed only when the randomly chosen seed is unknown to adversaries. This means that if a code of a PRFE is given to an adversary and the seed is embedded into the code, it may no longer satisfy the pseudorandomness. This is because some information about the seed may be extracted from the given code. A code obfuscator can be used to solve this problem. It converts the given code into another code that is functionally

[1] The one exception appeared in [HT98]. See Section 1.3.

identical to the original code so that the seed remain unknown to the adversary who is allowed to analyze the obfuscated code.

We sketch a definition of secure code obfuscators from the above point of view. We consider code obfuscation for a function ensemble $\mathcal{F} = \{F_n\}_{n\in\mathsf{N}}$ such that $F_n = \{f_s\}_{s\in\{0,1\}^{l_s(n)}}$. s is the seed that we want to remain unknown to an adversary. A code obfuscator C for $\mathcal{F}$, given a code $\pi(f_s)$, produces another code denoted by $\Pi(f_s)$ that is functionally identical to $\pi(f_s)$. We want to guarantee that the adversary should not gain any information about s given $\Pi(f_s)$ when s is chosen randomly. That is, we say that C is secure against an adversary A if whatever can be gained by A having access to the code $\Pi(f_s)$, can also be gained by a PPT machine having only blackbox-access to f_s. Roughly speaking, this guarantees that A can not reverse-engineer the code $\Pi(f_s)$ produced by C. This is formalized based on the simulation paradigm [GM84][GMR85] [Go99, Section 1.2.3]. In Section 3, we discuss defining secure code obfuscators in more detail.

1.3 Motivation and Results

As described in Section 1.1, it is an interesting open problem whether it holds that $Cl(BSZK) \subset Cl(AIZK)$, where $Cl(def)$ denotes the class of all interactive arguments satisfying the requirements of ZK definition def. Hada and Tanaka have constructed a 3-round *secret-coin* AIZK argument for an $\mathcal{NP}$-complete language, that is, they have shown that it holds that $Cl(BSZK) \subset Cl(AIZK)$ unless $\mathcal{NP}\subseteq\mathcal{BPP}$[HT98]. However, their construction requires a non-standard computational assumption with a strong reverse-engineering property. Roughly, it requires that given any code of any cheating verifier, one can efficiently extract the secret-coin used by the cheating verifier while one having only blackbox-access to the code can not do it. This paper addresses a question of whether the reverse implication holds, i.e., whether some reverse-engineering property is essential for the gap between AIZK and BSZK. If it is true then $Cl(BSZK) \subset Cl(AIZK)$ implies some negative result for code obfuscation. The purpose of this paper is to give such negative results.

We discuss the gap between AIZK and BSZK in more detail. We start by reviewing the definition of universal simulation zero-knowledge (USZK) introduced by Oren [Or87]. Oren showed that it is equivalent to AIZK, i.e., $Cl(USZK) = Cl(AIZK)$. In the definition of BSZK, given any cheating verifier, the simulator is required to output a simulated conversation simply by observing input/output behavior of the cheating verifier. On the other hand, the definition of USZK allows the simulator to take as input the code of the cheating verifier and analyze it. Therefore, we can say that if it holds that $Cl(BSZK) \subset Cl(USZK)$ (equivalent to $Cl(BSZK) \subset Cl(AIZK)$), then there exists a cheating verifier for which the simulation is possible by analyzing the code of it, but impossible by simply observing its input/output behavior. This will imply that it is impossible to obfuscate the code of such a cheating verifier. Indeed, by our security definition of code obfuscators, if it holds that $Cl(BSZK) \subset Cl(USZK)$ then there exists a cheating verifier (which can be

viewed as a function ensemble) for which secure code obfuscation is impossible (Theorem 3). However, this does not say anything about the actual behavior of such a cheating verifier.

We focus on the following two statements in order to consider the actual behavior of such a cheating verifier:

Gap_{pc}: There exists a constant round public-coin AIZK argument for a language outside of $\mathcal{BPP}$.

Gap_{sc}: There exists a 3-round secret-coin AIZK argument for a language outside of $\mathcal{BPP}$.

If either statement is true then it holds that $Cl(BSZK) \subset Cl(AIZK)$. Note that the analogous statements regarding BSZK are false [GoKr96] [2]. For each statement, we prove impossibility of secure code obfuscation for a specific cheating verifier (a specific function ensemble):

1. If Gap_{pc} is true, then there exists a cheating verifier behaving as a PRFE for which secure code obfuscators do not exist. In other words, there exists no secure code obfuscator for PRFEs (Theorem 4).

2. If Gap_{sc} is true, there exists a cheating verifier for which secure code obfuscators do not exist. The cheating verifier is equivalent to the prescribed (honest) verifier except that it computes its message as a pseudorandom function of the first message sent by the prover (Theorem 5).

We don't know whether these non-existence results are reasonable or not. However, our result is a good reason to believe that any construction of constant round public-coin or 3-round secret-coin AIZK arguments for non-trivial languages essentially requires a computational assumption with a reverse-engineering property. Indeed, the 3-round AIZK argument constructed in [HT98] required a computational assumption with a reverse-engineering property.

1.4 Related Works

Goldreich and Ostrovsky gave a definition of *software-protecting compilers* and an efficient construction of it [GoOs96]. It is quite different from the code obfuscator considered in this paper. In their setting, a code is encrypted and can be executed by a CPU having the corresponding decryption key and adversaries try to reconstruct the code from the encrypted one. The adversary is allowed to execute the program on the random-access machine (RAM) on arbitrary inputs of its choice and modify the data between the CPU and the memory. On the other hand, our code obfuscator never encrypts a given code.

In [DNRS99], Dwork et. al. showed the relationships among 3-round public-coin ZK arguments, selective decommitment and Fiat-Shamir methodology. They pointed out that the problem they studied is closely related to code obfuscation. However, they gave no results.

[2] The discussion in [GoKr96] are for ZK interactive proofs. However, their results extend to ZK interactive arguments. See Remarks 6.4 and 6.5 in that paper.

2 Preliminaries

We say that a function $\nu(\cdot) : \mathsf{N} \to \mathsf{R}$ is negligible in n if for every polynomial $poly(\cdot)$ and all sufficiently large n's, it holds that $\nu(n) < 1/poly(n)$. We often omit the expression "in n" when the definition of n will be clear by the context.

If S is any probability distribution then $x \leftarrow S$ denotes the operation of selecting an element randomly according to S. If S is a set then we use the same notation to denote the operation of picking an element x randomly from S. If A is a probabilistic machine then $A(x_1, x_2, \cdots, x_k)$ denotes the output distribution of A on inputs $(x_1, x_2, \cdots, x_k)$. Also, $\{x_1 \leftarrow S_1; x_2 \leftarrow S_2; \cdots x_k \leftarrow S_k : A(x_1, x_2, \cdots, x_k)\}$ denotes the output distribution of A on inputs $(x_1, x_2, \cdots, x_k)$ when the processes $x_1 \leftarrow S_1, x_2 \leftarrow S_2, \cdots, x_k \leftarrow S_k$ are performed in order. Let $\Pr[x \leftarrow S_1; x_2 \leftarrow S_2; \cdots; x_k \leftarrow S_k : E]$ denote the probability of the event E after the processes $x_1 \leftarrow S_1, x_2 \leftarrow S_2, \cdots, x_k \leftarrow S_k$ are performed in order.

We begin with reviewing the definitions of (non-uniform) computational indistinguishability and PRFEs.

Definition 1 (computational indistinguishability). *We define two types of computational indistinguishability.*

1. *Two distribution ensembles indexed by* N *and* $\{0,1\}^*$, $X = \{X_{n,w}\}_{n\in\mathsf{N},w\in\{0,1\}^*}$ *and* $Y = \{Y_{n,w}\}_{n\in\mathsf{N},w\in\{0,1\}^*}$ *are* computationally indistinguishable *if for every PPT machine D (the distinguisher), every polynomial $poly(\cdot)$, all sufficiently large n's and every string $z \in \{0,1\}^*$,*

$$|\Pr\left[\begin{array}{l} x \leftarrow X_{n,w}; \\ b \leftarrow D(1^n, w, x, z) \end{array} : b = 1\right] - \Pr\left[\begin{array}{l} y \leftarrow Y_{n,w}; \\ b \leftarrow D(1^n, w, y, z) \end{array} : b = 1\right]| < \frac{1}{poly(n)}.$$

2. *Two distribution ensembles indexed by a string set S and $\{0,1\}^*$, $X = \{X_{s,w}\}_{s\in S,w\in\{0,1\}^*}$ and $Y = \{Y_{s,w}\}_{s\in S,w\in\{0,1\}^*}$ are* computationally indistinguishable *if for every PPT machine D (the distinguisher), every polynomial $poly(\cdot)$ and all sufficiently long s's and every string $z \in \{0,1\}^*$,*

$$|\Pr\left[\begin{array}{l} x \leftarrow X_{s,w}; \\ b \leftarrow D(s, w, x, z) \end{array} : b = 1\right] - \Pr\left[\begin{array}{l} y \leftarrow Y_{s,w}; \\ b \leftarrow D(s, w, y, z) \end{array} : b = 1\right]| < \frac{1}{poly(|s|)}.$$

Definition 2 (function ensembles). *An* (l_{in}, l_{out}, l_s)-function ensemble *is a sequence $\mathcal{F} = \{F_n\}_{n\in\mathsf{N}}$ of function family $F_n = \{f_s : \{0,1\}^{l_{in}(n)} \to \{0,1\}^{l_{out}(n)}\}_{s\in\{0,1\}^{l_s(n)}}$, such that there exists a polynomial-time machine $Eval_{\mathcal{F}}$ (called the evaluator) so that for all $s \in \{0,1\}^{l_s(n)}$ and $x \in \{0,1\}^{l_{in}(n)}$, $Eval_{\mathcal{F}}(s, x) = f_s(x)$. In the sequel, we call s the* seed *of the function f_s. Also, we say that $\mathcal{F}$ is* non-uniformly computable *if the evaluator $Eval_{\mathcal{F}}$ is a non-uniform polynomial-time machine [Go99, Section A.2.3].*

Definition 3 (pseudorandom function ensembles (PRFEs)). *Let $\mathcal{U}_{l_{in},l_{out}} = \{U_{l_{in}(n),l_{out}(n)} : \{0,1\}^{l_{in}(n)} \to \{0,1\}^{l_{out}(n)}\}_{n\in\mathsf{N}}$ be a uniform function*

ensemble, i.e., $U_{l_{in}(n),l_{out}(n)}$ *is uniformly distributed over the set of* $\{0,1\}^{l_{in}(n)} \to \{0,1\}^{l_{out}(n)}$ *functions. An* (l_{in}, l_{out}, l_s)*-function ensemble* $\mathcal{F}$ *is called* pseudorandom *if for every PPT machine* M*, the following two distribution ensembles are computationally indistinguishable:* $\{s \leftarrow \{0,1\}^{l_s(n)} : M^{f_s}(1^n)\}_{n\in\mathsf{N}}$ *and* $\{u \leftarrow U_{l_{in}(n),l_{out}(n)} : M^u(1^n)\}_{n\in\mathsf{N}}$.

2.1 Interactive Arguments

We consider two probabilistic interactive machines called the prover and the verifier. The verifier is always a PPT machine. Initially both machines have access to a common input tape which includes x of length n. The prover and the verifier send messages to one another through two communication tapes. After exchanging a polynomial number of messages, the verifier stops in an accept state or in a reject state. Each machine only sees its own tapes, namely, the common input tape, the random tape, the auxiliary-input tape and the communication tapes. We denote by $A(x, y, m)$ the next message of A when x is the common input, y the auxiliary-input and m the messages exchanged so far. When we want to make explicit the random coins R used, we denote it by $A(x, y, m; R)$.

Let $\langle P_{x,w}, V_{x,y}\rangle$ denote the distribution of the decision (over $\{\mathsf{Acc}, \mathsf{Rej}\}$) of the verifier V having an auxiliary-input y when interacting on a common input x with the prover P having an auxiliary-input w, where the probability is taken over the random tapes of both machines. When auxiliary inputs w or y are empty, we omit them from $\langle P_{x,w}, V_{x,y}\rangle$ (e.g. $\langle P_{x,w}, V_x\rangle$ and $\langle P_x, V_{x,y}\rangle$).

There are two kinds of interactive protocols. One is "interactive proof" and the other is "interactive argument". The former requires that even a computationally unrestricted prover should be unable to make the verifier accept $x \notin L$, except with negligible (in n) probability [GMR85]. On the other hand, the latter requires that any cheating prover restricted to PPT should be unable to make the verifier accept $x \notin L$, except with negligible (in n) probability [BrCr86]. In this paper, we deal with interactive arguments.

Definition 4 (interactive arguments [Go99, page 62]). *Let* P, V *be two PPT interactive machines. The verifier* V *does not take any auxiliary-input. We say that* (P, V) *is an* interactive argument *for* L *if the following two conditions hold:* (1) Efficient Completeness: *For every polynomial* $poly(\cdot)$*, all sufficiently long* $x \in L$*, there exists an auxiliary-input* w *such that* $\Pr[b \leftarrow \langle P_{x,w}, V_x\rangle : b = \mathsf{Acc}] > 1 - 1/poly(|x|)$. (2) Computational Soundness: *For every PPT machine* $\hat{P}$ *(the polynomial-time bounded cheating prover), every polynomial* $poly(\cdot)$*, all sufficiently long* $x \notin L$ *and every auxiliary-input* w*,* $\Pr[b \leftarrow \langle P_{x,w}, V_x\rangle : b = \mathsf{Rej}] > 1 - 1/poly(|x|)$.

2.2 Zero-Knowledge

We recall the three definitions of AZIK, USZK and BSZK. A *view* of the verifier is a distribution ensemble which consists of the common input, the verifier's auxiliary input, the verifier's random coins and the sequence of messages sent by the

prover and the verifier during the interaction. Let $\mathsf{View}(P_x, V_{x,y}) = [x, y, m; R]$ denote V's view after interacting with P, where x is the common input, y the auxiliary input to V, R the random coins of V and m the sequence of messages sent by P and V. When the auxiliary input y to V is empty, we write $\mathsf{View}(P_x, V_x)$. When the random coins R used by V is fixed, we write $\mathsf{View}(P_x, V^R_{x,y})$ or $\mathsf{View}(P_x, V^R_x)$. For simplicity, when we talk about ZK, we omit the auxiliary-input to the prover P.

Definition 5 (AIZK [GoOr94]). *Let P, V be two probabilistic interactive machines. We say that (P, V) is* auxiliary-input zero-knowledge *for L if for every PPT machine $\hat{V}$ (the cheating verifier), there exists a PPT machine $S_{\hat{V}}$ (the simulator) such that the following two distribution ensembles are computationally indistinguishable: $\{S_{\hat{V}}(x, y)\}_{x\in L, y\in\{0,1\}^*}$ and $\{\mathsf{View}(P_x, \hat{V}_{x,y})\}_{x\in L, y\in\{0,1\}^*}$.*

Next, we recall the definition of USZK. For every polynomial $Q(\cdot)$, we denote by $V_{P,Q}$ the set of probabilistic *non-uniform* polynomial-time machines whose running time when interacting with the prover P is bounded by Q. It is important to note that $V \in V_{P,Q}$ is allowed to access an infinite advice sequence $a_1, a_2, \cdots$ such that $|a_n| < Q(n)$ [3]. V is not allowed to take any auxiliary-input, but instead allowed to use the advice string $a_{|x|}$ when x is a common input (So the auxiliary-input in V's view is always empty). Note that the encoding of a non-uniform polynomial-time machine V is an infinite sequence $EN_1(V), EN_2(V), \cdots$. Denote by $EN_n(V)$ an encoding of a machine V running on a common input x of length n, where a_n is incorporated into the encoding [4]. We denote by $EN(V)$ this sequence.

USZK allows the simulator to take as input the encoding of a cheating verifier.

Definition 6 (USZK [Or87]). *Let P, V be two probabilistic interactive machines. We say that (P, V) is* universal simulation zero-knowledge *for L if there exists a PPT machine US (the universal simulator) such that for every polynomial $Q(\cdot)$ and every $\hat{V} \in V_{P,Q}$, the following two distribution ensembles are computationally indistinguishable: $\{US(x, EN_{|x|}(\hat{V}))\}_{x\in L, EN_{|x|}(\hat{V})\in EN(\hat{V})}$ and $\{\mathsf{View}(P_x, \hat{V}_x)\}_{x\in L, EN_{|x|}(\hat{V})\in EN(\hat{V})}$, where EN is an arbitrary encoding.*

Finally, we recall the definition of BSZK, where the simulator is only allowed to use the cheating verifier as a blackbox.

Definition 7 (BSZK [GoOr94]). *Let P, V be two probabilistic interactive machines. We say that (P, V) is* blackbox-simulation zero-knowledge *for L if there exists a PPT machine BS (the blackbox-simulator) such that for every polynomial $Q(\cdot)$ and every $\hat{V} \in V_{P,Q}$, the following two distribution ensembles are computationally indistinguishable even when the distinguishers are allowed blackbox access to $\hat{V}$: $\{BS^{\hat{V}}(x)\}_{x\in L}$ and $\{\mathsf{View}(P_x, \hat{V}_x)\}_{x\in L}$.*

[3] Refer to [Go99, Section A.2.3] for more detail of non-uniform polynomial-time machines.

[4] Refer to [HU79, Section 8.3] for an example of the encoding of machines.

The following theorem says that USZK is equivalent to AIZK [Or87].

Theorem 1 (Oren [Or87]). $Cl(AIZK) = Cl(USZK)$.

3 Defining Secure Code Obfuscator

In this section, we provide security definitions of code obfuscators. We deal with the code obfuscators for function ensembles [5]. For simplicity, we identify a function f_s with its evaluator, i.e., a (non-uniform) polynomial-time machine which evaluates it. That is, the encoding of a function means the encoding of the machine which evaluates it. If the function is non-uniformly computable, the encoding must be done depending on the input length.

Definition 8 (code obfuscator). *Let $\mathcal{F}$ be a function ensemble. Let $\pi(\cdot)$ be any encoding. A* code obfuscator *C for $\mathcal{F}$ is a PPT machine, which takes as input a code $\pi(f_s)$ of f_s and outputs another code $\Pi(f_s)$ which is also a code of f_s.*

As sketched in Section 1.2, we try to define the security of code obfuscators based on the simulation paradigm. We require that whatever can be gained by an adversary having access to the code $\Pi(f_s)$ produced by a code obfuscator, can also be gained by a PPT machine having only blackbox-access to the function f_s.

We first give two unsatisfactory definitions.

Definition 9. *A code obfuscator C for $\mathcal{F}$ is* semantically secure *if for every encoding π and every PPT machine A (the adversary), there exists a PPT machine M (the simulator) such that the following two distribution ensembles are computationally indistinguishable: $\{s \leftarrow \{0,1\}^{l_s(n)}; \Pi(f_s) \leftarrow C(\pi(f_s)) : A(1^n, \Pi(f_s), z)\}_{n\in\mathsf{N},z\in\{0,1\}^*}$ and $\{s \leftarrow \{0,1\}^{l_s(n)} : M^{f_s}(1^n, z)\}_{n\in\mathsf{N},z\in\{0,1\}^*}$.*

Consider a simulator which chooses a seed s' randomly, produces a obfuscated code $\Pi(f_{s'})$ and outputs $A(1^n, \Pi(f_{s'}))$. Clearly, this simulator can *perfectly* simulate the output of any adversary. Therefore, this definition does not make sense. By adding the obfuscated cod $\Pi(f_s)$ to two distributions, we can prevent a simulator from taking such a strategy.

Definition 10. *A code obfuscator C for $\mathcal{F}$ is* semantically secure *if for every encoding π and every PPT machine A (the adversary), there exists a PPT machine M (the simulator) such that the following two distribution ensembles are computationally indistinguishable: $\{s \leftarrow \{0,1\}^{l_s(n)}; \Pi(f_s) \leftarrow C(\pi(f_s)) : (\Pi(f_s), A(1^n, \Pi(f_s), z))\}_{n\in\mathsf{N},z\in\{0,1\}^*}$ and $\{s \leftarrow \{0,1\}^{l_s(n)} : (\Pi(f_s), M^{f_s}(1^n, z))\}_{n\in\mathsf{N},z\in\{0,1\}^*}$.*

[5] Although we can define the security for functions rather than function ensembles, we don't deal with them in this paper.

Again, this definition does not make sense. Consider an adversary who outputs the given obfuscated code as it is. For such an adversary, we can't expect the existence of a simulator which outputs some code of f_s by accessing f_s in blackbox fashion. Such a simulator exists only for a function ensemble that can be learned efficiently and such a function ensemble is not interesting here. For example, consider a function ensemble $\mathcal{F}_{\oplus} = \{F_n\}_{n\in\mathsf{N}}$ such that $F_n = \{f_s(x) = x \oplus s : \{0,1\}^n \to \{0,1\}^n\}_{s\in\{0,1\}^n}$. We can easily compute s by accessing f_s in blackbox fashion. Therefore, there is a trivial secure code obfuscator C for $\mathcal{F}_{\oplus}$, that is, C outputs a given code $\pi(f_s)$ as it is and the simulator M computes the seed s and outputs $\pi(f_s)$.

Due to the above failure, we restrict our attention to a particular adversary (rather than every adversary) and define the security only against it. We give two definitions: one is based on the simulation paradigm like the above and the other is based on indistinguishability of obfuscations.

Definition 11 (semantic security against an adversary). *A code obfuscator C for $\mathcal{F}$ is* semantically secure against *an adversary A (a PPT machine) if for every encoding π, there exists a PPT machine M (the simulator) such that the following two distribution ensembles are computationally indistinguishable:* $\{s \leftarrow \{0,1\}^{l_s(n)}; \Pi(f_s) \leftarrow C(\pi(f_s)) : (\Pi(f_s), A(1^n, \Pi(f_s), z))\}_{n\in\mathsf{N},z\in\{0,1\}^*}$ *and* $\{s \leftarrow \{0,1\}^{l_s(n)} : (\Pi(f_s),\ M^{f_s}(1^n, z))\}_{\ n\in\mathsf{N},z\in\{0,1\}^*}$.

Another definition is based on indistinguishability of obfuscated codes of a given pair of functions $(f_s, f_{s'})$.

Definition 12 (indistinguishable security against an adversary). *A code obfuscator C for $\mathcal{F}$ is* indistinguishably secure against *an adversary A (a PPT machine) if for every encoding π, the following two distribution ensembles are computationally indistinguishable:* $\{s \leftarrow \{0,1\}^{l_s(n)};\ \Pi(f_s) \leftarrow C(\pi(f_s)) : (\Pi(f_s), A(1^n, \Pi(f_s), z))\}_{n\in\mathsf{N},z\in\{0,1\}^*}$ *and* $\{s \leftarrow \{0,1\}^{l_s(n)};\ s' \leftarrow \{0,1\}^{l_s(n)};\ \Pi(f_{s'}) \leftarrow C(\pi(f_{s'})) : (\Pi(f_s), A(1^n, \Pi(f_{s'}), z))\}_{\ n\in\mathsf{N},z\in\{0,1\}^*}$.

Consider an adversary who outputs the size of a given code. Because there is a code obfuscator such that the size of an obfuscated code is uniquely determined from the size of the seed. Therefore, there exists a trivial secure code obfuscator which is indistinguishably secure against such an adversary.

It is easy to see that indistinguishable security implies semantic security. However, we don't know whether the reverse implication holds.

Theorem 2. *If a code obfuscator C for a function ensemble $\mathcal{F}$ is indistinguishably secure against an adversary A then it is semantically secure against A.*

We don't know whether there is a "non-trivial" function ensemble for which there is a code obfuscator semantically secure against a "non-trivial" adversary. As shown in the next section, this problem is related to the gap between AIZK and BSZK.

4 Zero-Knowledge and Code Obfuscation

In this section, we present our results. First of all, we show that the gap between AIZK and BSZK implies the existence of a cheating verifier for which no code obfuscators is semantically secure against an adversary.

Let (P, V) be an interactive argument in $Cl(USZK)$ but not in $Cl(BSZK)$. We consider a cheating verifier $\hat{V} \in V_{P,Q}$. Denote by $l_r(\cdot), l_m(\cdot)$ and $l_s(\cdot)$ the polynomials bounding the number of rounds, the length of a message and the number of random coins used by $\hat{V}$, respectively. We can identify $\hat{V}$ with a *non-uniformly computable* function ensemble $\hat{\mathcal{V}} = \{\hat{V}_n\}_{n\in\mathsf{N}}$, $\hat{V}_n = \{\hat{v}_s : \{0,1\}^{n+l_r(n)l_m(n)} \to \{0,1\}^{l_m(n)}\}_{s\in l_s(n)}$ such that $\hat{v}_s(x, m) = \hat{V}(x, -, m; s)$.

Since (P, V) achieves USZK, for every code obfuscator C for $\hat{\mathcal{V}}$, every encoding π and every cheating verifier $\hat{V} \in V_{P,Q}$, there exists a universal simulator US such that for every polynomial Q, the distribution ensemble $\{s \leftarrow \{0,1\}^{l_s(|x|)}; \Pi_{|x|}(\hat{v}_s) \leftarrow C(\pi_{|x|}(\hat{v}_s)) : (\Pi_{|x|}(\hat{v}_s), US(x, \Pi_{|x|}(\hat{v}_s)))\}_{x\in L}$ is computationally indistinguishable from a view of $\hat{V}$ interacting with P, that is, $\{s \leftarrow \{0,1\}^{l_s(|x|)}; \Pi_{|x|}(\hat{v}_s) \leftarrow C(\pi_{|x|}(\hat{v}_s)) : (\Pi_{|x|}(\hat{v}_s), \mathsf{View}(P_x, \hat{V}_x^s))\}_{x\in L}$. We consider the universal simulator US as an adversary.

On the other hand, since (P, V) does not achieve BSZK, there exists a polynomial Q and a non-uniform cheating verifier $\hat{V} \in V_{P,Q}$ such that for every blackbox-simulator BS, the distribution ensemble $\{s \leftarrow \{0,1\}^{l_s(|x|)}; \Pi_{|x|}(\hat{v}_s) \leftarrow C(\pi_{|x|}(\hat{v}_s)) : (\Pi_{|x|}(\hat{v}_s), BS^{\hat{v}_s}(x))\}_{x\in L}$ is **NOT** computationally indistinguishable from $\{s \leftarrow \{0,1\}^{l_s(|x|)}; \Pi_{|x|}(\hat{v}_s) \leftarrow C(\pi_{|x|}(\hat{v}_s)) : (\Pi_{|x|}(\hat{v}_s), \mathsf{View}(P_x, \hat{V}_x^s))\}_{x\in L}$.

From the above, the theorem follows:

Theorem 3. *Assume that it holds that $Cl(BSZK) \subset Cl(AIZK)$. Then no code obfuscator for $\hat{\mathcal{V}}$ is semantically secure against US.*

Now we know that $Cl(BSZK) \subset Cl(AIZK)$ implies the existence of a function ensemble for which no code obfuscator is semantically secure against an adversary. However, we don't know the actual behavior of such a function ensemble. In the rest of this section, we focus on two cases Gap_{pc} and Gap_{sc} to prove non-existence of secure code obfuscators for specific function ensembles.

4.1 The Case of Gap_{pc}

Theorem 4. *Assume that Gap_{pc} is true. Let $\mathcal{F}$ be any PRFE (both input and output length functions are specified in the proof). Then, no code obfuscator for $\mathcal{F}$ is semantically secure against an adversary (The behavior of the adversary is specified in the proof).*

Proof. Let (P_0, V_0) be a constant round public-coin AIZK argument for a language L outside of $\mathcal{BPP}$. We use the following notations for (P_0, V_0). Denote by x the common input and by n the length of x. For simplicity of the exposition we make some assumptions on the form of the protocol without loss of generality. We assume both the first and last messages are sent by P_0. By adding dummy

message any protocol can be converted into one of this form. Note that in such a protocol, the number of rounds is always an odd number $2m+1$, where m is a constant. The messages sent by P_0 are denoted by $\alpha_1, \alpha_2, \cdots, \alpha_m$ and γ (α_i is ith message and γ is $m+1$th message). The messages sent by V_0 are denoted by $\beta_1, \beta_2, \cdots, \beta_m$ (β_i is ith message). We assume that for every i, α_i and β_i have length $l_\alpha(n)$ and $l_\beta(n)$, respectively. We also assume that for every i, V_0 chooses β_i randomly in $\{0,1\}^{l_\beta(n)}$. The predicate computed by V_0 in order to decide whether to accept or reject is denoted by $\rho(x, \alpha_1, \beta_1, \cdots, \alpha_m, \beta_m, \gamma)$. That is, V_0 accepts x if and only if $\rho(x, \alpha_1, \beta_1, \cdots, \alpha_m, \beta_m, \gamma) = \mathsf{Acc}$. This predicate may be a randomized function.

Let $\mathcal{F}$ be an (l_α, l_β, l_s)-PRFE. We transform (P_0, V_0) into another interactive argument (P_1, V_1) using $\mathcal{F}$. P_1 is same as P_0. For every i ($1 \leq i \leq m$), V_1 computes $\beta_i = f_s(\alpha_i)$ instead of choosing it randomly, where s is randomly chosen from $\{0,1\}^{l_s(n)}$ at the beginning of the protocol only at once. From the pseudorandomness of $\mathcal{F}$, it follows that (P_1, V_1) is also a $(2m+1)$-round public-coin AIZK interactive argument for L.

Let C be a code obfuscator C for $\mathcal{F}$. We further transform (P_1, V_1) into a 2-round protocol (P_2, V_2) using C. The idea behind this transformation is that V_2 sends to the prover P_2 a code required for the computation of V_1 and makes P_2 compute the messages of V_1. Let π be any encoding.

Protocol: (P_2, V_2), where x is a common input of length n and w is an auxiliary input to P_2.
R1: V_2 randomly chooses s from $\{0,1\}^{l_s(n)}$ to get $\pi(f_s)$. Then V_2 use the code obfuscator C to produce a code $\Pi(f_s)$ and send it to P_2.
R2: Using the code $\Pi(f_s)$, P_2 computes $\alpha_1 \leftarrow P_0(x, w, -)$, $\beta_1 = f_s(\alpha_1)$, $\alpha_2 \leftarrow P_0(x, w, \alpha_1\beta_1)$, $\beta_2 = f_s(\alpha_2)$, $\alpha_3 \leftarrow P_0(x, w, \alpha_1\beta_1\alpha_2\beta_2)$, $\cdots$, $\beta_m = f_s(\alpha_m)$, $\gamma \leftarrow P_0(x, w, \alpha_1\beta_1 \cdots \alpha_m\beta_m)$. Then P_2 sends $(\alpha_1, \cdots, \alpha_m, \gamma)$ to V_2.
Decision: V_2 outputs $\rho(x, \alpha_1, f_s(\alpha_1), \cdots, \alpha_m, f_s(\alpha_m), \gamma)$.

Claim. (P_2, V_2) achieves AIZK. (P_2, V_2) satisfies the efficient completeness, but doesn't satisfy the computational soundness.

Proof. Firstly, we show that (P_2, V_2) achieves AIZK. We have a universal simulator US guaranteed by the USZK or AIZK property of (P_1, V_1). For every cheating verifier $\hat{V}_2 \in V_{P_2,Q}$, we can use US to simulate the conversation between P_2 and $\hat{V}_2$. The simulation is as follows: (i) Simulate $\hat{V}_2$ to get a code $\Pi(f_s)$. (ii) Produce the code of V_1 using f_s from $\Pi(f_s)$. We denote it by $\Pi(V_1^{f_s})$. (iii) Output $US(x, \Pi(V_1^{f_s}))$. It is easy to see that this output distribution is computationally indistinguishable from the real interaction between P_2 and $\hat{V}_2$.

The efficient completeness is clearly satisfied. From the triviality result regarding AIZK [GoOr94], it follows that if (P_2, V_2) satisfies the computational soundness, then $L \in \mathcal{BPP}$. This contradicts our assumption. Therefore, the computational soundness is not satisfied. □

Now we return to the proof of Theorem 4. We construct an adversary A for which any simulator fails to satisfy the requirement in Definition 11. Recall

that (P_2, V_2) does not satisfy the computational soundness (Claim 4.1). Let $\hat{P}_2$ be a cheating prover who can violate the computational soundness. Let x be a string of length n such that $x \notin L$. Let w be an auxiliary-input to $\hat{P}_2$. Given a code of a function f_s, A tries to output an accepting conversation. Given an input $(1^n, \Pi(f_s), (x, w))$, A simply outputs $\hat{P}_2(x, w, \Pi(f_s))$, i.e., the messages $(\alpha_1, \alpha_2, \cdots, \gamma)$. Since $\hat{P}_2$ violates the computational soundness, it follows that for every code obfuscator C for $\mathcal{F}$,

$$\Pr\left[\begin{array}{l} s \leftarrow \{0,1\}^{l_s(n)}; \Pi(f_s) \leftarrow C(\pi(f_s)); \\ (\alpha_1, \alpha_2, \cdots, \alpha_m, \gamma) \leftarrow A(1^n, \Pi(f_s), (x, w)); \\ b \leftarrow \rho(x, \alpha_1, f_s(\alpha_1), \cdots, \alpha_m, f_s(\alpha_m), \gamma) \end{array} : b = \mathsf{Acc}\right]$$

is **NOT** negligible in n.

On the other hand, from the argument in [GoKr96, Proof of Lemma 6.4], it follows that for every PPT machine M (simulator), every auxiliary-input w and every code obfuscator C for $\mathcal{F}$,

$$\Pr\left[\begin{array}{l} u \leftarrow U_{l_\alpha(n), l_\beta(n)}; (\alpha_1, \alpha_2, \cdots, \alpha_m, \gamma) \leftarrow M^u(1^n, (x, w)); \\ b \leftarrow \rho(x, \alpha_1, f_s(\alpha_1), \cdots, \alpha_m, f_s(\alpha_m), \gamma) \end{array} : b = \mathsf{Acc}\right]$$

is negligible in n. By the pseudorandomness of $\mathcal{F}$, we can replace the uniform function u by the function f_s. It follows that for every PPT machine M, every auxiliary-input w and every code obfuscator C,

$$\Pr\left[\begin{array}{l} s \leftarrow \{0,1\}^{l_s(n)}; (\alpha_1, \alpha_2, \cdots, \alpha_m, \gamma) \leftarrow M^{f_s}(1^n, (x, w)); \\ b \leftarrow \rho(x, \alpha_1, f_s(\alpha_1), \cdots, \alpha_m, f_s(\alpha_m), \gamma) \end{array} : b = \mathsf{Acc}\right]$$

is negligible in n. Since any simulator fails to simulate A, no code obfuscator C for $\mathcal{F}$ can be semantically secure against A. □

The theorem does not extend to the case of non-constant rounds since we use the argument in [GoKr96, Proof of Lemma 6.4].

4.2 The Case of Gap_{sc}

In this section, we consider the second case. The argument here is essentially equivalent to the one in the previous section.

Let (P_0, V_0) be a 3-round secret-coin AIZK argument for a language L outside of $\mathcal{BPP}$. We denote by α and γ the messages sent by P_0. We denote by β the message sent by V_0. Denote by R_{sc} the secret-coin used by V_0 to compute $\beta \leftarrow V_0(x, -, \alpha)$. The length functions of α, β, R_{sc} are denoted by $l_\alpha(\cdot), l_\beta(\cdot)$ and $l_{R_{sc}}(\cdot)$, respectively. The predicate computed by V_0 in order to decide whether to accept or reject is denoted by $\rho(x, \alpha, \gamma, R_{sc})$.

Let $\mathcal{F}$ be an $(l_\alpha, l_{R_{sc}}, l_s)$-PRFE. We transform (P_0, V_0) into another interactive argument (P_1, V_1) using $\mathcal{F}$. P_1 is same as P_0. V_1 computes β as follows: Chooses s randomly from $\{0,1\}^{l_s(n)}$, computes $R_{sc} = f_s(\alpha_i)$ and $\beta = V_0(x, -, \alpha; R_{sc})$. We denote by $\mathcal{CV} = \{CV_n\}_{n \in \mathsf{N}}$ a function ensemble such

that $CV_n = \{CV_s : CV_s(x, \alpha) = V_0(x, -, \alpha; f_s(\alpha))\}_{s \in \{0,1\}^{l_s(n)}}$. From the pseudorandomness of $\mathcal{F}$, it follows that (P_1, V_1) is a 3-round secret-coin AIZK interactive argument for L.

Now we prove our second theorem.

Theorem 5. *Assume that* Gap_{sc} *is true. Let* $\mathcal{CV}$ *be a function ensemble specified as the above. Then, no code obfuscator for* $\mathcal{CV}$ *is semantically secure against an adversary (The behavior of the adversary is specified in the proof).*

Proof. Let C be a code obfuscator C for $\mathcal{CV}$. We further transform (P_1, V_1) into a 2-round protocol (P_2, V_2) using C. Let π be any encoding.

Protocol: (P_2, V_2), where x is a common input of length n and w is an auxiliary input to P_2.
R1: V_2 randomly chooses s from $\{0,1\}^{l_s(n)}$ to get $\pi(CV_s)$. Then V_2 use the code obfuscator C to produce a code $\Pi(CV_s)$ and send it to P_2.
R2: Using the code $\Pi(CV_s)$, P_2 computes $\alpha \leftarrow P_0(x, w, -)$, $\beta = CV_s(x, \alpha) = V_0(x, -, \alpha; f_s(\alpha))$ and $\gamma \leftarrow P_0(x, w, \alpha\beta)$. Then P_2 sends (α, γ) to V_2.
Decision: V_2 outputs $\rho(x, \alpha, \gamma, f_s(\alpha))$.

Claim. (P_2, V_2) achieves AIZK. (P_2, V_2) satisfies the efficient completeness, but doesn't satisfy the computational soundness.

Proof. The proof is essentially equivalent to the one of Claim 4.1. □

The rest of the proof is equivalent to the corresponding one of Theorem 4 except that we use the argument in [GoKr96, Section 6.3] instead of the one in [GoKr96, Proof of Lemma 6.4]. The detail is omitted. □

The theorem does not extend to the case of more than 3 rounds since we use the argument in [GoKr96, Section 6.3].

5 Concluding Remarks

In this paper, we have shown the gap between $Cl(AIZK)$ and $Cl(BSZK)$ is closely related to code obfuscation. We have focused on the following two statements: (1) There exists a constant round public-coin AIZK argument for a language outside of $\mathcal{BPP}$. (2) There exists a 3-round secret-coin AIZK argument for a language outside of $\mathcal{BPP}$. We have shown that if these statements are true, it implies negative results for code obfuscation. If the former is true, there exists no semantically secure code obfuscator for a PRFE. A similar negative result regarding the latter statement has also been shown. We don't know whether these non-existence results are reasonable or not. However, our result is a good reason to believe that any construction of constant round public-coin or 3-round secret-coin AIZK arguments for non-trivial languages essentially requires a computational assumption with a reverse-engineering property. Indeed, the 3-round AIZK argument constructed in [HT98] requires a computational assumption with a reverse-engineering property.

Dwork et al. showed that if there exits a secure bit commitment function resilient to selective decommitment then there exist 3-round public-coin ZK arguments for any $\mathcal{NP}$ language [DNRS99]. It is an open problem whether there exists a bit commitment function resilient to selective decommitment. They showed that in several special cases, such a bit commitment function exists in a weaker sense. However, the weaker resilience seems to be insufficient for the existence of 3-round public-coin ZK arguments for an $\mathcal{NP}$ language. Combining their result with Theorem 4, we can easily obtain the following relationship: Under the assumption that it does not hold that $\mathcal{NP} \subseteq \mathcal{BPP}$, if there exits a secure bit commitment function resilient to selective decommitment, then there is no semantically secure code obfuscator for PRFEs[6].

We considered only the setting of ZK arguments, but the problem studied in this paper really applies to any setting where we need a simulator [GM84][Ca00]. It is interesting to investigate what could be proven in other settings.

References

[BrCr86] G. Brassard and C. Crépeau, "Non-Transitive Transfer of Confidence : A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond, " Proceedings of 27th FOCS, 1986.

[Ca00] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols," Journal of Cryptology, Vol.13, No. 1, pp.143-202, 2000.

[CTL97] C. Collberg, C. Thomborson and D. Low, "A Taxonomy of Obfuscating Transformations," Technical Report 148, Department of Computer Science, University of Auckland, 1997.

[DNRS99] C. Dwork, M. Naor, O. Reingold and L. Stockmeyer, "Magic Functions, " Proceedings of 40th FOCS, 1999.

[Go99] O. Goldreich, "Modern Cryptography, Probabilistic Proofs and Pseudorandomness," Algorithms and Combinatorics Vol.17, Springer, 1999.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, "How to Construct Random Functions," Journal of the ACM, Vol.33, No.4, pp.792-807, 1986.

[GoKr96] O. Goldreich and H. Krawczyk, "On the Composition of Zero-Knowledge Proof Systems," SIAM Journal on Computing, Vol.25, No.1, pp.169-192, 1996.

[GoOr94] O. Goldreich and Y. Oren, "Definitions and Properties of Zero-Knowledge Proof Systems," Journal of Cryptology, Vol.7, No. 1, pp.1-32, 1994.

[GoOs96] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," Journal of the ACM, Vol.43, No.3, pp.431-473, 1996.

[GM84] S. Goldwasser and S. Micali, "Probabilistic Encryption," J. Comput. System Sci., 28, pp.270-299, 1984.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proofs," Proceedings of 17th STOC, pp.291-304, 1985.

[HT98] S. Hada and T. Tanaka, "On the Existence of 3-Round Zero-Knowledge Protocols, " Proceedings of CRYPTO'98, pp. 408-423, 1998. A revised version is available as Theory of Cryptography Library: Record 99-9.

[6] This requires the assumption regarding the $\mathcal{BPP}$ v.s. $\mathcal{NP}$ problem. We can remove it by directly observing the definitions of secure bit commitment functions resilient to selective decommitment and secure code obfuscator.

[HU79] J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, 1979.

[Or87] Y. Oren, "On the Cunning Power of Cheating Verifiers: Some Observations about Zero-Knowledge Proofs," Proceedings of 28th FOCS, pp. 462-471, 1987.