A Note on (Im)Possibilities of Obfuscating Programs of Zero-Knowledge Proofs of Knowledge

Ning Ding and Dawu Gu

Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
{dingning,dwgu}@sjtu.edu.cn

Abstract. Program obfuscation seeks efficient methods to write programs in an incomprehensible way, while still preserving the functionalities of the programs. In this paper we continue this research w.r.t. zero-knowledge proofs of knowledge. Motivated by both theoretical and practical interests, we ask if the prover and verifier of a zero-knowledge proof of knowledge are obfuscatable. Our answer to this question is as follows. First we present two definitions of obfuscation for interactive probabilistic programs and then achieve the following results:

1. W.r.t. an average-case virtual black-box definition, we achieve some impossibilities of obfuscating provers of zero-knowledge and witness-indistinguishable proofs of knowledge. These results state that the honest prover with an instance and its witness hardwired of any zero-knowledge (or witness-indistinguishable) proof of knowledge of efficient prover's strategy is unobfuscatable if computing a witness (or a second witness) for this instance is hard. Moreover, we extend these results to t-composition setting and achieve similar results. These results imply that if an adversary obtains the prover's code (e.g. stealing a smartcard) he can indeed learn some knowledge from it beyond its functionality no matter what measures the card designer may use for resisting reverse-engineering.
2. W.r.t. a worst-case virtual black-box definition, we provide a possibility of obfuscating the honest verifier (with the public input hardwired) of Blum's 3-round zero-knowledge proof for Hamilton Cycle. Our investigation is motivated by an issue of *privacy protection* (e.g., if an adversary controls the verifier, he can obtain all provers' names and public inputs. Thus the provers' privacy may leak). We construct an obfuscator for the verifier, which implies that even if an adversary obtains the verifier's code, he cannot learn any knowledge, e.g. provers' names, from it. Thus we realize the anonymity of provers' accesses to the verifier and thus solve the issue of privacy protection.

Keywords: Program Obfuscation, Zero Knowledge, Witness Indistinguishability, Proofs of Knowledge.

D. Lin, G. Tsudik, and X. Wang (Eds.): CANS 2011, LNCS 7092, pp. 292–311, 2011.

1 Introduction

In recent years, cryptography community has focused on a fascinating research line of program obfuscation. Loosely speaking, obfuscating a program P is to construct a new program which can preserve the functionality of P, but its code is fully "unintelligent". Any adversary can only use the functionality of P and cannot learn anything more than this, i.e. cannot reverse-engineering nor understand it. In other words, an obfuscated program should not reveal anything useful beyond executing it.

[2] formalized the notion of obfuscation through a simulation-based definition called the virtual black-box property, which says that every adversary has a corresponding simulator that emulates the output of the adversary given oracle (i.e. black-box) access to the same functionality being obfuscated. Following [2], many works focused on how to obfuscate different cryptographic functionalities. Among them, there are some negative results, e.g. [2,19,22]. [2] showed there doesn't exist any general obfuscation method for all programs. [19,22] showed that this negative result holds even w.r.t. some other definitions of obfuscation. On the other hand, some obfuscation methods have been proposed for several cryptographic functionalities. [26,24] demonstrated how to securely obfuscate re-encryption and encrypted signature respectively. [13] showed that with fully homomorphic encryption in e.g. [17], a category of functionalities, which can be characterized as first secret operation and then public encryption, are obfuscatable. Some works e.g. [27,25,4,8,11,10,9] focused on obfuscation for a basic and simple primitive, i.e. (multiple-bit) point functions, traditionally used in some password based identification systems, and its applications in constructing encryption and signature schemes of strong security.

We continue this research line w.r.t. zero-knowledge [21] (as well as witness-indistinguishable [16]) protocols in this paper. Motivated by theoretical and practical interests, we ask whether or not the prover and verifier of a zero-knowledge (or witness-indistinguishable) proof or argument of knowledge are obfuscatable. Before demonstrating our motivation, we first present some works related to this issue. [23] first addressed this issue, which gave some definitions of obfuscators (which security allows the distinguisher to obtain the obfuscated program, like a stronger definition in [25], but only requires the indistinguishability holds w.r.t. an individual adversary). [23] showed that there exists a cheating verifier for a constant-round non-black-box zero-knowledge protocol which is unobfuscatable, while the first such protocol was later constructed by [1]. Actually the non-black-box simulator in [1] doesn't reverse-engineering verifiers' codes. [20] investigated obfuscation for protocols based on tamper-proof hardware, and showed that with the help of hardware, all programs including provers are one-time obfuscatable.

We now turn to our motivation. As we know, zero-knowledge proofs of knowledge are a secure way to implement identification as well as other tasks. We use identification as an example to illustrate the motivation. Briefly, an identification scheme consists of a trusted party and multi-clients and multi-servers. The trusted party generates a verifiable public input (a.k.a. statement or instance, e.g. a directed graph or a modular square) and a witness (e.g. a Hamilton cycle

or a modular square root) for each client. A client's name and its public input constitute a record and the trusted party maintains all records. Each client stores his witness secretly and when he needs to identify himself to a server, he proves to the server the knowledge of the witness via the identification scheme.

We now first present the motivation of obfuscating provers (with public inputs and witnesses hardwired). Consider the applications of smartcards, in which a holder of a smartcard may use his card to identify himself to a server via some zero-knowledge proof of knowledge where the card runs as prover and the server runs as verifier. If an adversary obtains the card, he can perform side channel attacks, physical attacks and white-box cryptoanalysis etc. to recover the witness (i.e. secret) in the prover's program. Thus card designers need to develop many countermeasures in designing prover's programs against such attacks. Since these approaches are heuristic without rigorous security proofs, they are usually broken by new developed attacks. Thus it is very interesting to ask if there is a way to write the prover's program such that any adversary cannot break it even if he possesses the actual program. Namely, can we obfuscate the prover? We think that this question is of both theoretical and practical interests.

We then present the motivation of obfuscating verifiers. Note that the impossibility in [23] says that a cheating verifier with its random coins hardwired is unobfuscatable, while herein we investigate obfuscation for honest verifiers without random coins hardwired. Thus an obfuscated verifier in this paper, if possible, is still a PPT algorithm, which needs fresh coins in execution. Due to this motivation, a question immediately arises. That is, a honest verifier doesn't contain any secret and is publicly available, so why do we need to obfuscate it? Our answer to this question is that what we need to obfuscate is a honest verifier with a *public input* hardwired rather than the verifier's strategy only, and the obfuscation for such verifiers is meaningful for an issue of *privacy protection*. (Some works e.g. [12] discussed the input privacy of zero-knowledge protocols, which especially work as sub-protocols of large cryptographic protocols. We here only concern the privacy issue for verifiers of zero-knowledge protocols in the stand alone setting and obfuscation seems to be a conceptually direct solution.)

For instance, consider the case of ID cards, which have been wildly used in our society. However, some secret governmental organizations may need to read data from cards for some secret goals. Before their card readers can read data from cards, they usually need to pass identification. In this scenario the readers of the organizations act as prover and cards act as verifier. Thus cards should store all legal clients' (i.e. organizations) names and their public inputs, while on the contrary these organizations may not want their names stored in any card since they would like to access cards anonymously without being identified. If cards indeed store their names, a holder (or an adversary) can of course extract these names from the verifier's program in his own card, which frustrates the anonymity of their accesses. Thus, to realize the anonymity for them, it is very natural to ask if we can obfuscate the verifier's program (incorporated with clients' names and public inputs) before an ID card is issued officially such that any holder or adversary cannot find this information from the card. More

discussion of this motivation can be referred to Section 5.1. In this paper we are interested in finding the answers to these questions, and correspondingly achieve the following results.

1.1 Our Results

As we see, most known works on obfuscation focused upon non-interactive programs, while the main objects in this paper, i.e. provers and verifiers, are interactive. Nevertheless, there are still some works focused upon interactive programs. For instance, [23] presented some definitions for interactive deterministic programs. In this paper we first discuss how to combine the known definitions with the probabilistic property of provers and verifiers to define obfuscation for interactive probabilistic programs. We then present two definitions. Both of these two definitions require a functionality property and a virtual black-box (VBB) property. They differs mainly in the characterization of the VBB property.

The first one adopts the average-case virtual black-box (ACVBB) requirement introduced by [25,26], which is proposed for investigating obfuscation for provers. Note that to use zero-knowledge proofs of knowledge, a general paradigm is that a trusted party first samples a random hard public input (i.e. instance) and a witness for each prover, which then keeps its witness secret, and later a prover proves to a verifier the knowledge of the witness. Since the public input and witness are chosen randomly, it is natural to require that the prover's program with the *random* public input and witness hardwired satisfies ACVBB only. For this consideration it is reasonable to adopt this definition when investigating the obfuscation for provers.

The second definition adopts the worst-case VBB requirement introduced by [7,8,29], which is proposed for investigating obfuscation for verifiers. In this case, our goal is to hide all provers' names and public inputs in the verifier's program. Although the public inputs are usually randomly chosen, provers' names are not random. Thus the worst-case VBB requirement is more suitable when investigating obfuscation for verifiers.

Based on the ACVBB definition, we achieve some impossibilities of obfuscating honest provers of zero-knowledge and witness-indistinguishable (WI) proofs of knowledge. These results state that the prover with a public input and its witness hardwired of any zero-knowledge (or witness-indistinguishable) proof of knowledge of efficient prover's strategy is unobfuscatable if computing a witness (or a second witness) for the public input is hard. Moreover, we extend these results to t-composition setting and achieve similar results. These results imply that if an adversary obtains the prover's code (e.g. stealing a smartcard) he can indeed learn some knowledge beyond its functionality from it no matter what measures the card designer may use for resisting reverse-engineering. Note that this result doesn't contradict the general obfuscation in [20] since [20] assumed the tamper-proof hardware.

Based on the worst-case VBB definition, we provide a possibility of obfuscating the honest verifier (with any public input hardwired) of Blum's 3-round zero-knowledge proof for Hamilton Cycle [5]. We construct an obfuscation for

the verifier, which implies that even if an adversary obtains the verifier's code, he cannot learn any knowledge, e.g. a prover's name, from it. Thus we realize the anonymity of the prover's access to the verifier and thus solve the issue of privacy protection. We further present some second-level considerations, e.g. achieving negligible soundness error, extending the possibility to multi-prover setting etc. Lastly, we point out that the approach for obfuscating the verifier is not suitable for obfuscating verifiers of other known zero-knowledge protocols e.g. Goldreich et al.'s proof for Graph 3-colorability [18]. Note that this result doesn't contradict the impossibility in [23], which showed a deterministic cheating verifier (without any public input hardwired) of some non-black-box protocol is unobfuscatable, while our possibility says that the honest verifier of Blum's (black-box) proof with any public input hardwired is obfuscatable and the obfuscation is still a PPT algorithm.

1.2 Organizations

The rest of the paper is arranged as follows. In Section 2, we present the preliminaries for this paper. In Section 3 we present the two definitions of obfuscation for interactive probabilistic programs. In Section 4 we present the impossibilities of obfuscating honest provers of zero-knowledge and witness-indistinguishable protocols. In Section 5 we present an obfuscation for the honest verifier of Blum's proof for Hamilton Cycle.

2 Preliminaries

For space limitations we assume familiarity with the notions of negligible functions (we use $\mathsf{neg}(n)$ to denote an unspecified negligible function), computational indistinguishability, interactive proofs [21] and arguments [6], and show the definitions of the following notions.

2.1 Point Functions and Their Obfuscation

A point function, $I_x : \{0,1\}^{|x|} \to \{0,1\}$, outputs 1 if and only if its input matches x, i.e., $I_x(y) = 1$ iff $y = x$, and outputs 0 otherwise. Let $\mathcal{I}_n$ denote the family of all I_x where $|x| = n$. [7,8,29] showed some constructions of obfuscation for $\mathcal{I}_n$ based on some non-standard complexity assumptions w.r.t. the following definition.

Definition 1. *Let $\mathcal{F}$ be any family of functions. A PPT, O, is called an obfuscator of $\mathcal{F}$, if:*

Approximate Functionality. *For any $f \in \mathcal{F}$: $\Pr[\exists x, O(F)(x) \neq F(x)]$ is negligible, where the probability is taken over all choices of O's coins.*

Polynomial Slowdown. *There is a polynomial p such that, for any $f \in \mathcal{F}$, $O(F)$ runs in time at most $p(T_f)$, where T_f is the worst-case running time of f.*
VBB. *For any non-uniform PPT A and any polynomial p, there exists a non-uniform PPT S such that for any $f \in \mathcal{F}$ and sufficiently large n: $|\Pr[A(O(f)) = 1] - \Pr[S^f(1^n) = 1]| \leq 1/p(n)$.*

2.2 Zero-Knowledge

Zero-knowledge was introduced by [21]. Usually, it is defined as that the view of any verifier in a real interaction can be simulated by the simulator. Here we adopt an equivalent definition for this paper which says the verifier's output in a real interaction can be simulated by the simulator, as follows.

Definition 2. *Let $L = L(R)$ be some language and let (P, V) be an interactive proof or argument for L. We say (P, V) is zero-knowledge if there exists a probabilistic polynomial-time algorithm, called simulator, such that for every polynomial-sized circuit V^* and every $(x, w) \in R$, the following two probability distributions are computationally indistinguishable:*
1. *V^*'s output in the real execution with $P(x, w)$.*
2. *The simulator's output on input (x, V^*).*

Concurrent zero-knowledge [14] is a generalization of zero-knowledge, which says that for any verifier taking part in multiple sessions there exists a simulator which output is indistinguishable from the verifier's output in the multiple sessions. The detailed definition can be found in [14].

2.3 Witness Indistinguishability

Witness indistinguishability is a weaker property than zero-knowledge, introduced by [16]. In a witness indistinguishable proof system if both w_1 and w_2 are witnesses that $x \in L$, then it is infeasible for the verifier to distinguish whether the prover used w_1 or w_2 as auxiliary input. We adopt the following definition.

Definition 3. *Let $L = L(R)$ be some language and (P, V) be a proof or argument system for L. We say that (P, V) is witness indistinguishable if for any polynomial-sized circuit V^*, any x, w_1, w_2 where $(x, w_1) \in R$ and $(x, w_2) \in R$, it holds that V^*'s output in the interaction with $P(x, w_1)$ is computationally indistinguishable from V^*'s output in the interaction with $P(x, w_2)$.*

It is shown in [16] that witness indistinguishability is preserved under concurrent composition.

2.4 Proofs of Knowledge

We adopt the following definition of proofs of knowledge shown in [15,3].

Definition 4. *Let $L = L(R)$ and let (P, V) be a proof (argument) system for L. We say that (P, V) is a proof (argument) of knowledge for L if there exists a probabilistic polynomial-time algorithm E (called the knowledge extractor) such that for every polynomial-sized prover P^* and for every $x \in \{0,1\}^n$, if we let p' denote the probability that V accepts x when interacting with P^*, then* $\Pr[E(P^*, x) \in R(x)] \geq p'(n) - \mathsf{neg}(n)$.

3 Definitions of Obfuscation for Interactive Probabilistic Programs

In this section we show the two definitions of obfuscation for interactive probabilistic programs. Basically, these definitions follow the paradigm in the known definitions by requiring the functionality and the (AC)VBB properties. In Section 3.1 we present the considerations in formalizing these two properties and in Section 3.2 we give the definitions.

3.1 Considerations

We first show our consideration in formalizing the functionality property. Recall that [23] presented some definitions for interactive deterministic programs, which require that an obfuscation of a program is of same functionality with it. First notice that an interactive program, on a message sequence, may generate a communicated message and a local output (if there is no such message or output, let it be empty). Thus the functionality property in [23] says that on input a same message sequence, the program and its obfuscation generate same communicated messages and same local outputs (if the message sequence is illegal, the communicated message and local output can be $\perp$).

For interactive probabilistic programs, we could also adopt a probabilistic variant of this functionality property. That is, we could require that on input a fixed same message sequence, a program and its obfuscation should generate identically distributed communicated messages and identically distributed local outputs. Actually, this is indeed the main idea in formalizing the functionality property in this work. But we relax it in the two definitions in different ways.

First, we can relax the exact identical distribution requirement to a computational indistinguishability requirement. Informally, let $f \in \mathcal{F}$ denote an interactive probabilistic program and assume O is a possible obfuscator for $\mathcal{F}$. Our relaxed requirement says that for any PPT machine M, in the executions of $M^{<f>}$ and $M^{<O(f)>}$, the outputs of M in the two executions are computationally indistinguishable and f's and $O(f)$'s local outputs (which M cannot access) are also computationally indistinguishable. We will adopt this relaxation in Definition 6 for investigating obfuscation for provers, which can make our impossibilities even stronger.

Here $< f >$ (resp. $< O(f) >$) denotes the (probabilistic) functionality of f (resp. $O(f)$) and any machine cannot access its internal state (including coins). The notation of $M^{\langle f \rangle}$ (resp. $M^{<O(f)>}$) denotes a joint computation between M and f (resp. $O(f)$). We introduce this new notation in order to distinguish it from some known notations in literature, e.g. A^I where A is a PPT machine and I is a point function, or S'^V where S' is a simulator for proving zero-knowledge and V is a verifier, or E^P where E is a knowledge extractor and P is a valid prover. The formal definition of this joint computation is shown in Definition 5.

Second, we can also adopt a similar relaxation in e.g. [8] by requiring that $O(f)$ and f satisfy the identical distribution requirement except for a negligible

fraction of O's coins, i.e. the approximate functionality (for interactive probabilistic programs). Actually, we will adopt this relaxation in Definition 7 for investigating obfuscation for verifiers.

We now turn to our consideration in formalizing the (AC)VBB property. Informally, we require that for any adversary A, there is a PPT simulator S satisfying that $S^{<f>}$ can output a fake program such that A cannot tell it from $O(f)$ (w.r.t. ACVBB in Definition 6) or what A can learn from $O(f)$ is also computable by $S^{<f>}$ (w.r.t. VBB in Definition 7). Notice that here we also employ the new notation of $S^{<f>}$. Thus all that is left is to characterize the joint computation of $S^{<f>}, M^{<f>}, M^{<O(f)>}$, as follows.

We illustrate this w.r.t. $S^{<f>}$ and then $M^{<f>}, M^{<O(f)>}$ can be defined similarly. This is actually the issue how to characterize the black-box access to f. Notice that if f were deterministic, f's response to a same S's query, which can be either a message or a message sequence, would be always same. But the situation becomes complex if f is interactive and probabilistic. We think that a reasonable characterization that S can access oracle f which conforms to the intuition of black-box access to f is as follows. First, S can play a consecutive interaction with f, as a real interaction. In each step, f on receiving a new message, computes the response and the local output using fresh coins. S can either play with f throughout the whole interaction, or cancel the interaction by sending an invalid message to f which then aborts the current interaction. Notice that f is a stateful machine in a consecutive interaction. By stateful, we mean that f can preserve its current internal state for later computation. When this interaction is finished or aborted, f cleans its internal state and is reset to the initial state. Second, after one interaction, S may invoke a new interaction with f, which is also reset to the initial state and computes the message with the fresh coins for its first step.

(We remark that the access of S to f in $S^{<f>}$ should be distinguished from the access of a black-box simulator for proving zero-knowledge to a verifier. Notice that this simulator is allowed to obtain the verifier's code and the term of "black-box" only means that the simulator doesn't reverse-engineering the verifier, but it can access the verifier's internal state and accordingly the simulator can rewind the verifier etc. It can be seen that such kind of access requires the knowledge of the verifier's program instead of the knowledge of the verifier's input-output behavior only. In $S^{<f>}$, S cannot behave like this.)

Thus we can see in each S's access to f, S should specify the way it tries to access f. For simplicity of characterizing $S^{<f>}$, we assume that in each access S should additionally send a bit b indicating if this access is for a consecutive step of the current interaction or for the first step of a new interaction. More concretely, each S's access to f consists of a message m and a bit b. If m is for a consecutive step of the current interaction, S resets $b = 0$ and requests that f responds to m based on its current internal state. If m is for the first step of a new interaction (m can be arbitrary if it is f that sends the first message), S sets $b = 1$ and requests that f responds to m where f is reset to the initial state. Formally, our characterization of $S^{<f>}$ is shown as follows.

Definition 5. *(characterization of $S^{<f>}$) For any two interactive probabilistic machines S, f, $S^{<f>}$ refers to the following joint computation between S and f: S starts a local computation (with an input) and then during the computation it needs to access f for responses. Each S's access to f consists of a message m and a bit $b \in \{0,1\}$. On receiving (m,b), f responds as follows:*

1. *if $b = 0$: f continues the interaction with it current internal state, computes the message corresponding to m and the local output where it tosses fresh coins if needed, and responds with the message. If this interaction is finished after this step, f is reset to the internal state.*
2. *if $b = 1$: f is reset to the internal state and computes the message corresponding to m and the local output where it tosses fresh coins if needed, and responds with the message. If this interaction is finished after this step, f is reset to the internal state.*
3. *if $b \notin \{0,1\}$: f responds with $\perp$ and is reset to the internal state.*

When receiving f's response, S proceeds to its local computation until the next access to f.

3.2 Definitions

Now we present our definitions. Since our objects in this work are provers and verifiers which are programs with some inputs hardwired, it is convenient to use circuits to represent such programs. So we present the definitions w.r.t. circuits explicitly, in which the meaning of $M^{<f>}, M^{<O(f)>}, S^{<f>}$ conforms to Definition 5. First, we present the definition w.r.t. ACVBB which basically follows the known definitions in [25,26], as Definition 6 shows.

Definition 6. *Let $\mathcal{F}_n$ be a family of interactive probabilistic polynomial-size circuits. Let $\mathcal{O}$ be a PPT algorithm which maps (description of) each $f \in \mathcal{F}_n$ to a circuit $O(f)$. We say that $\mathcal{O}$ is an obfuscator for $\mathcal{F}_n$ w.r.t. distribution β_n iff the following holds:*

Computational Functionality: *for random $f \in \mathcal{F}_n$ chosen from distribution β_n, for a random sample of $O(f)$, for any PPT machine M, in the two executions of $M^{<f>}$ and $M^{<O(f)>}$, M's outputs are computationally indistinguishable and f's and $O(f)$'s local outputs are also computationally indistinguishable.*

ACVBB: *there is a PPT simulator S such that for each non-uniform PPT D and random $f \in \mathcal{F}_n$ chosen from distribution β_n, $|\Pr[D(O(f)) = 1] - \Pr[D(S^{<f>}(1^n)) = 1]| = \mathsf{neg}(n)$, where the probabilities are taken over all choices of f and O's, S's and (the oracle) f's coins.*

Definition 6 is proposed for establishing our impossibilities. We could further require that D is given the access to oracle f, as [25,26] adopted. This weaker handling makes the impossibilities stronger. On the other hand, to show the possibility of obfuscating verifiers, we follows [7,8,29] to adopt the following definition, which requires the VBB property holds for each f rather than for a random f.

Definition 7. *Let $\mathcal{F}_n, O$ be as Definition 6 shows. We say that $\mathcal{O}$ is an obfuscator for $\mathcal{F}_n$ iff the following holds:*

Approximate Functionality: *for each $f \in \mathcal{F}_n$, $O(f)$ and f are of equivalent functionality except for a negligible fraction of O's coins.*

VBB: *for each non-uniform PPT A and each polynomial p, there is an non-uniform PPT S such that for* each *$f \in \mathcal{F}_n$, $|\Pr[A(O(f)) = 1] - \Pr[S^{<f>}(1^n) = 1]| < 1/p(n)$, where the probabilities are taken over all choices of A's, S's and O's coins.*

As shown in [28], the above two definitions are incomparable. To use zero-knowledge proofs of knowledge to realize identification, a general paradigm is to first sample a random hard instance and a witness for each prover, which then keeps its witness secret. Thus when investigating the obfuscation for provers, it is natural to adopt Definition 6. On the other hand, when investigating the obfuscation for verifiers, our goal is to hide the instance and the client's name where the latter is not random. Thus Definition 7 is more suitable in this scenario.

4 Impossibilities of Obfuscating Provers

In this section we address the question whether or not the honest prover's algorithm incorporated with a public input and a witness is obfuscatable. The results in this section basically say that for a zero-knowledge (or WI) proof or argument of knowledge which uses an efficient prover's strategy, the honest prover is unobfuscatable w.r.t. Definition 6 under some hardness requirements. In Section 4.1 we present the negative results. In Section 4.2 we extend these negative results to t-composition setting.

4.1 Impossibilities for Zero Knowledge and Witness Indistinguishability

We now present the impossibilities for obfuscating provers of zero-knowledge and WI proofs of knowledge. We first present the impossibility for zero-knowledge. Assume (P, V) is a zero-knowledge proof or argument of knowledge for a relation R and P admits an efficient strategy if given $(x, w) \in R$. Moreover, we require that there is a hard-instance sampling algorithm that can randomly select a hard instance x and a witness w such that given x it is hard to compute a witness for x. More precisely, we have the following definition.

Definition 8. *(hard-instance sampler) A PPT machine Sam_R is called a hard-instance sampler for relation R, if for any (non-uniform) PPT A, $\Pr[\mathsf{Sam}_R(1^n) \rightarrow (x, w), (x, w) \in R_n : A(x) \rightarrow w' \text{ s.t. } (x, w') \in R_n] = \mathsf{neg}(n)$, where R_n denotes the subset of R in which all instances are of length n and $\mathsf{Sam}_R(1^n)$'s output is always in R_n, and the probability is taken over all choices of Sam_R's and A's coins.*

A probability distribution β_{R_n} over R_n is called a distribution induced by Sam_R, if letting ξ denote the random variable conforming to β_{R_n}, for each $(x,w) \in R_n$, $\Pr[\xi = (x,w)]$ is defined as $\Pr[\mathsf{Sam}_R(1^n) = (x,w)]$, where the latter is taken over all choices of Sam_R's coins.

For instance, for an arbitrary one-way function f let R_f denote $\{(f(x),x)\}$ for all x. Then a hard-instance sampler for $\{(f(x),x)\}$ can be the one simply choosing $x_0 \in \{0,1\}^n$ uniformly and outputting $(f(x_0),x_0)$. Usually, it is unnecessary to calculate β_{R_n} explicitly. For instance, in the case of R_f, we don't need to calculate the induced distribution of $(f(x_0),x_0)$ generated by the sampler explicitly. Actually, we only need to be ensured that given $f(x_0)$ it is hard to compute a x' satisfying $f(x') = f(x_0)$.

For $(x,w) \in R_n$, let $P(x,w)$ denote P's program with (x,w) hardwired. Let $P(X,W)$ denote the family of all $P(x,w)$ for all $(x,w) \in R_n$. Our first impossibility says that $P(X,W)$ cannot be obfuscated, as the following claim states.

Claim 1. *Assume (P,V) is a zero-knowledge proof or argument of knowledge for relation R. Assume Sam_R is a hard-instance sampler for R and β_{R_n} is the probability distribution over R_n induced by Sam_R. Then $P(X,W)$ is unobfuscatable w.r.t. β_{R_n} under Definition 6.*

Proof. Suppose the claim is not true. This means that there exists an obfuscator O for $P(X,W)$. For $(x,w) \leftarrow_R \mathsf{Sam}_R(1^n)$ (i.e. choose (x,w) conforming to β_{R_n}), let Q denote $O(P(x,w))$. Then there exists a PPT simulator S satisfying that for any polynomial-sized distinguisher D, $|\Pr[D(Q)=1] - \Pr[D(S^{<P(x,w)>}(1^n))=1]| = \mathsf{neg}(n)$. (Here we assume S can obtain x by querying $P(x,w)$ with a specific command. We do this for the consideration that we are trying to provide negative results. If S given x cannot satisfy ACVBB, then S without x by no means satisfies ACVBB.) Since Q is an obfuscation for $P(x,w)$, Q can convince V that $x \in L(R)$ with overwhelming probability for every true x or V rejects x with overwhelming probability for every false x. Since $|\Pr[D(Q)=1] - \Pr[D(S^{<P(x,w)>}(1^n))=1]| = \mathsf{neg}(n)$, we have that S's output is also a valid prover's program which behaves indistinguishably from Q.

Now let us focus on $S^{<P(x,w)>}(1^n)$. First, according to Definition 5, we have that S can perform arbitrary interaction (including some half-baked executions) with $P(x,w)$. Since (P,V) is zero-knowledge, the interaction can be simulated by the simulator for proving zero-knowledge of (P,V). Let S' denote the simulator for proving zero-knowledge of (P,V). Then for S (now viewed as a verifier) and x, S' on input S's code and x can simulate S's output with the interaction with $P(x,w)$ (S' can send x to S). That is, S''s output is indistinguishable from S's output. Let Q' denote S''s output.

Thus, we can construct a PPT algorithm, denoted $D(S')$, which can compute a witness for $x \in L(R)$ with overwhelming probability. $D(S')$ works as follows. On input x generated by $\mathsf{Sam}_R(1^n)$, $D(S')$ runs $S'(x)$ (w.l.o.g. assume S' having S's code hardwired) to generate Q'. Then since (P,V) is a proof or argument of knowledge, D runs the knowledge extractor E of (P,V) on input Q' to output a witness. Since Q' is indistinguishable from S's output in the execution

of $S^{<P(x,w)>}(1^n)$, which is indistinguishable from Q, Q' is also a valid prover program which can convince V $x \in L(R)$ with overwhelming probability. Thus the probability that E on input Q' succeeds in generating a witness for x is overwhelming. This contradicts the assumption on Sam_R. This absurdity is caused by the hypothesis that $P(X, W)$ is obfuscatable. So the claim follows.

□

Now let us consider the case that (P, V) is not zero-knowledge but witness-indistinguishable. Is $P(X, W)$ still unobfuscatable? Actually, we cannot show this fact. But if each x has at least two witnesses and assume that for any PPT A, given $(x, w) \leftarrow_{\mathrm{R}} \mathsf{Sam}_R(1^n)$, $\Pr[\mathsf{Sam}_R(1^n) \to (x, w), (x, w) \in R_n : A(x, w) \to w' \text{ s.t. } (x, w') \in R_n, w' \neq w] = \mathsf{neg}(n)$, where the probability is taken over all choices of Sam_R's and A's coins, we have that $P(x, w)$ is indeed unobfuscatable. We say that Sam_R is now second-witness resistant. For instance, let f, g denote two one-way permutations, $R_{f,g}$ denote $\{((y_1, y_2), x)) : f(x) = y_1 \text{ or } g(x) = y_2.\}$. A second-witness resistant sampler for $R_{f,g}$ can simply choose $x_1, x_2 \in_{\mathrm{R}} \{0,1\}^n$ and output $((f(x_1), g(x_2)), x_1))$. Any A on input $((f(x_1), g(x_2)), x_1))$ cannot output x_2 with non-negligible probability. Thus the impossibility for WI is shown as follows.

Claim 2. *Assume (P, V) is a WI proof or argument of knowledge for relation R and each x in $L(R)$ has at least two witnesses. Assume Sam_R is a second-witness resistant sampler for R and β_{R_n} is the probability distribution over R_n induced by Sam_R. Then $P(X, W)$ is unobfuscatable w.r.t. β_{R_n} under Definition 6.*

Proof. Suppose that $P(X, W)$ is obfuscatable. Then there exists an obfuscator O for $P(X, W)$. For $(x, w) \leftarrow_R \mathsf{Sam}_R(1^n)$ let Q denote $O(P(x, w))$. Then there exists a PPT simulator S such that for any polynomial-sized distinguisher D, $|\Pr[D(Q) = 1] - \Pr[D(S^{<P(x,w)>}(1^n)) = 1]| = \mathsf{neg}(n)$.

Let Q' denote $S^{<P(x,w)>}(1^n)$'s output. Since Q is an obfuscation for $P(x, w)$, the extractor E of (P, V) on input Q's description can output a witness for $x \in L(R)$ with non-negligible probability. By the indistinguishability of Q and Q', we have that E on input Q''s description can also output a witness for $x \in L(R)$ with non-negligible probability.

Then we can show that there exists a verifier which can tell which witness P is using. The verifier runs as follows. It adopts S's strategy to interact with P and finally output a program, denoted Q'. Then it adopts E's strategy on input Q' to extract a witness and output it. Since Sam_R is second-witness resistant, we claim that this witness is w. Otherwise, we can construct an A which can violate the second-witness resistance. That is, on input $(x, w) \leftarrow_{\mathrm{R}} \mathsf{Sam}_R(1^n)$, A constructs $P(x, w)$ and adopts S's strategy with $P(x, w)$ to generate Q' and then runs E with input Q' to output a witness. If this witness is not x, the second-witness resistance of Sam_R is violated.

Thus it can be seen that the verifier's outputs in different interactions where P uses different witnesses for $x \in L(R)$ are obviously distinguishable. However, notice that any sequential repetitions of (P, V) are still witness-indistinguishable. It is a contradiction. Thus the claim follows.

□

Note that the result in this subsection requires the proof of knowledge property for (P, V). Then what about those proof systems without the proof of knowledge property? We leave this an interesting question in future works.

4.2 Extending the Impossibilities to t-Composition Setting

Definition 6 defines that S given access to f can output a fake program which is indistinguishable from the true obfuscated program. What about the case that the distinguisher obtains multiple such programs? To address this question, [8,9,4] introduced the notion of t-composeability of obfuscated programs, which requires that ACVBB or VBB holds in t-composition setting. Basically, an obfuscation w.r.t. t-composition setting requires that even if the distinguisher can obtain t obfuscated programs, he cannot distinguish them from t fake programs generated by S given access to the t programs. We now follow [8,9,4] to extend Definition 6 to t-composition setting as follows.

Definition 9. *Let $\mathcal{F}_n, \mathcal{O}, V$ be those in Definition 6. We say that $\mathcal{O}$ is an obfuscator for $\mathcal{F}_n$ w.r.t. distribution β_n in t-composition setting for any polynomial $t(n)$ iff the same conditions appeared in Definition 6 hold except that the ACVBB property is now modified as follows:*

ACVBB: *there is a uniform PPT oracle simulator S such that for each non-uniform PPT D, and randomly chosen $f_1, \cdots, f_t$ conforming to distribution β_n, $Q_i = \mathcal{O}(f_i)$ for all i, $|\Pr[D(Q_1, \cdots, Q_t) = 1] - \Pr[D(S^{<f_1>,\cdots,<f_t>}(1^n, t)) = 1]| = \mathsf{neg}(n)$, where the probabilities are taken over all choices of S's coins and Q_i for all i, and the access of S to each f_i conforms to Definition 5.*

The result in this subsection says that if (P, V) is concurrent zero-knowledge or witness-indistinguishable, then the result in the previous subsection still holds.

Claim 3. *Assume (P, V) is a concurrent zero-knowledge proof or argument of knowledge for R. Assume Sam_R is a hard-instance sampler for R and β_{R_n} is the probability distribution over R_n induced by Sam_R. Then $P(X, W)$ is unobfuscatable w.r.t. β_n under Definition 9.*

Proof (Proof Sketch:). The proof is similar to that of Claim 1. The route for proving this claim is that if there exists a simulator S which can output t indistinguishable programs, then it is easy to compute a witness for $x_i \in L(R)$. The key point in the proof is that in the computation of $S^{<P(x_1,w_1)>,\cdots,<P(x_t,w_t)>}(1^n, t)$ S can interact with all $P(x_i, w_i)$ for all i and it can adopt any scheduling's strategy for all sessions, so the computation of $S^{<P(x_1,w_1)>,\cdots,<P(x_t,w_t)>}(1^n, t)$ is actually a concurrent interaction between S and these $P(x_i, w_i)$. Then using the similar proof and the simulator for proving concurrent zero-knowledge, we can also construct a PPT algorithm which computes a witness for x_i. It is a contradiction. The claim follows. □

Claim 4. *Given the same conditions in Claim 2, $P(X, W)$ is unobfuscatable w.r.t. β_{R_n} under Definition 9.*

Proof (Proof Sketch:). The proof is similar to that of Claim 2. The route for proving this claim is that if there exists a simulator S which can output t indistinguishable programs, then there exists a verifier interacting with $P(x_i, w_i)$ for all i which can tell the witnesses that t copies of P are using. Notice that witness indistinguishability can be preserved in concurrent setting. The construction of the verifier can be referred to the proof of Claim 2. The claim follows. □

5 Possibilities of Obfuscating Verifiers

In this section we turn to the possibility of obfuscating verifiers of zero-knowledge protocols. The result in this section is that we provide an obfuscation for the honest verifier of Blum's proof for HC w.r.t. Definition 7. In Section 5.1 we present the privacy issue and the motivation of obfuscating verifiers aroused by this issue in detail. In Section 5.2 we present an obfuscation for the honest verifier of Blum's proof for HC which can hide the privacy information (i.e. public inputs).

5.1 Motivation for Obfuscating Verifiers

To run a zero-knowledge protocol, the prover and verifier first need to know the public input. Then the prover proves to the verifier that the public input is true or he knows a witness for it. Usually, the public input is not a secret and both the parties as well as adversaries can obtain it. The security requirements of zero-knowledge protocols don't concern the secrecy of public inputs either. Thus, we ask *if public inputs to zero-knowledge protocols are really not secrets and thus we don't need to hide or protect them at all.*

Consider the scenario of identification. As we know, zero-knowledge proofs of knowledge are a secure way to implement identification. Briefly, an identification scheme consists of a trusted party and multi-clients (i.e. provers) and multi-servers (i.e. verifiers). The trusted party generates a verifiable public input (a.k.a. statement or instance, e.g. a directed graph or a modular square) and a witness (e.g. a Hamilton cycle or a modular square root) for each client. A client's name and its public input constitute a record and the trusted party maintains all records. Each client stores his witness secretly and when he needs to identify himself to a server, he proves to the server the knowledge of the witness via the identification scheme. It is obvious that the server needs to obtain his public input firstly. Actually, it has two options in obtaining this public input as follows.

One is that the server queries the trusted party online with a client's name for his public input each time the client needs to identify himself. But this online method is quite impractical or even impossible in some applications (consider the case that there are some secret governmental organizations that the trusted party cannot publish their names and public inputs, and the case that the trusted party cannot be available in many offline applications). The other one is that the server stores all clients' names and public inputs locally (incorporated in verifier's program) and when a client invokes an identification procedure by informing the

server his name, the server can find the corresponding public input and proceeds with the procedure.

Although the second method is practical, there is a risk for leaking clients' privacy. That is, if an adversary can control a server absolutely, he can gain all clients' names from the verifier's program and thus the privacy of these clients leaks. First consider the case of medical systems. If an adversary controls some medical server, which clients are usually patients, then he may extract all patients' names in this server and thus these patients' privacy leaks.

Second, consider the case of ID cards. As we see, ID cards are wildly used in our society. Some secret governmental organizations may need to read data from cards for some secret goals. Before their card readers can read data from a card, they need to pass identification procedures. Thus, by adopting the second method, cards store all legal clients' names and their public inputs. However, these governmental organizations may not want their names stored in cards since they would like to access them anonymously without being identified. Thus a adversarial card-holder can of course extract their names from the verifier's program in his own card, which frustrates the anonymity of their accesses.

Thus, clients' names and public inputs should be protected in some applications. In both of the two cases above, even if a server encrypts clients' names and public inputs, adversaries can first extract the encryption key in the server when they control it and then perform decryption to obtain the desired data. This shows to us that it is important to obfuscate verifiers' programs to hide all information, especially, clients' names and public inputs. In the next subsection, we will provide a desired verifier's strategy of Blum's protocol which can solve this privacy issue.

5.2 Obfuscation for Verifiers

In this subsection we provide an obfuscation for the honest verifier of Blum's protocol [5] while keeping the prover's strategy unchanged. Before proceeding to the obfuscation, we first explain why to choose Blum's proof to demonstrate the possibility of obfuscating verifiers. Basically, when the question of obfuscating verifiers is proposed, we would like to first provide a (theoretical) solution and then propose some practical solutions (as the research community usually does for other problems). Currently we can only present such a possibility for the verifier of Blum's proof. Moreover, we remark in the end of this section that the approach in obfuscating the verifier of Blum's proof cannot be directly used for obfuscating verifiers of other protocols.

For each directed graph G, we use M to denote the entries of G's adjacency matrix. For each M, let I_M denote the point function that on input M outputs 1 or outputs 0 otherwise. Let O_{I_M} denote an obfuscation of I_M w.r.t. Definition 1.

We assume familiar with Blum's protocol. For convenience of constructing obfuscation, we require that the verifier of Blum's proof lastly sends its accept/reject decision to the prover. In the following we present the basic version of our modified protocol in detail and after that we present some second-level considerations such as reducing soundness error via parallel repetitions or

Prover's input: a directed graph $G = (V, E)$ with $n = |V|$; C: a directed Hamilton cycle, $C \subset E$, in G.

Verifier's input: O_{I_M}: an obfuscation of point function I_M where M is the entries of the adjacency matrix of G.

Step P1: The prover selects a random permutation π of the vertices V and computes a graph $G' = (V', E')$ where $V' = \pi(V)$ and E' consists of those edges $(\pi(i), \pi(j))$ for each $(i, j) \in E$. Then it commits to the entries of the adjacency matrix of G'. That is, it sends an n-by-n matrix of commitments such that the $(\pi(i), \pi(j))$ entry is a commitment to 1 if $(i, j) \in E$ and is a commitment to 0 otherwise.

Step V2: The verifier uniformly selects $\sigma \in \{0, 1\}$ and sends it to the prover.

Step P3: If $\sigma = 0$, then the prover sends π to the verifier along with the revealing (i.e., pre-images) of all commitments. Otherwise, the prover reveals to the verifier only the commitments to entries $(\pi(i), \pi(j))$, with $(i, j) \in C$.

Step V4: (This step differs from the original construction.) If $\sigma = 0$, then the verifier computes the inverse permutation of π, denoted π^{-1}, and computes $H = \pi^{-1}(G')$. Then it inputs the entries of the adjacency matrix of H to $O(I_M)$. If $O(I_M)$ returns 1, then the verifier accepts or else rejects. If $\sigma = 1$, the verifier simply checks that all revealed values are 1 and that the corresponding entries form a simple n-cycle. (Of course, in both cases, the verifier checks that the revealed values do fit the commitments.) The verifier accepts if and only if the corresponding condition holds. Lastly, the verifier sends its accept/reject decision to the prover.

Protocol 1 . The obfuscated verifier for Blum's proof for HC

sequential repetitions of the basic protocol, extending the protocol to multi-prover setting, adding provers' names in the verifier's program.

Our obfuscated verifier is shown in Protocol 1. It can be first seen that the verifier doesn't receive G as input and instead its input is O_{I_M}. Second, our verifier differs from the original verifier in [5] is in Step V4. Third, ensured by Claim 5, this protocol is also a zero-knowledge proof of knowledge. If it is used for identification, the prover needs to send its name to the verifier before invoking the identification procedure. But now we ignore the step of prover's sending its name and accordingly, we only need to refer to point function I_M. We will later show that when considering that step, we only need to slightly modify the point function by adding prover's name.

Let Ver denote the honest verifier of Blum's protocol, $Ver(G)$ denote Ver having G hardwired. Let Ver' denote the verifier in Protocol 1, $Ver'(O_{I_M})$ denote Ver' having O_{I_M} hardwired (satisfying that O_{I_M}'s description can be read from $Ver'(O_{I_M})$'s code). Then we have the following claim.

Claim 5. *Assume O_{I_M} is an obfuscation of point function I_M w.r.t. Definition 1. Then $Ver'(O_{I_M})$ is an obfuscation of $Ver(G)$ w.r.t. Definition 7.*

Proof. We now show that the two properties in Definition 7 can be satisfied.

Approximate Functionality. The difference between $Ver'(O_{I_M})$ and $Ver(G)$ lies in Step V4. So we only focus on the two verifiers' outputs in Step V4. By [5] and Protocol 1, in the case of $\sigma = 1$, the two verifiers behaves identically. Thus we only need to show that their outputs are also identical in the case of $\sigma = 0$ in the following. Due to [5], in this case $Ver(G)$ checks if G' and G are isomorphic via π. If they are isomorphic, $Ver(G)$ accepts or rejects otherwise. On the other hand, for $Ver'(O_{I_M})$, due to Protocol 1 if G' and G are isomorphic via π, then $H = \pi^{-1}(G')$ is indeed G. So by Definition 1, except for negligible probability (in generating O_{I_M}), O_{I_M} on input the adjacency matrix of H should output 1. Thus $Ver'(O_{I_M})$ accepts. If the two graphs are not isomorphic, both the two verifiers reject, except for negligible probability. Thus, the approximate functionality property is satisfied.

VBB. To establish the VBB property, we need to show that for each A and each polynomial $p(n)$, there exists a simulator S satisfying that $|\Pr[A(Ver'(O_{I_M})) = 1] - \Pr[S^{<Ver(G)>}(1^n) = 1]| < 1/p(n)$ where $n = |V|$. We can see that for A, there exists a PPT A' such that $A(Ver'(O_{I_M}))$'s output is identical to A''s output on input $O(I_M)$. Thus $\Pr[A(Ver'(O_{I_M})) = 1] = \Pr[A'(O_{I_M}) = 1]$.

Since O_{I_M} is an obfuscation of I_M w.r.t. Definition 1, for A' and $2p(n)$, there exists a simulator S' satisfying that $|\Pr[A'(O_{I_M}) = 1] - \Pr[S'^{I_M}(1^n) = 1]| < 1/2p(n)$. We now turn to construct the desired S such that $|\Pr[S'^{I_M}(1^n) = 1] - \Pr[S^{<Ver(G)>}(1^n) = 1]| < 1/2p$. S works as follows. It emulates S''s computation and is responsible for answering all S' queries. During the emulation, for each S''s query S adopts the following strategy. For an arbitrary S''s query M', Let $G_{M'}$ denote the graph corresponding to M' (if M' is an invalid encoding of the adjacency matrix of a graph of n vertexes, S simply responds with 0). S adopts the prover's strategy of Blum's protocol to commit to the entries of the adjacency matrix of a random isomorphism to $G_{M'}$ and sends the commitments to its oracle $Ver(G)$. Then $Ver(G)$ responds with a random bit σ. If $\sigma = 0$, then S sends the permutation and opens all the commitments to $Ver(G)$. If $Ver(G)$ accepts, S knows that $G_{M'}$ is identical to G and then responds to S' with 1. Otherwise, S responds to S' with 0. If $\sigma = 1$, S aborts the current interaction and re-interacts with $Ver(G)$ from the beginning. We allow S to perform the re-interactions at most n times.

It can be seen that S runs in polynomial-time and the probability that σ is always 1 in n interactions is 2^{-n} (notice that Ver is the honest verifier). On the occurring that there exists an interaction in which $\sigma = 0$, S can respond to S with the correct answer. Thus $|\Pr[S'^{I_M}(1^n) = 1] - \Pr[S^{<Ver(G)>}(1^n) = 1]| = 2^{-n} < 1/2p$. Thus, $|\Pr[A'(O_{I_M}) = 1] - \Pr[S^{<Ver(G)>}(1^n) = 1]| < 1/p$. Namely, $|\Pr[A(Ver'(O_{I_M})) = 1] - \Pr[S^{<Ver(G)>}(1^n) = 1]| < 1/p$. Thus the VBB property is satisfied. Taking the above two conclusions, we have the claim follows.

□

Claim 5 ensures that even if an adversary obtains $Ver'(O_{I_M})$'s description totally, he cannot learn anything from this program beyond its functionality, e.g. the prover's public input. We remark that to protect some provers' privacy in

practice, the trusted party should keep these provers' names and public inputs secret, or else an adversary may obtain these records easily and input each of them to O_{I_M} one by one to locate the prover. Now we present several second-level considerations for Protocol 1 as follows.

Achieving Negligible Soundness Error. Protocol 1 admits almost 1/2 soundness error. We now may adopt $\omega(\log n)$ repetitions of Protocol 1 or $\omega(1)$ repetitions of the $\log n$ parallel composition of Protocol 1 to reduce the soundness error to negligible.

Extension to Multi-prover Setting. In practice there are multi-provers who need to prove to Ver' via Protocol 1. In this case, Ver' needs to store the public inputs of these provers securely such that an adversary cannot extract these inputs even he can obtain the verifier's program totally. Based on Protocol 1, we can easily extend the obfuscation of the verifier to the multi-prover setting.

Assume there exist m provers with public inputs $G_1, \cdots, G_m$ which are pairwise different. Let $M_1, \ldots, M_m$ denote the the entries of the adjacency matrix of $G_1 \cdots, G_m$ respectively. The solution for hiding these public inputs in Ver' is to replace O_{I_M} in Protocol 1 by $O_{I_{M_1,\cdots,M_m}}$, which denotes an obfuscation for the multi-point function $I_{M_1,\cdots,M_m}$ shown in [8]. In Step V4 of each session, Ver' sends H to $O_{I_{M_1,\cdots,M_m}}$ and if $O_{I_{M_1,\cdots,M_m}}$ returns 1 then it accepts or else rejects in the case of $\sigma = 0$. It can be seen that Ver' with $O_{I_{M_1,\cdots,M_m}}$ hardwired is also an obfuscation of $Ver(G_1, \cdots, G_m)$.

Adding Provers' Names in the Point Function. As mentioned previously, a prover may send its name before invoking the identification procedure. We consider the case that there is only a prover for simplicity. Assume that the prover's name is $\alpha \in \{0,1\}^*$ and its public input is G. We construct an obfuscated point program $O_{I_{\alpha \circ M}}$, which now becomes Ver''s input. In Step V4, in the case of $\sigma = 0$, Ver' computes H and sends $\alpha \circ H$ to $O_{I_{\alpha \circ M}}$, where $\circ$ denotes concatenation. Notice that Ver' receives α from the prover before running the identification procedure. Using the similar analysis, we have the result in this subsection still holds.

Remark 1. It can be seen that our approach for obfuscating the verifier is to let the verifier receive an obfuscated whole information of G as input and then compare the information in Step P3 to the whole information. While, for some other constructions e.g. Goldreich et al.'s protocol [18], their verifiers don't behave in such way. Thus our approach doesn't suit for obfuscating these verifiers.

6 Conclusions

In this paper we address the question of obfuscating provers and verifiers of zero-knowledge proofs of knowledge, motivated by theoretical and practical interests. First, we achieve some impossibilities of obfuscating provers w.r.t. the ACVBB definition, i.e., provers are unobfuscatable. This negative result means that if someone obtains a prover's program, he can indeed learn some knowledge.

Namely we can say the prover's program contains knowledge beyond its functionality. Second, we present an obfuscation for the verifier of Blum's protocol w.r.t. the VBB definition. This positive result means that for anyone who possesses the verifier's program, he cannot learn any knowledge beyond its functionality, which solves the privacy issue of hiding provers' names and public inputs.

Acknowledgments. The authors show their deep thanks to the reviewers of CANS 2011 for their detailed and useful comments, with the help of which the presentation of this work is significantly improved. This work is supported by the National Natural Science Foundation of China (61100209) and Shanghai Postdoctoral Scientific Program (11R21414500) and Major Program of Shanghai Science and Technology Commission (10DZ1500200).

References

1. Barak, B.: How to go beyond the black-box simulation barrier. In: FOCS, pp. 106–115 (2001)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of Obfuscating Programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
3. Bellare, M., Goldreich, O.: On Defining Proofs of Knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993)
4. Bitansky, N., Canetti, R.: On Strong Simulation and Composable Point Obfuscation. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 520–537. Springer, Heidelberg (2010)
5. Blum, M.: How to prove a theorem so no one else can claim it. In: The International Congress of Mathematicians, pp. 1441–1451 (1986)
6. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
7. Canetti, R.: Towards Realizing Random Oracles: Hash Functions that Hide all Partial Information. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 455–469. Springer, Heidelberg (1997)
8. Canetti, R., Dakdouk, R.R.: Obfuscating Point Functions with Multibit Output. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 489–508. Springer, Heidelberg (2008)
9. Canetti, R., Kalai, Y.T., Varia, M., Wichs, D.: On Symmetric Encryption and Point Obfuscation. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 52–71. Springer, Heidelberg (2010)
10. Canetti, R., Rothblum, G.N., Varia, M.: Obfuscation of Hyperplane Membership. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 72–89. Springer, Heidelberg (2010)
11. Canetti, R., Varia, M.: Non-malleable Obfuscation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 73–90. Springer, Heidelberg (2009)
12. Crescenzo, G.D.: You Can Prove so Many Things in Zero-Knowledge. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 10–27. Springer, Heidelberg (2005)
13. Ding, N., Gu, D.: A Note on Obfuscation for Cryptographic Functionalities of Secret-Operation then Public-Encryption. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 377–389. Springer, Heidelberg (2011)

14. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC, pp. 409–418 (1998)
15. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. J. Cryptology 1(2), 77–94 (1988)
16. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC, pp. 416–426. ACM (1990)
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178. ACM (2009)
18. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: FOCS, pp. 174–187. IEEE (1986)
19. Goldwasser, S., Kalai, Y.T.: On the impossibility of obfuscation with auxiliary input. In: FOCS, pp. 553–562. IEEE Computer Society (2005)
20. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-Time Programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
21. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC, pp. 291–304. ACM (1985)
22. Goldwasser, S., Rothblum, G.N.: On Best-Possible Obfuscation. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 194–213. Springer, Heidelberg (2007)
23. Hada, S.: Zero-Knowledge and Code Obfuscation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 443–457. Springer, Heidelberg (2000)
24. Hada, S.: Secure Obfuscation for Encrypted Signatures. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 92–112. Springer, Heidelberg (2010)
25. Hofheinz, D., Malone-Lee, J., Stam, M.: Obfuscation for cryptographic purposes. J. Cryptology 23(1), 121–168 (2010)
26. Hohenberger, S., Rothblum, G.N., Shelat, A., Vaikuntanathan, V.: Securely Obfuscating Re-Encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 233–252. Springer, Heidelberg (2007)
27. Lynn, B., Prabhakaran, M., Sahai, A.: Positive Results and Techniques for Obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004)
28. Varia, M.: Studies in Program Obfuscation. Ph.D. thesis, Massachusetts Institute of Technology (2010)
29. Wee, H.: On obfuscating point functions. In: Gabow, H.N., Fagin, R. (eds.) STOC, pp. 523–532. ACM (2005)