2012 IEEE 53rd Annual Symposium on Foundations of Computer Science

From the Impossibility of Obfuscation to a New Non-Black-Box Simulation Technique
(Extended Abstract)

Nir Bitansky
Tel Aviv University
nirbitan@tau.ac.il

Omer Paneth
Boston University
omer@bu.edu

***Abstract*—The introduction of a non-black-box simulation technique by Barak (FOCS 2001) has been a major landmark in cryptography, breaking the previous barriers of black-box impossibility. Barak's techniques were subsequently extended and have given rise to various powerful applications.**

We present the first non-black-box simulation technique that does not rely on Barak's technique (or on non-standard assumptions). Our technique is based on essentially different tools: *it does not invoke universal arguments*, nor does it rely on collision-resistant hashing. Instead, the main ingredient we use is the impossibility of general program obfuscation (Barak et al., CRYPTO 2001).

Using our technique, we construct a new resettably-sound zero-knowledge (rsZK) protocol. rsZK protocols remain sound even against cheating provers that can repeatedly reset the verifier to its initial state and random tape. Indeed, for such protocols black-box simulation is impossible. Our rsZK protocol is the first to be based solely on semi-honest oblivious transfer and does not rely on collision-resistant hashing; in addition, our protocol does not use PCP machinery.

In the converse direction, we show a generic transformation from any rsZK protocol to a family of functions that cannot be obfuscated.

I. Introduction

Zero-knowledge (ZK) protocols [17] are a cornerstone of modern cryptography; they can express all NP computations [16], and are essential to almost any form of secure computation.

The ZK guarantee of a protocol is established by exhibiting an efficient *simulator* that can simulate the entire view of any malicious verifier from the verifier's code and the statement alone. Following the common practice of black-box reductions, the first ZK protocols all relied on simulators that only use the verifier as a black-box, without making any explicit use of its code. However, while sufficient for a variety of powerful applications, ZK protocols with black-box simulation were soon shown to have inherent limitations. A known example is the impossibility of constant-round public-coin ZK [15]. Surpassing such black-box impossibilities was considered to be an inconceivable task.

This barrier was crossed with the groundbreaking result of Barak [1] that introduced non-black-box simulation techniques, which allowed, in particular, to achieve constant-round public-coin ZK. The non-black-box techniques of Barak were then utilized to achieve various cryptographic goals, most of which where previously limited by black-box impossibilities [2], [4], [6], [7], [11], [18], [19], [22]–[26].

Room for new non-black-box techniques: Barak's result, indeed, opened up a new frontier of non-black-box research. However, our understanding of non-black-box techniques is yet far from being satisfactory; in fact, we do not know of any non-black-box techniques that do not rely on (or extend) those of Barak. Also, we still do not understand to what extent the tools and assumptions used by Barak are inherent.

For example, a dominant feature of Barak's protocol is the reliance on *universal arguments* (UAs) [3], which in turn use rather heavy PCP machinery. While, in theory, the state of the art PCPs may guarantee good asymptotic complexity, from a practical perspective, their efficiency is still not well understood. Another feature of Barak's protocol (which might be a byproduct of using UAs) is the reliance on *collision-resistant hash functions*.

Are Barak's assumptions and tools inherent in non-black-box simulation?
Can its applications be achieved without universal arguments? and from which assumptions?

Resetting attacks: One field that developed tremendously due to Barak's techniques is the study of *resetting attacks* [8]. In the resetting model, honest parties are restricted to one fixed random tape, while the adver-

Supported by the Check Point Institute for Information Security, a Marie Curie grant PIRG03-GA-2008-230640, an ISF grant 0603805843, an NSF grant 1218461, and the Fulbright program.

0272-5428/12 $26.00 © 2012 IEEE
DOI 10.1109/FOCS.2012.40

sary has the power to "reset" these parties to their initial state, repeating the interaction in any way it chooses (equivalently, honest parties can be rewound to any previous state). Indeed, the threat of reset attacks arises in real life settings where fresh randomness cannot be generated on the fly, and parties are subject to physical resets; common examples are: parties running on smart cards or virtual machines (which emulate real machines and, typically, support state snapshot and revert features).

Canetti et al. [8] defined and constructed resettably-ZK protocols (rZK) that remain ZK against resetting verifiers. Barak et al. [4] defined and constructed resettably-sound ZK protocols (rsZK) that remain sound against resetting provers. Deng, Goyal and Sahai [11] solved the (long standing open) question of constructing protocols that are simultaneously resettably-ZK and resettably-sound. Leveraging on these results, positive results have also been shown for the more general setting of resettable two-party and multi-party computation [18], [19]. In addition, there has been a large body of work trying to construct improved rZK and rsZK protocols in the relaxed *bare public-key model* [8], [10], [21]; in this work, however, we focus on the plain model.

All the works mentioned above encounter the non-black-box barrier: rsZK cannot be based on black-box simulators (except for languages in BPP) [4], [15]. Indeed, a resetting prover has, essentially, the same power as a black-box simulator, and can hence utilize such a simulator to break soundness. To overcome this inherent difficulty, [4] rely on Barak's non-black-box protocol. Indeed, constant-round public-coin protocols suggest a rather direct approach for obtaining resettable-soundness: instead of sending random coins, apply a pseudo-random function to all messages previously sent by the prover. All the other resettably-sound protocols mentioned above also rely on the non-black-box techniques of Barak (notably, [11] extend Barak's techniques in a powerful and beautiful way).

Resettable-soundness with lighter machinery: Going back to the reliance on UAs in Barak's techniques, we note that efficiency and simplicity are of major importance for resettably-sound ZK protocols, as we would like to execute these protocols on computationally weak devices. However, for the purpose of resettable-soundness it may be possible to use lighter machinery. Indeed, Barak's protocol achieves constant-round public-coin ZK, a seemingly harder problem than rsZK; however, going to the model of resetting must result in a private-coin solution (e.g., the pseudo-random function described above). One may hope that, by directly considering a private-coin solution, it is possible to gain in terms of simpler and more efficient machinery, and perhaps even assumptions.

Can we base resettably-sound ZK on alternative non black-box techniques? Can we avoid universal arguments? Can we rely on different assumptions?

A. Our Results

In this work, we construct a constant-round resettably-sound ZK protocol in the plain model, using new non-black-box simulation techniques. The protocol is also a resettable argument of knowledge. Our protocol does not invoke UAs and relies on rather simple and practical tools. The protocol can be based on *semi-honest oblivious transfer*, which is incomparable to the existence of collision-resistant hash-functions used in previous protocols. Interestingly, the main ingredient in our construction is the *impossibility of general program obfuscation* shown by Barak et al. [5]. In the converse direction, we show a generic transformation from any resettable argument of knowledge ZK protocol to a family of functions that cannot be obfuscated.

B. Paper Organization

In Section II, we describe the high-level ideas behind our protocol. We then go over the main issues arising when trying to follow these high-level ideas, and explain how to resolve them. In Section III, we present an overview of the entire protocol and state our main theorem. In Section IV, we present an overview of the ZK simulation procedure. In Section V, we show that a converse relation between rsZK and unobfuscatable functions also holds.

The detailed protocol, as well as the proofs of security, are omitted here, and can be found in the full version of this paper. The formal definitions of resetabble-soundness and unobfuscatable functions can be found in Appendices A and B. Other formal definitions can be found in the full version of this paper.

II. Our Protocol and Techniques - an Overview

To describe the main ideas behind our protocol, we first recall the notion of *unobfuscatable functions*.

Unobfuscatable functions: The problem of program obfuscation deals with "compiling" a given program $\mathcal{P}$ into an obfuscated analog $\mathcal{O}(\mathcal{P})$ that, on one hand, has the same functionality as $\mathcal{P}$, but on the other, does not leak any other information on $\mathcal{P}$ (except for its input-output behavior). Assuming only one-way functions, Barak et al. [5] construct a family of functions

$\{f_k\}$ that is unobfuscatable in the following sense: first, no efficient algorithm that is given black-box access to a randomly chosen f_k can learn k. Second, there is an efficient extractor E that, given the description of *any* program that computes f_k, manages to learn k. In particular, any obfuscation $\mathcal{O}(f_k)$ leaks information beyond the input-output behavior of f_k.

An abstract rsZK protocol: At a very high-level, our protocol follows the FLS paradigm [12]; namely, the protocol proceeds in two phases: the first phase defines a *hard to compute trapdoor*, and in the second phase, the prover gives a *witness indistinguishable* (WI) proof that either the statement is correct or that it "knows" the trapdoor. In our case, to obtain resettable-soundness, we will have to ensure that the trapdoor is actually "hard to compute", even given the ability to reset the verifier, and also that the WI proof given at the second phase is resettably sound. For ZK, we will exhibit a simulator that, given the code of the verifier, can obtain the trapdoor and successfully complete the simulation.

More concretely, in the trapdoor phase of our protocol, the verifier chooses a random function f_k from a family of unobfuscatable functions and commits to k, which will set as the trapdoor. Then, the prover and verifier will engage in a secure function evaluation protocol (SFE), which should allow the prover to learn the value $f_k(x)$ for an input x of his choice. The SFE protocol should be able to guarantee privacy for the verifier (the SFE evaluator), and privacy and correctness for the prover (the SFE receiver). Specifically, from the perspective of the verifier, the protocol should only allow the resetting prover to learn $f_k(x)$ for multiple inputs x of his choice (or in other words, it should essentially be equivalent to black-box access to f_k). From the prover's perspective, the protocol should ensure that the prover never accepts an incorrect value $y \neq f_k(x)$; moreover, as we will see later on, we will also need the SFE protocol to guarantee that the verifier cannot distinguish different prover inputs from one another.

In the proof phase, after the SFE execution is completed, the prover provides a WI argument of knowledge of either k or of a witness for the statement (the WI argument of knowledge should also hold against a resetting prover).

Establishing resettable-soundness and ZK: The above strategy seem to yield resettable-soundness and ZK in the following way. Resettable-soundness should follow from the fact that, effectively, the resetting prover only gains black-box access to f_k, which is unobfuscatable, and hence, in particular, not black-box learnable. ZK would rely on the fact that, given any code that computes f_k, one can efficiently extract the secret k. This suggests that the ZK simulator, which is given the code of the verifier, may be able extract the trapdoor and successfully simulate. To transform the verifier's code to a code that computes f_k, we rely on the fact that the SFE protocol guarantees both correctness and input privacy for the prover. Indeed, correctness guarantees that the verifier cannot deviate from the SFE protocol, except by aborting. Input privacy guarantees that as long as the verifier does not abort and evaluates f_k with noticeable probability, for some input x, it must also evaluate f_k, for all other inputs x', with roughly the same probability. This means that, using repetitions, the simulator can obtain a code that (almost perfectly) computes f_k on all inputs, and thus extract the trapdoor.

When realizing an SFE protocol with the appropriate features, as well as executing the above simulation strategy, we encounter several technical difficulties. We next describe the main challenges and the way we overcome each one of them.

Implementing the abstract protocol: challenges and solutions: A natural starting point for obtaining the appropriate SFE protocol is any semi-honest two-party SFE protocol, which can be constructed, for example, based on semi-honest oblivious-transfer (OT) and Yao's garbled circuit technique [20].

The next step would be to enforce semi-honest behavior on both the resetting prover and the verifier. Clearly, we cannot simply GMW-compile the protocol by adding ZK proofs of valid behavior; for this to work, the ZK proofs given by the prover should be resettably-sound, and this is exactly what we set out to achieve. In fact, for GMW-compilation, soundness is not enough — we need an argument of knowledge property. This poses a problem when considering the verifier's proofs of honest behavior; the verifier would need to give resettably-ZK proofs of knowledge, and such proofs are only known to exist relying on resettably sound ZK (and require non-black-box extraction).

Relation to resettable two-party computation [19]: The problem of designing SFE protocols where one party can be reset was addressed by Goyal and Sahai [19]. (We consider SFE protocols where only the non-resettable party obtains output, while [19] consider general two-party protocols.) In this work, similarly to [19], we compile a semi-honest SFE protocol into one that is secure against malicious parties in the presence of resets. The key difference is that, being interested in resettable SFE as a goal, Goyal and Sahai aim at satisfying a strong and general notion of security. Naturally, their construction relies on rsZK protocols as a main tool.

Aiming to construct rsZK, we clearly cannot use the [19] construction. To compile a semi-honest SFE into one that is secure against malicious parties, we will use substantially weaker tools and obtain a weaker security guarantee than the one obtained by [19]. Nevertheless, this, together with the properties of the unobfuscatable function family in use, will be sufficient for our purposes. (In particular, we do not use full-fledged resettable-ZK, and are, therefore, able to maintain a constant number of rounds.)

We now go further into the protocol, explaining how to resolve the above issues (as well as additional issues that arise in the process). We start by describing in Section II-A how to enforce semi-honest behavior of the resetting prover and obtain resettable-soundness. We then describe in Section II-B how to enforce semi-honest behavior on the verifier (without compromising resettable-soundness). Then, in Section II-C, we review the main challenges that arise in establishing ZK. In Section II-D, we compare our non-black-box technique to that of Barak's.

A. Challenges in Proving Resettable-Soundness

At high-level, to show resettable-soundness, we would like to follow the same paradigm as in [4]; namely, the verifier V will have a hardwired pseudo-random function, and will derive its randomness for each round by applying the function to the protocol's transcript so far. Then, we would like to transform any resetting prover P^* into a non-resetting prover $\tilde{\mathsf{P}}^*$ for one instance of the protocol and rely on the soundness of the stand-alone protocol. Unlike [4], who apply this paradigm to a public-coin protocol, we would like to apply it to a private-coin protocol; that is, the verifier will also apply a pseudo random function to derive its randomness for each and every round, but some of this randomness will be kept private and used in subsequent stages of the protocol. This difference makes the transformation from a resetting P^* to a non-resetting $\tilde{\mathsf{P}}^*$ for one instance of the protocol more involved.

Specifically, we show how to transform any convincing resetting P^* to a non-resetting $\tilde{\mathsf{P}}^*$ that also gets black-box access to the function f_k. Then, we deduce soundness based on: (a) the fact that k cannot be learned, given black-box access to f_k, (b) the stand alone security of the SFE protocol (which is yet to be established), and (c) the prover's WI argument of knowledge (in the proof phase).

The transformation from the resetting P^* to the non-resetting $\tilde{\mathsf{P}}^*$: Like in [4], the high-level idea behind the transformation is to note that any convincing interaction, between the verifier V and the prover P^*, includes a "convincing session" among the multiple sessions that the prover engages (via resetting). The strategy applied by $\tilde{\mathsf{P}}^*$ is to internally emulate P^* in all sessions, except in a random one, which will be conducted with the external verifier V. If we manage to *emulate all other sessions consistently with the external one*, and in addition "hit" the convincing session, we will convince the external verifier.

Hitting the convincing sessions occurs with noticeable probability so long as the number of rounds in the protocol is constant (in fact, we show that we can also deal with SFE protocols that have a super-constant number of rounds). The main challenge is the emulation of the internal sessions, and more concretely, of the SFE protocol. Indeed, such emulation typically requires that we will be able to simulate "continuations" for sessions that share a prefix with the external session, while we do not have the coins used by the external verifier. In particular, all sessions must be emulated with respect to the same k chosen by V at the beginning of the protocol.

When emulating the internal SFE protocol, we will have to manage without having k, but only having black-box access to f_k. As a warmup, we can consider provers that employ a semi-honest resetting strategy: they invoke many SFE sessions with different inputs, but each one is done semi-honestly and without any resettings within. For such provers, we can invoke the simulator of the semi-honest SFE, given the randomness used by the prover, its input for the computation, and the f_k-oracle.

To enforce such behavior, we can try and perform a GMW-like (coin flipping plus validity proofs) compilation: the prover commits to its input x and random coins r, and provides a WI argument of knowledge (WIAOK) of (r, x) or of a witness for the statement. Then, the verifier sends a random r' and, from this point on, the prover is required to provide after each step a WI proof that it followed the semi-honest protocol with coins $r \oplus r'$ and input x. This also has the effect that the prover cannot open more than one session after his commitment, implying that we will not need to simulate continuations of the external session (which is somewhat similar to the [8] transformation). All the WI proofs given are standard 3-round proofs (e.g., Blum), and are, in particular, resettably sound.

A concurrency problem and its resolution: The above approach encounters a problem when trying to invoke the AOK guarantee (of the first proof); indeed, to obtain (x, r), $\tilde{\mathsf{P}}^*$ must apply the AOK extractor that has to rewind P^*. However, since P^* may start a new session at any point, we might be required to

perform recursive rewinding, which, as in the setting of *concurrent ZK*, may result in an exponential blowup. The recursive simulation problem can be solved using known techniques from the concurrent ZK regime (e.g, [27]), but these involve adding a large number of rounds. Instead, we choose an alternative approach that avoids the problem of recursive rewinding and allows us to maintain a constant-round protocol: we add a preliminary phase to the protocol, in which the prover commits to a trapdoor, and proves that it knows the trapdoor (or a witness for the statement) using a WIAOK. Given such a trapdoor, we can later extract the prover's input and randomness (x, r) without having to rewind during the SFE emulation. For example, the prover's trapdoor can be a secret key sk of a symmetric key encryption scheme, and the prover will send (x, r) encrypted under sk.

A key point is that the prover's trapdoor is set *before* the verifier commits to k. If P^* resets the verifier and commits to a different trapdoor, the verifier will commit to a different value k'. This means that we only need to extract the trapdoor for sessions that share the external commitment to sk; other sessions can be simulated from scratch, by choosing k' at random and simulating honestly. It is important to notice that after the initial trapdoor phase, the prover is only committed to sk and may still choose x independently of k; this will be crucial for simulation. However, it turns out that by introducing the initial commitment to sk, we have introduced a subtle problem in the simulation, which we explain next.

B. The Verifier's Proofs: Resettable WI Arguments of Knowledge

As outlined above, to establish the ZK guarantee of the protocol, we will need to ensure that the prover's inputs to the SFE protocol remain private, and that the verifier indeed evaluates the function f_k it committed to. For this purpose, we add a coin-flipping stage to determine the coins to be used by the verifier in the SFE protocol, and have V^* prove after each step that it followed the semi-honest protocol with respect to k and the coins established in the coin-flipping stage.

For this to work, the verifier's proofs (at least the first one) should be an argument of knowledge (AOK); on the other hand, they should be hiding against a resetting prover so as not to compromise resettable-soundness. This presents a seemingly inherent limitation: proofs cannot be resettably ZK (or WI) and an AOK at the same time, assuming that the knowledge extraction is black-box. Barak et al. [4] construct (constant-round) rWIAOK and (super-constant-round) rZKPOK using non-black-box extraction; however, their constructions rely on rsZK, which is what we are trying to obtain in the first place.

Instead, we make use of an instance-dependent rWIAOK^y protocol. The protocol is not rWI and AOK simultaneously, but rather, if the instance y is *not* in the language $\mathcal{L}$, the protocol is rWI, while if $y \in \mathcal{L}$, the protocol is an AOK with a black-box extractor that works given a witness $w \in \mathcal{R}_{\mathcal{L}}(y)$. Such a protocol is, indeed, sufficient as the AOK property is only needed to establish ZK and the rWI property is only needed for resettable-soundness. We use this protocol (where y is the instance for which the proof is given) in order to have the verifier prove, at any point, that it follows the semi-honest SFE protocol with respect to f_k (and the coin flipping) or that it "knows" the prover's trapdoor sk. (We note that the ZK simulator does not use the knowledge extractor of the rWIAOK^y directly, since it makes use of a witness w, which the simulator lacks. We only use the AOK property in a reduction showing that the verifier does not break the semantic security of the SFE scheme, and does not choose f_k dependently on the prover's trapdoor sk.)

Instance-dependent rWIAOK from trapdoor commitments: The main tool we use to construct an rWIAOK^y protocol is a trapdoor commitment Com^y [9], [13] (also known as an equivocal commitment). Such a commitment scheme Com^y is (statistically) binding when $x \notin \mathcal{L}$, but if $x \in \mathcal{L}$, one can simulate a commitment and use the witness w in order to open the commitment to any value. Given such a commitment, we can take a standard 3-round public coin WIAOK protocol and add an initial message where the verifier of the rWIAOK^y (the prover in our rsZK protocol) commits to its challenge using Com^y. If $y \notin \mathcal{L}$, the verifier is committed to its challenge and cannot benefit from resetting; indeed, [8] show that such a protocol is rWI. If $y \in \mathcal{L}$, we can use the witness w to equivocate the commitment and just run the knowledge extractor of the WIAOK protocol. (In fact, the scheme has a slightly stronger property — it allows anyone to extract, from any resetting WI distinguisher, a witness $w \in \mathcal{R}_{\mathcal{L}}(y)$. This property, eventually, allows us to get an argument of knowledge and not only resettable-soundness.)

C. Challenges in ZK

Recall that the high-level idea behind the ZK simulation is to derive, from the code of any malicious verifier V^*, a circuit C that computes the function f_k, and then use the extraction procedure $\mathsf{E}(C)$, given by the fact that the f_k is drawn from an unobfuscatable family. Concretely, the ZK simulator $\mathcal{S}$ honestly simulates the

trapdoor phase: it selects a random sk, feeds V^* with a commitment to sk, and obtains a commitment to some k. Then, $\mathcal{S}$ will use the code of the residual verifier in order to construct the required circuit C. The natural way to achieve this is to "wrap" the verifier's circuit with a (randomized) circuit that gets an arbitrary input x, and emulates the SFE phase with the residual V^*.

However, by now the simulator $\mathcal{S}$ is committed to a specific trapdoor sk; in particular, $\mathcal{S}$ (or actually the SFE emulator wrapping V^*) should encrypt any input x with the secret key sk. This may pose an obstacle if we want the circuit C to compute the exact same function as f_k on all inputs x, because of possible dependencies between x and sk. To exemplify this point, we can think of sk as a key for a one-time-pad encryption, and assume that V^* decides to abort whenever it obtains an encryption $x \oplus \mathsf{sk} = 0^{|\mathsf{sk}|}$, which implies that V^* can abort whenever $\mathsf{sk} = x$, without actually knowing sk.

The question is whether the extractor E of the unobfuscatable family can do with a circuit C that does not compute f_k exactly. To try to understand the answer, it may be instructive to look into the extractor E constructed for the unobfuscatable family of [5]. Their extractor, given input circuit C, evaluates C as a black-box on several inputs that are derived from the code of C itself. If, for some input x chosen by E, $C(x) \neq f_k(x)$, then E may fail to extract k. In our case, the circuit C constructed by $\mathcal{S}$ implicitly depends on sk (for example, it uses sk to encrypt x) and hence inputs selected by E may also implicitly depend on sk.

Strongly unobfuscatable functions: To ensure that the inputs queried by E are independent of sk, we use another notion, introduced by [5], of strongly unobfuscatable functions. These are functions f_k, where k cannot be learned using black-box access to f_k, but k can be learned given a random circuit from a family that "approximates" f_k rather than computes it exactly (for a precise definition, see Section B). Barak et al. also show how to construct such strongly unobfuscatable functions assuming one-way functions. The extractor E that they construct selects inputs that are almost independent of the structure of C and only depend on its functionality (specifically, these inputs are chosen from one of a constant number of fixed distributions).

Using such unobfuscatable functions, our task becomes much simpler as it is reduced to showing that the function f_k chosen by V^* is independent of sk (in some computational sense, guaranteed by the hiding of the commitment to sk). To show that this is indeed the case, we require that V^* provides an AOK of the function f_k it commits to; this prevents, for example, mauling attacks where V^* obliviously uses the commitment to sk to commit to a related k. Obtaining an AOK proof which is rZK or rWI brings forward another problem to be dealt with, which we discuss in the next paragraph.

We note that, eventually, in order to use strongly unobfuscatable functions as is, we are required to commit to sk, using a *statistically-hiding commitment*; however, by using unobfuscatable functions with slightly stronger properties, we can settle for computationally hiding commitments as well. The stronger properties required are satisfied by the [5] construction (which only relies on one-way functions).

Simulation running time: Finally, we touch the main issues concerning the running time of the ZK simulator. At high-level, the simulation time depends on the probability ϵ that the cheating verifier V^* does not abort before evaluating the function f_k on the prover's input. When ϵ is noticeable, we can construct a circuit C that computes f_k with overwhelming probability by repeating the code of V^* roughly $O(n/\epsilon)$ times (where n is the security parameter or, say, the size of the instance). If ϵ is negligible, simulation is trivial, since V^* almost always aborts before we need to simulate any proof. Concretely, to control the expected simulation time, we follow the approach of Goldreich and Kahan [14], where a similar situation is encountered.

While the expected time required to construct the circuit C is polynomial, the part of the simulation that dominates the running time is not the construction of the circuit, but rather the running of the (strongly) unobfuscatable function-family's extractor on C. For example, the extractor of [5] runs in time $O(|C|^2)$, which will result in expected simulation time $(\mathrm{poly}(n)/\epsilon)^2 \cdot \epsilon$ which may not be bounded by a polynomial in case ϵ is negligible. To cope with this problem we add several more rounds to the protocol: the prover and verifier will run the SFE protocol twice sequentially. Now, if the probability that V^* does not abort in both executions is ϵ, we can show that at least in one of the executions, V^* does not abort with probability at least $\sqrt{\epsilon}$. The size of the circuit C is therefore $O(n/\sqrt{\epsilon})$ and, accordingly, the expected running time of the extractor is $(\mathrm{poly}(n)/\sqrt{\epsilon})^2 \cdot \epsilon = \mathrm{poly}(n)$. (In the general case, if the extractor for the unobfuscatable function family runs in time $O(|C|^d)$, for some constant d, we can repeat the SFE protocol d times sequentially to get the same result).

D. Comparing our Technique to Barak's

Barak's non-black-box technique [1] makes essential use of universal arguments (UAs) and collision-resistant hash functions, while our technique does not; this can

be seen as the result of a more fundamental difference between the two techniques. Both our protocol and Barak's protocol follow the FLS paradigm, in which the prover ultimately proves a statement that has a trapdoor witness. In Barak's protocol, this trapdoor is the code of some program, the length of which is not bounded by any fixed polynomial. In particular, this program may be as long as any cheating verifier. Proving a statement using such a long witness is made possible using UAs and collision-resistant hash functions.

In contrast, in our protocol, the trapdoor witness is a random key for the unobfuscatable function family and is of fixed polynomial length. The use of such a "short" trapdoor is made possible since, in our protocol, the verifier has private coins. (Indeed, in public-coin protocols, the verifier has no secrets, and hence a secret trapdoor cannot be determined by the protocol's transcript alone.) We believe that our non-black-box simulation technique with short trapdoor will be useful in other applications as well.

One additional advantage of using a short trapdoor is that it allows us to transform our rsZK protocol into a protocol that is simultaneously a proof; namely, it is sound against unbounded non-resetting provers. (Indeed, [4] show that it is impossible to get security against an unbounded resetting provers.) We note, however, that such a protocol, which is simultaneously a rsZK argument and a stand-alone proof, can also be constructed by sequentially composing any rsZK argument with a stand-alone proof. The details of the transformation of our rsZK protocol into a proof are omitted.

III. The Full Protocol

We now provide a high-level description of the full protocol described through the overview of Section II.

Protocol 1 gives rise to the following theorem:

Theorem III.1. *Assuming constant-round semi-honest oblivious transfer, there exists a constant-round resettably-sound ZK argument system. The system is also a resettable argument of knowledge. The system does not invoke universal arguments.*

IV. An Overview of the Simulator

In this section, we provide a high-level overview of our non-black-box simulator.

The simulator $\mathcal{S}$ proceeds in three main steps:

1) $\mathcal{S}$ simulates all the interaction up until the Proof of Statement Phase in a black-box way by following the honest prover strategy. Indeed, the prover only makes use of the witness in the proof phase. If at any point the verifier aborts (or causes the honest prover to abort), the simulator $\mathcal{S}$ also aborts and outputs the transcript up to this point. Otherwise, it proceeds to the next step.
2) $\mathcal{S}$ runs an extraction procedure Ext to extract the key (for the unobfuscatable function) that the verifier committed to in the Verifier's Trapdoor Commitment Phase. The extraction procedure will make use of the code of the cheating verifier and is described below.
3) After extracting the verifier's trapdoor, simulate the interaction in the Proof of Statement Phase in a black-box way using the verifier's trapdoor as a witness.

The non-black-box trapdoor extraction procedure Ext*:* Ext uses the code of the residual verifier whose state is set to right after the committment to a key k for an unobfuscatable function. Recall that the Function Evaluation Phase consists of d executions of the SFE protocol, with respect to the same function f_k. The extraction procedure proceeds as follows:

1) First, the extractor approximates the probability p_i that the verifier does not abort (or deviates from the protocol in a way that causes an abort) in each particular iteration i among the d iterations. This is still done in a black-box way using rewinding, by repeatedly running the verifier up to the i-th SFE execution, using the same input $x = 0^n$ and fresh randomness.
2) Let i be such that the approximated value of p_i is maximal. The extractor Ext now uses the code of the verifier to construct a new circuit F that computes f_k with high-probability. On input x the circuit F interacts with the code of the verifier, executing the i-th SFE protocol on input x. If the execution of the i-th SFE protocol does not abort, F will get an output from the SFE protocol and output the same. In case of abort, F will output $\perp$. Intuitively, it follows from the correctness and security of the SFE protocol that, for every x, $\mathsf{F}(x) = f_k(x)$ with probability negligibly close to p_i, and $\perp$ with probability negligibly close to $1-p_i$.
3) Next, the extractor creates an amplified circuit F^m that executes m copies of F with independent randomness on the same input, and outputs the same as the first copy that does not output $\perp$. If all copies output $\perp$, F^m outputs $\perp$ as well. Here, m is set according to the approximation of p_i, so that F^m outputs $\perp$ (on any input) only with negligible probability, and with overwhelming

Protocol 1

Input and randomness:

- The prover P and verifier V have joint input x. P has additional input $w \in \mathcal{R}_{\mathcal{L}}(x)$.
- The verifier's randomness is fixed to a seed for a function F drawn from a pseudo random function family. In each round, the verifier will apply the function F to the transcript of all sessions so far, in order to derive randomness for the current round.

P and V execute the following protocol:

Prover's Trapdoor Commitment Phase:

- P commits to a random secret key sk for a symmetric key encryption, as its trapdoor.
- P proves with a WIAOK that $x \in \mathcal{L}$ or that it knows sk.

Verifier's Trapdoor Commitment Phase:

- V commits to a random key k for an unobfuscatable function f_k, as its trapdoor.

Function Evaluation Phase:

- P and V run a coin-flipping protocol to obtain randomness $(\mathsf{r}_\mathsf{P}, \mathsf{r}_\mathsf{V})$ to be used in the SFE protocol.
- P sends an encryption under sk of both its randomness r_P and an input $y := 0^n$ for the SFE protocol. Then it proves with a WIAOK that the encryption is consistent with its commitment to sk, or that $x \in \mathcal{L}$.
- P and V run the semi-honest SFE protocol, at the end of which P should learn $f_k(y)$. P uses the input y and randomness r_P. V uses the input f_k and the randomness r_V.
- After every message P sends in the SFE protocol, P proves with a WIAOK that the message was computed honestly with respect to $(y, \mathsf{r}_\mathsf{P})$, or that $x \in \mathcal{L}$.
- After every message that V sends in the SFE protocol, V proves with an x-dependent rWIAOK^x that the message was computed honestly with respect to $(k, \mathsf{r}_\mathsf{V})$, or that it knows the prover's trapdoor sk.
- P and V repeat the function evaluation phase $d = O(1)$ times, where d is the determined by the running time of the extractor for the unobfuscatable function family.

Proof Phase:

- P proves using a WIAOK that it knows the verifier's trapdoor k, or that $x \in \mathcal{L}$.

Figure 1. A constant-round rsZK protocol for $\mathcal{L}$

probability computes the function f_k [1].

4) Finally, Ext invokes the extractor of the unobfuscatable function family on the circuit F^m and obtains the verifier's trapdoor k.

As outlined in Section II-C proving the validity of the above simulation process encounters several challenges, such as proving the correctness of the circuit F^m, based on the properties of the SFE protocol and analyzing the running time of the simulator (which is done similarly to [14]). The details are given in the full version of this paper.

V. From rsZK back to Unobfuscatable Functions

Our main construction provides a general transformation from a family of (strongly) unobfuscatable functions to an rsZK argument of knowledge. We show that, in fact, a converse relation also holds by exhibiting a simple transformation from any rsZK argument of knowledge to a family of unobfuscatable functions.

We give an informal overview of this transformation. Given a d-round rsZK argument of knowledge $\langle \mathsf{P}, \mathsf{V} \rangle$ with d rounds, and given a one-way function g, we construct a family of functions $\{f_k\}_{k \in \{0,1\}^n}$ that implements the honest verifier strategy on some "hard to prove statement". More precisely, we interpret the key of the function as a triplet $k = (x, \mathsf{r}, b)$, where x is an input to g, r is randomness for V and b is a bit. The function f_k is defined as follows:

1) Given a special input GET-IMAGE, f_k outputs $y = g(x)$.
2) Given a partial protocol transcript consisting of prover messages $\mathsf{T} = (p_1, \ldots, p_i)$ for $i < d$, f_k runs the honest verifier V with the random tape r and the statement "y is in the image set of g" and feeds it with the messages $(p_1, \ldots, p_i)$ as the first i prover messages. f_k obtains the next verifier message v_i generated by V and outputs v_i.
3) Given a full transcript $\mathsf{T} = (p_1, \ldots, p_d)$, f_k runs V_r as above. If V_r accepts, f_k outputs b; otherwise,

[1] In the actual proof, we will only be able to construct a circuit that approximates the function f_k, which will still be sufficient for the extraction to succeed

it outputs $\perp$.

We argue (informally) that $\{f_k\}_{k\in\{0,1\}^n}$ is a family of unobfuscatable functions; indeed:

1) b **is unlearnable:** no poly-size algorithm L that is given black-box access to f_k where k is sampled uniformly, can guess b with more than negligible advantage (over a random guess). Indeed, the bit b remains information theoretically hidden, unless L produces a convincing proof for the fact that $y = g(x)$ for some x. Since L only interacts with f_k as a black-box, we can easily transform L into a resetting adversary for the rsZK argument of knowledge, which convinces the resettable verifier V. Therefore, by invoking knowledge extractor, we can also extract x and invert g, in contradiction to its one-wayness. (Note that the function f_k can, indeed, be constructed from $g(x)$ alone, without any extra information regarding x.)
2) b **is learnable from any code:** there exists a PPT algorithm Ext that, given any circuit C that computes the function f_k, can learn the value of b. Ext uses the code of C to construct an interactive verifier V' for the rsZK protocol. Ext then runs the ZK simulator with the code of V' and obtains a transcript T. It follows from the correctness of the simulation that T is, indeed, an accepting transcript. Ext runs C on the sequence of prover messages in T and gets as outputs the bit b.

Acknowledgements

We thank Ran Canetti for valuable discussions and comments.

REFERENCES

[1] B. Barak, "How to go beyond the black-box simulation barrier," in *FOCS*, 2001, pp. 106–115.

[2] ——, "Constant-round coin-tossing with a man in the middle or realizing the shared random string model," in *FOCS*, 2002, pp. 345–355.

[3] B. Barak and O. Goldreich, "Universal arguments and their applications," *SIAM J. Comput.*, vol. 38, no. 5, pp. 1661–1694, 2008.

[4] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell, "Resettably-sound zero-knowledge and its applications," in *FOCS*, 2001, pp. 116–125.

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *CRYPTO*, 2001, pp. 1–18.

[6] B. Barak and Y. Lindell, "Strict polynomial-time in simulation and extraction," in *STOC*, 2002, pp. 484–493.

[7] B. Barak and A. Sahai, "How to play almost any mental game over the net - concurrent composition via super-polynomial simulation," *Electronic Colloquium on Computational Complexity (ECCC)*, no. 096, 2005.

[8] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, "Resettable zero-knowledge (extended abstract)," in *STOC*, 2000, pp. 235–244.

[9] I. Damgård, "On the existence of bit commitment schemes and zero-knowledge proofs," in *CRYPTO*, 1989, pp. 17–27.

[10] Y. Deng, D. Feng, V. Goyal, D. Lin, A. Sahai, and M. Yung, "Resettable cryptography in constant rounds - the case of zero knowledge," in *ASIACRYPT*, 2011, pp. 390–406.

[11] Y. Deng, V. Goyal, and A. Sahai, "Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy," in *FOCS*, 2009, pp. 251–260.

[12] U. Feige, D. Lapidot, and A. Shamir, "Multiple noninteractive zero knowledge proofs under general assumptions," *SIAM J. Comput.*, vol. 29, no. 1, pp. 1–28, 1999.

[13] U. Feige and A. Shamir, "Zero knowledge proofs of knowledge in two rounds," in *CRYPTO*, 1989, pp. 526–544.

[14] O. Goldreich and A. Kahan, "How to construct constant-round zero-knowledge proof systems for np," *J. Cryptology*, vol. 9, no. 3, pp. 167–190, 1996.

[15] O. Goldreich and H. Krawczyk, "On the composition of zero-knowledge proof systems," *SIAM J. Comput.*, vol. 25, no. 1, pp. 169–192, 1996.

[16] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity for all languages in np have zero-knowledge proof systems," *J. ACM*, vol. 38, no. 3, pp. 691–729, 1991.

[17] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[18] V. Goyal and H. K. Maji, "Stateless cryptographic protocols," in *FOCS*, 2011, pp. 678–687.

[19] V. Goyal and A. Sahai, "Resettably secure computation," in *EUROCRYPT*, 2009, pp. 54–71.

[20] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *J. Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[21] S. Micali and L. Reyzin, "Min-round resettable zero-knowledge in the public-key model," in *EUROCRYPT*, 2001, pp. 373–393.

[22] R. Pass, "Simulation in quasi-polynomial time, and its application to protocol composition," in *EUROCRYPT*, 2003, pp. 160–176.

[23] R. Pass and A. Rosen, "Bounded-concurrent secure two-party computation in a constant number of rounds," in *FOCS*, 2003, pp. 404–413.

[24] ——, "Concurrent nonmalleable commitments," *SIAM J. Comput.*, vol. 37, no. 6, pp. 1891–1925, 2008.

[25] ——, "New and improved constructions of nonmalleable cryptographic protocols," *SIAM J. Comput.*, vol. 38, no. 2, pp. 702–752, 2008.

[26] R. Pass, A. Rosen, and W.-L. D. Tseng, "Public-coin parallal zero-knowledge for np," 2011.

[27] M. Prabhakaran, A. Rosen, and A. Sahai, "Concurrent zero knowledge with logarithmic round-complexity," in *FOCS*, 2002, pp. 366–375.

APPENDIX

A. Resettable Soundness

We briefly recall the definitions of resettable-soundness presented in [4]. In the setting of resettable-soundness, the prover P^* has the power to reset the verifier V. Specifically, the random tape of V is chosen at random and fixed once and for all and, from that point on, the prover can interact multiple times with the residual deterministic verifier $\mathsf{V}_\mathsf{r}(x)$ induced by r and the common input x. Each such interaction is called a session.

Note that the adversary may repeat in a current session the same messages sent in a prior session, resulting in an identical prefix of an interaction (since the verifier's randomness is fixed). Furthermore, by deviating in the next message, the adversary may obtain two different continuations of the same prefix of an interaction. (A generalization of the above model, also considered in [4], is to allow the prover to interact with multiple "incarnations" of the verifier. In this extended abstract, we restrict attention to the simpler model of a single incarnation, although our results extend to the multiple incarnations model.)

Definition A.1 (Resettably sound argument [4])**.** *A resetting attack of a malicious prover P^* on a resettable verifier V is defined by the following random process, indexed by a security parameter n:*

1) *Uniformly select a random-tape r for V, resulting in a deterministic strategy V_r.*
2) *For a given $x \in \{0,1\}^n$, a prover P^* of size $\mathrm{poly}(n)$ can initiate up to $\mathrm{poly}(n)$ complete interactions with $\mathsf{V}_\mathsf{r}(x)$.*

An argument system $\langle \mathsf{P}, \mathsf{V} \rangle$ is a resettably sound argument for $\mathcal{L}$ if, for any resetting poly-size P^*, the probability that in some session during a resetting attack, P^* convinces $\mathsf{V}_\mathsf{r}(x)$ to accept, while $x \notin \mathcal{L}$, is negligible in n.

B. Unobfuscatable Functions

At high-level, an unobfuscatable function ensemble $\{f_k\}$ is parameterized by a secret k, and has the guarantee that: (a) no efficient algorithm can learn k, given black-box to a random f_k drawn from the family, and (b) given any circuit (or code) which computes f_k, k can be efficiently extracted.

A strengthening of this notion is to consider a setting where extraction of k can also be done (with high-probability) given a random circuit from a distribution that only "approximates" f_k, rather than computes the exact same function. This notion was termed by Barak et al. [5] – "strongly unobfuscatable functions". Our formal definition follows that of Barak et al. [5].

Definition A.2 (Unobfuscatable functions)**.** *Let $\mathcal{F} = \left\{f_k : \{0,1\}^{\ell(n)} \to \{0,1\}^*\right\}_{k \in \{0,1\}^n, n \in \mathbb{N}}$ be an ensemble of circuits of size $\mathrm{poly}(n)$. $\mathcal{F}$ is* **unobfuscatable** *if:*

1) **It is black-box unlearnable:** *for any poly-size learning algorithm $\mathsf{L} = \{\mathsf{L}_n\}_{n \in \mathbb{N}}$, and for all large enough $n \in \mathbb{N}$:*

$$\Pr_{k \leftarrow \{0,1\}^n}[\mathsf{L}^{f_k} = k] \leq \mathrm{negl}(n) \ .$$

2) **It is learnable given the code of a proper approximation:** *there exists a PPT extraction algorithm E such that for any distribution on circuits $\mathcal{C}$ and any $k \in \{0,1\}^n$ such that $\mathcal{C}$ approximates f_k and is of bounded expected size:*
   - $\mathbb{E}_{C \leftarrow \mathcal{C}}[|C|] \leq S$
   - $\forall x \in \{0,1\}^{\ell(n)} : \Pr_{C \leftarrow \mathcal{C}}[C(x) = f_k(x)] > 1 - \epsilon$,

   the extractor E can extract k from C:

$$\Pr_{C \leftarrow \mathcal{C}}[\mathsf{E}(C) = k] \geq 1 - S \cdot \epsilon \ .$$

In this work, we can rely on any unobfuscatable function ensemble, such as the one constructed by Barak et al. from minimal cryptographic assumptions:

Theorem A.1 (Existence of (strongly) unobfuscatable functions [5])**.** *If there exist one-way functions, then there exist (strongly) unobfuscatable functions.*