Obfuscation-based Non-black-box Simulation and Four Message Concurrent Zero Knowledge for NP

Omkant Pandey* Manoj Prabhakaran† Amit Sahai‡

Abstract

As recent studies show, the notions of *program obfuscation* and *zero knowledge* are intimately connected. In this work, we explore this connection further, and prove the following general result. If there exists *differing input obfuscation* (diO) for the class of all polynomial time Turing machines, then there exists a *four message*, fully concurrent zero-knowledge proof system for all languages in **NP** with *negligible* soundness error. This result is constructive: given diO, our reduction yields an explicit protocol along with an *explicit* simulator that is "straight line" and runs in strict polynomial time.

Our reduction relies on a new non-black-box simulation technique which does not use the PCP theorem. In addition to assuming diO, our reduction also assumes (standard and polynomial time) cryptographic assumptions such as collision-resistant hash functions.

The round complexity of our protocol also sheds new light on the *exact* round complexity of concurrent zero-knowledge. It shows, for the first time, that in the realm of non-black-box simulation, concurrent zero-knowledge may not necessarily require more rounds than *stand alone* zero-knowledge!

1 Introduction

Zero-knowledge and program obfuscation. Zero-knowledge proofs, introduced by Goldwasser, Micali and Rackoff [GMR85] are the classical example of the *simulation paradigm*. They allow a *prover* to convince a *verifier* that a mathematical statement $x \in \mathbf{L}$ is true while giving *no additional knowledge* to the verifier. Prior to 2001, all known zero-knowledge simulators used the (cheating) verifier V^* as a *black-box* to produce their output (called the simulated view). Barak [Bar01] demonstrated how to take advantage of verifier's program to build, more powerful, *non-black-box* simulation techniques. Constructing and analyzing non-black-box simulators is significantly more challenging task.

The reason why taking advantage of verifier's code is difficult is because of the intriguing possibility of *program obfuscation*. Roughly speaking, program obfuscation is a method to transform a computer program (say described as a Boolean circuit) into a form that is executable but otherwise completely "unintelligible." In its strongest form, an obfuscated program leaks no information about the program beyond its "functionality" or the "input-output behavior". Therefore, access to the obfuscated program is no better than having black box access to it. This property, as formalized by Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [BGI+01], is called the *virtual black box* (VBB) security. It was shown in [BGI+01] that VBB-secure obfuscation is impossible in general. In the hindsight, this negative result is also the fundamental reason why non-black-box (NBB) simulation techniques prove to be more powerful than black box techniques.

*University of Illinois at Urbana-Champaign, Email: omkant@uiuc.edu

†University of Illinois at Urbana-Champaign, Email: mmp@uiuc.edu

‡University of California, Los Angeles, Email: sahai@cs.ucla.edu

Zero-knowledge, in particular non-black-box simulation, is intimately connected to program obfuscation. This connection has been explicitly studied in the works of Hada [Had00], and Bitansky and Paneth [BP12b, BP12a, BP13a], and alluded to in several other works, e.g., [HT99, Bar01]). In this work, we explore this line of research further, particularly in light of recent breakthrough work on *indistinguishability obfuscation* (iO) [GGH+13].

Indistinguishability obfuscation. Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13] present a candidate construction for a weaker notion of obfuscation called *indistinguishability obfuscation* [BGI+01]. Roughly speaking, iO guarantees that if two (same size) programs C_0, C_1 are functionally equivalent, then their obfuscations are computationally indistinguishable. A closely related notion is that of *differing input obfuscation* (diO) [BGI+01] which, roughly speaking, guarantees that the obfuscations of C_0 and C_1 are computationally indistinguishable *provided that* it is hard to find an input x such that $C_0(x) \neq C_1(x)$.

Garg et. al. [GGH+13] present a candidate construction of iO for the class of all polynomial size circuits. Candidate constructions of diO for the class of all polynomial time *Turing machines* were recently constructed by Ananth et. al. [ABG+13], and Boyle, Chung, and Pass [BCP14].

Our results. In this work we show how to use program obfuscation to build a new non-black-box simulation strategy that works for fully *concurrent* zero-knowledge. More specifically, we show that:

- If differing-input obfuscation (diO) exists for the class of all polynomial time Turing machines, then there exists a constant round, fully concurrent zero knowledge protocol for $\mathbf{NP}$ with negligible soundness error. The protocol has an *explicit* simulator;[1] the simulator is "straight line" and runs in strict polynomial time.
- We also show how to implement the core ideas of the above protocol in only *four* rounds. That is, our new protocol requires sending only four messages between the prover and the verifier.

Our protocol can be instantiated using the diO construction of Ananth et. al. [ABG+13] which obfuscates polynomial time Turing machines that can accept inputs of variable length (at most polynomial in the security parameter).[2] We stress that we are able to obtain an *explicit* simulator for our protocol *irrespective of the computational assumptions* underlying the above mentioned diO. This is because we use the security—i.e., indistinguishability property—of obfuscation only in proving the *soundness* of our protocol. The simulator only depends on the correctness or the *functionality* of the obfuscated program, and hence can be described explicitly. As is usually the case with most cryptographic applications of obfuscation, we also require that obfuscation is "secure" w.r.t. *auxiliary information*. In our case the auxiliary information will consist of the transcripts of Barak's preamble (see theorems 5.1 and 6.1 for a precise statement).

Other than (auxiliary input) diO, our reduction only assumes standard (polynomial time hardness) assumptions, namely injective one-way functions and collision-resistant hash functions. Interestingly, our reduction does not *explicitly* depend on CS-proofs/universal-arguments [Kil92, Mic94, Kil95, BG02]; in particular, if we instantiate the construction of [ABG+13] using the "SNARKs" of Bitansky et al. [BCCT13] (which does not rely on the PCP theorem), we obtain an instantiation of our protocol that also does not rely on the PCP theorem.

[1] In some protocols, specifically those based on knowledge-type assumptions [HT99], by virtue of the assumption that there exists an "extractor," we only obtain an *existential* result that a simulator exists; however, the actual program of the simulator is not explicitly given in the security proofs.

[2] See Section 3.2 for the cryptographic assumptions underlying the candidate construction of [ABG+13].

The round complexity of our final protocol also sheds new light on the *exact* (as opposed to asymptotic) round-complexity of concurrent zero-knowledge. Even in the simpler case of *stand alone* zero knowledge, the best known constructions require at least four rounds [FS89], and historically, concurrent zero-knowledge has always required more rounds than stand alone zero-knowledge.[3] Our four round protocol, for the first time, closes the gap between the best known upper bounds on round complexities of concurrent versus standalone zero-knowledge protocols (whose simulators can be explicitly described).

In retrospect, the fact that obfuscation actually *helps* non-black-box simulation can be perplexing. Indeed, in all prior works along this line [Had00, BP12b, BP13a], the core ideas for simulation are of *opposite* nature: it is the *inability* to obfuscate the "unobfuscatable functions" that helps the simulator. In our case, similar to [BP12a], it is the *ability* to obfuscate programs that allows polynomial time simulation.

1.1 Technical Overview: Non-black-box Simulation via Program Obfuscation

Let us start by considering the simplest approach to zero-knowledge from (the possibility of) program obfuscation. For now, let us restrict ourselves to the case of *stand alone* zero-knowledge for $\mathbf{NP}$-languages. Let $x \in \mathbf{L}$ be the statement and R be the witness-relation.

One simple approach is to have the verifier send an obfuscation of the following program $M_{x,s}$ which contains a secret string $s \in \{0,1\}^n$: $M_{x,s}(a) = s$ if and only $R(x,a) = 1$ and $M_{x,s}(a) = 0^n$ otherwise. Let $\widetilde{M}_{x,s}$ denote the iO-secure obfuscation of $M_{x,s}$. The real prover can recover s by using a witness w to x. Further, if x is false, $M_{x,s}$ is identical to $M_{x,0^n}$ and therefore must hide s, ensuring the soundness.[4] This gives us a two-message, *honest verifier* ZK proof. However, this idea does not help the simulation against malicious verifiers.

To fix this, let us try to use Barak's preamble (called $\mathsf{GenStat}$ [Bar01]) which has the following three rounds: first, the verifier sends a collision-resistant hash function $h : \{0,1\}^* \to \{0,1\}^n$, then the prover sends a commitment c to 0^n (using a perfectly binding scheme Com), and then the verifier sends a string $r \in \{0,1\}^n$. The transcript defines a "fake statement" $\lambda = \langle h, c, r \rangle$. A "fake witness" ω for the statement λ consists of a pair (Π, u) such that $c = \mathsf{Com}(h(\Pi)\ ;\ u)$ and Π is a program of length $\mathsf{poly}(n)$ which outputs the string r on input the string c (say, in $n^{\log\log n}$ steps). If h is a good collision-resistant hash function, then it was shown in [Bar01, BG02], no efficient prover P^* can output a satisfying witness ω to the statement λ (sampled in an interaction with the honest verifier). However, a simulator can commit to $h(V^*)$ (instead of 0^n) so that it will have a valid witness to the resulting transcript λ.

Coming back to our protocol, we use this idea as follows. We modify our first idea, and require the verifier to send a the obfuscation of a new program $M_{\lambda,s}$ (instead of $M_{x,s}$) where $\lambda = \langle h, c, r \rangle$ is the transcript of $\mathsf{GenStat}$. The new program $M_{\lambda,s}$ outputs s if and only if it receives a valid witness ω to the statement λ (as described earlier) and 0^n on all other inputs. To prove the statement x will be proven by proving the knowledge of either a witness w to x or the secret s (using an ordinary *witness-indistinguishable proof-of-knowledge* (WIPOK)). A simulator can "succeed" in the simulation as before: it commits to verifier's program in c to obtain (an indistinguishable statement) λ, then uses the fake witness ω (which it now has) to execute the program $\widetilde{M}_{\lambda,s}(\omega)$ and learn s and complete the WIPOK using s.

We now draw attention to some important points arising due to the use of λ in the obfuscation (instead of x). First, the length of the fake witness ω that the simulator has depends on the length of the program of V^*. Since the protocol needs to take into account V^* of *every* polynomial length, the obfuscated program

[3] Barak's (bounded-concurrent ZK) protocol [Bar01] and recent construction of Chung, Lin, and Pass [CLP13b] require at least six rounds even after optimizations; the recent protocol of Gupta and Sahai [GS12b] requires five rounds.

[4] By security of iO, $\widetilde{M}_{x,s} \overset{c}{\approx} \widetilde{M}_{x,0^n}$ and $\widetilde{M}_{x,0^n}$ has no information about s.

$\widetilde{M}_{\lambda,s}$ must accept inputs ω of arbitrary, a-priori unknown, polynomial length. In other words, the obfuscated program $\widetilde{M}_{\lambda,s}$ must be a *Turing machine* which accepts inputs of arbitrary, a-priori unknown, (polynomial) length. Therefore, we will have to use program obfuscation for Turing machines.

Second, the statement $\lambda = \langle h, c, r\rangle$ is not a "false" statement since an all powerful prover can always find collisions in h and obtain a satisfying input to $M_{\lambda,s}$. The only guarantee we have is that if λ is sampled as above, then it would be *hard* for any efficient prover—even those with a valid witness to x—to find a satisfying input for $M_{\lambda,s}$. Therefore, unlike before (when x was used instead of λ), obfuscations $\widetilde{M}_{\lambda,s}$ and $\widetilde{M}_{\lambda,0^n}$ are not guaranteed to be indistinguishable if we use an iO-secure obfuscation; this is because the Turing machines $M_{\lambda,s}$ and $M_{\lambda,0^n}$ are not functionally equivalent. Therefore, we will have to use diO-secure obfuscation (since finding a differing input is still hard for these programs). As a matter of fact, we will need to assume *auxiliary input* diO as discussed later.

By putting these ideas together, we actually a get a standalone ZK protocol for $\mathbf{NP}$ (summarized below). The protocol needs to use some kind of reference to s other than the obfuscated program. This is done by using a $f(s)$ where f is a one-way function. This protocol has a "straight line" simulator. Further, unlike Barak's protocol, this protocol does not use universal arguments (and hence the PCP theorem).

Standalone Zero-Knowledge using Obfuscation. The protocol has three stages.

1. Stage-1 is the 3 round preamble $\mathsf{GenStat}$: V sends a CRHF h, P sends a commitment $c = \mathsf{Com}(0^n; u)$ and V sends a random $r \leftarrow \{0,1\}^n$.
2. In stage 2, V sends $(f, \widetilde{s}, \widetilde{M}_{\lambda,s})$ where f is a one-way function, $\widetilde{s} = f(s)$, and $\widetilde{M}_{\lambda,s}$ is the obfuscation of *Turing machine* $M_{\lambda,s}$ described earlier and $\lambda = \langle h, c, r\rangle$ is the transcript of stage-1. V also proves that $(f, \widetilde{s}, \widetilde{M}_{\lambda,s})$ are correctly constructed (using a standard ZK proof).
3. In stage-3 P provess, using a standard WIPOK, the knowledge of "either a witness w to x or secret s such that $\widetilde{s} = f(s)$."

Standalone ZK of this protocol can be proven by following Barak's simulator which commits to the code of V^* and therefore has an ω for simulated statement λ such that $\widetilde{M}_{\lambda,s}(\omega) = s$ within a polynomial number of steps; the simulator computes s and uses it in the WIPOK. The soundness of the protocol relies on the diO-security of obfuscation. Indeed, following [Bar01], for a properly sampled λ, it is hard to find ω such that $M_{\lambda,s}(\omega) \neq M_{\lambda,0^n}(\omega)$, and therefore it is hard to distinguish $\widetilde{M}_{\lambda,s}$ from $\widetilde{M}_{\lambda,0^n}$ by diO-security of obfuscation. Now, soundness is argued using three hybrid experiments: first use the simulator of the ZK protocol in stage 2, then replace $\widetilde{M}_{\lambda,s}$ from $\widetilde{M}_{\lambda,0^n}$, and finally extract s from the WIPOK in stage 3 and violate the hardness of one-way function f (since x is false, extraction must yield s).

The issue of auxiliary information. An important point we wish to highlight here is that of *auxiliary input*. A cheating prover P^* in the protocol above, will have access to the statements λ in addition to the obfuscated program $\widetilde{M}_{\lambda,s}$. Therefore, λ is the *auxiliary information* that the receiver of the obfuscated program already has! Therefore, we must require the obfuscation to satisfy the (stronger) notion of *auxiliary input* diO [ABG$^+$13, BCP14] w.r.t. the transcripts of $\mathsf{GenStat}$ (i.e., Barak's preamble).

1.2 Towards Constant Round Concurrent Zero-knowledge

The simplest way to see why the protocol of previous section does not work in the concurrent setting is to consider its execution in a recursively interleaved schedule (described by Dwork, Naor, and Sahai [DNS98]).

In the context of our protocol, this schedule will have n sessions interleaved recursively as follows: session n does not "contain" any messages of any other session, and all messages of session i are contained between messages c_{i-1} and r_{i-1} of session $i-1$ for every i , starting from $i = n$. For completeness, this scheduling is shown in figure 1 (towards the end of the paper) with respect to 3 sessions. The double-headed arrows marked by π_i represent the rest of the messages of the i-th session. Roughly speaking, the simulation fails because of the following: in order to simulate session i, the simulator needs to extract the secret s_i by running the program $\widetilde{M}_{\lambda_i,s_i}$; however, the execution of $\widetilde{M}_{\lambda_i,s_i}$ contains an execution of $\widetilde{M}_{\lambda_{i+1},s_{i+1}}$ and due to this recursion, simulator's total running time in session 1 is exponential in n.

More formally, consider the scheduling given in figure 1. Let $t_3 \geq 1$ be the time taken by the verifier in computing r_3 on input the string c_3. Then clearly, the time taken by the simulator in running the obfuscated machine $\widetilde{M}_{\lambda_3,s_3}$ is $T_3 \geq t_3$. Then, if t_2 denotes the time taken by the simulator to obtain string r_2, we have that $t_2 \geq t_3 + T_3 \geq 2t_3$. Clearly, the time taken by the simulator to extract s_2 by running the program $\widetilde{M}_{\lambda_2,s_2}$ will be at least $T_2 \geq t_2 \geq 2t_3$. By repeating this argument for session 1, we have that $T_1 \geq t_1 \geq t_2 + T_2 \geq 2t_2 \geq 2^2 t_3$. Repeating this argument for n sessions in the DNS schedule, the total time taken by the simulator will be $\geq 2^{n-1}$.

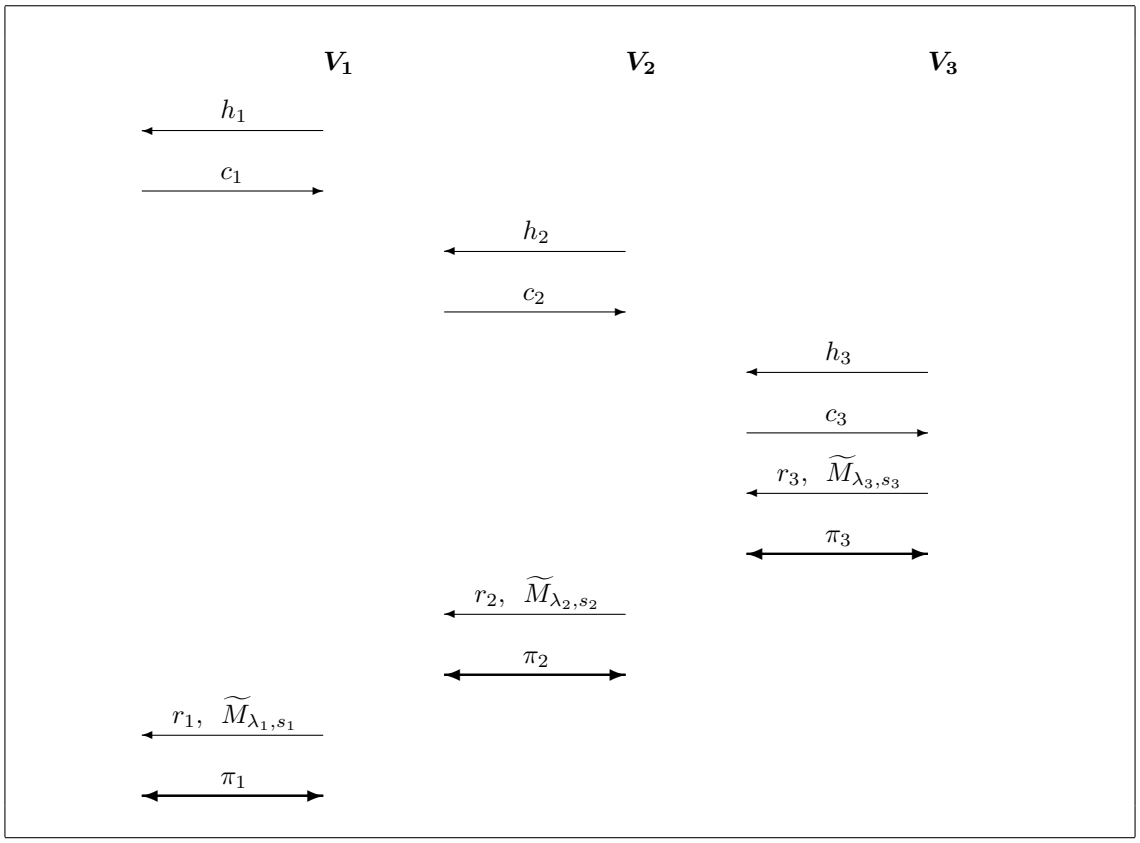

Figure 1: DNS scheduling for our protocol

Avoiding recursive computation via DGS-oracle. It is clear that the reason our stand-alone simulator runs in exponential time is because in order to compute s_i for session i, the simulator runs (the obfuscation

of) a program which recursively runs such a program for every interleaved session between c_i and r_i. That is, the program $\widetilde{M}_{\lambda_i,s_i}$ ends up *recomputing* all of the secrets of the interleaved sessions even though they have already been computed.

We can avoid this recomputation as follows. Let $\mathcal{I}$ be an oracle which takes as input queries of the form $(f, \widetilde{s})$—where f is an *injective* one-way function and $\widetilde{s}$ is in the range of f—and returns the unique value s such that $f(s) = \widetilde{s}$.[5] Now consider an arbitrary program $\Pi^{\mathcal{I}}$ which has access to the inversion oracle $\mathcal{I}$. Clearly, if r is chosen randomly, then for any (fixed) program $\Pi^{\mathcal{I}}$ and any fixed input a, the probability that $\Pi^{\mathcal{I}}(a) = r$ is at most 2^{-n}. This is because once the description of the oracle program $\Pi^{\langle\cdot\rangle}$ is fixed, the output of $\Pi^{\mathcal{I}}(a)$ is deterministically fixed (for any fixed input a chosen prior to seeing r) and r hits this value with probability at most 2^{-n}.

Our main point here is that it is hard to come up with a satisfying "fake witness" ω to the transcripts $\lambda = \langle h, c, r\rangle$ *even if* the program committed in c is given access to the inversion oracle $\mathcal{I}$. On the other hand, the simulator can still predict r as before. However, more importantly, by means of the oracle $\mathcal{I}$ we can avoid the recursive re-computation of the secrets in the concurrent setting as follows.

Consider an *alternative* simulator $S^{\langle\cdot\rangle}$ which will be given access to the oracle $\mathcal{I}$. This simulator will have access to both, the program of the verifier V^* as well as *its own program*, given as explicit inputs, collectively denoted as $\Pi^{\langle\cdot\rangle}_{S,V^*}$. The simulator, on input a session index i, will work by initiating an execution of V^*. It will commit to program $\Pi^{\langle\cdot\rangle}_{S,V^*}(j)$ in session j (ignoring for the moment the fact that simulator needs fresh randomness); finally, this simulator *does not run any obfuscated program* to compute the secrets. Instead it queries the oracle $\mathcal{I}$ on "well formed" $(f_j, \widetilde{s}_j)$ for every session $j \neq i$; when $j = i$ it simply returns the string r_i. Then, if all goes well, observe that program $\Pi^{\langle\cdot\rangle}(i)$ predicts string r_i in polynomial time (given $\mathcal{I}$) and this holds for every session i. In particular, there is no recursive recomputation of the secrets since they can be fed to the program directly once they have been computed. We note that such an oracle was first used by Deng, Goyal, and Sahai [DGS09] to construct the first *resettably-sound resettable zero-knowlege* protocol for $\mathbf{NP}$.

It should be clear that the actual simulation will be performed by a "main" simulator S_{main} which will *not have access to any* inversion oracle, and run in (strict) polynomial time. The main simulator will run in the same manner as the alternative simulator $S^{\langle\cdot\rangle}$ except that instead of using $\mathcal{I}$, it will run the obfuscated programs (only once for each session) to recover the secrets. To ensure efficient simulation, once a session secret has been recovered, it will be stored in a global table $\mathcal{T}$ (which will be used to simulate answers of $\mathcal{I}$). Therefore the "fake witness" will now have the form $\omega = \langle u, \Pi^{\langle\cdot\rangle}, \mathcal{T}\rangle$, but the statements will still have the same form $\lambda = \langle h, c, r\rangle$; and we require that $\Pi^{\mathcal{T}}$ outputs r within finite steps. These requirements will be formally captured by defining a relation $\mathbf{R}_{\text{sim}}$ w.r.t. the preamble $\mathsf{GenStat}$ in Section 4. We will discuss the overview of four round construction in Section 6.

1.3 Related Work

Concurrent zero-knowledge and non-black-box simulation. From early on, it was understood and explicitly proven in [FS90, GK96], that zero-knowledge is not preserved under parallel repetition where multiple sessions of the protocol run at the same time. The more complex notion of concurrent zero-knowledge (cZK) was introduced and achieved by Dwork, Naor, and Sahai [DNS98] (assuming "timing constraints" on the underlying network). A large body of research on cZK studied the round-complexity of black-box concurrent ZK with improving lower bounds on the same [KPR98, Ros00, CKPR03]. The state of art is the

[5] We assume that it is easy to test that f is injective and that $\widetilde{s}$ is in the range of f. These requirements are only for simplicity and the protocol works even if it is not easy to test these properties.

lower-bound is by Canetti, Kilian, Petrank, and Rosen [CKPR03] who prove that black-box cZK requires at least $O(\log n/\log\log n)$ rounds where n is the length of the statements being proven. Prabhakaran, Rosen, and Sahai [PRS02], building upon the prior works of Richardson and Kilian [RK99] and Kilian and Petrank [KP01], presented a cZK protocol for $\mathbf{NP}$ which has $\widetilde{O}(\log n)$ rounds, matching the lower bound of [CKPR03].

The central open question in this area is to construct a constant round cZK protocol for $\mathbf{NP}$ languages based on standard (or at least reasonable) assumptions. Barak [Bar01] showed that in the *bounded concurrent* setting where there is an a-priori upper bound on the number of sessions, there exists a constant round non-black-box cZK protocol for $\mathbf{NP}$; the protocol is based on the existence of collision-resistant hash functions [Bar01] and uses universal arguments [Kil92, Mic94, Kil95, BG02]. The communication complexity of Barak's protocol depends on the a-priori bound on the sessions.

It has proven difficult to extend Barak's NBB techniques to the setting of fully concurrent ZK (i.e., to unbounded polynomially many sessions) in $o(\log n)$ rounds. Nevertheless, NBB techniques have enjoyed great success resulting in the construction of resettable protocols [BGGL01, DL07, DGS09, GM11], non-malleable protocols [Bar02, PR05b, PR05a], leakage-resilient ZK [Pan14], bounded-concurrent secure computation [PR03, Pas04], adaptive security [GS12a], and so on. Bitanksy and Paneth [BP12a] showed that it is possible to perform non-black-box simulation using oblivious transfer (instead of collision-resistant hash functions and universal arguments). This eventually led to the construction of resettablly-sound ZK under one-way functions [BP13a, CPS13, COPV13]. Goyal [Goy13] presents a non-black-box simulation technique in the fully concurrent setting and achieves the first public-coin cZK protocol in the plain model.[6]

An alternative approach to construct round-efficient zero-knowledge proofs is to use "knowledge assumptions" [Dam91, HT99, BP04]. The recent work of Gupta and Sahai [GS12b] shows that such assumptions also yield a constant round concurrent ZK protocol for $\mathbf{NP}$. However, all known ZK protocols based on knowledge-type assumptions do not yield an *explicit* simulator. This is because the knowledge-type assumptions assume the existence of a special "extractor" machine (which is not explicitly known); this extractor is used by the simulator of ZK protocols and only provides an "existential" result.

Chung, Lin, and Pass [CLP13b] recently presented the first construction of a constant-round fully concurrent ZK protocol which has an explicit simulator. Their result is based on a new complexity-theoretic assumption, namely the existence of so called "strong $\mathbf{P}$-certificates."

Another alternative proposed in the literature is to assume some kind of a setup such as timing constraints, (untrusted) public-key infrastructure, and so on [DNS98, DS98, CGGM00, Dam00, Gol02, PTV10, GJO+13] or switch to super-polynomial time simulation [Pas03, PV08]. We will not consider such models further in this work.

Program obfuscation. After the strong impossibility results of [BGI+01], research in program obfuscation proceeded in two main directions. The first line of research focussed on constructing obfuscation for specific functionalities such as point functions and their variants, proxy re-encryption, encrypted signatures, hyperplanes, conjunctions, and so on[Wee05, LPS04, HRSV07, Had10, CRV10, BR13a]. The other line of research focussed on finding weaker definitions and alternative models. Goldwasser and Rothblum [GR07] considered the notion of *best possible obfuscation* (and is equivalent to iO when the obfusactor is polynomial time); and Bitansky and Canetti [BC10] considered *virtual grey box* security. Alternative models for obfuscation such as the hardware model were considered in [GIS+10, BCG+11].

After [GGH+13], an improved construction of iO was presented by Barak et. al. [BGK+13]. Further,

[6] The protocol requires $\mathsf{poly}(n)$ rounds. Canetti et al. [CLP13a] obtain a similar result, albeit in the "global hash" model where a global hash function—which the simulator cannot program—is known to all parties.

in an idealized "generic encodings" model it is shown that VBB-obfuscation for all circuits can be achieved [CV13, BR13b, BGK+13]. These results often involve a "bootstrapping step"; Applebaum [App13] presents an improved technique for bootstrapping obfuscation. Further complexity-theoretic results appear in recent works of Moran and Rosen [MR13], and Barak et. al. [BBC+14].

Sahai and Waters [SW13] show that indistinguishability obfuscation is a powerful tool and use it to successfully construct several (old and new) cryptographic primitives; further applications of iO appear in [HSW13, BZ13, BCP14, KRW13, MO13]

Differing input obfuscation was studied by Ananth et. al. [ABG+13], who present a candidate construction of diO for the class of polynomial time Turing machines and demonstrate new applications. Another variant of their construction allows the Turing machines to accept variable length inputs. Concurrent work of Boyle, Chung, and Pass [BCP14] introduces a related notion of *extractability obfuscation* and shows conditions under which this notion (and diO) are implied by iO. In addition, it also presents obfuscation for the class of polynomial time Turing machines, building upon the work of Brakerski and Rothblum [BR13a].

The issue of *auxiliary information* in program obfuscation was first considered by Goldwasser and Kalai [GK05], and further explored in [GK13, BCPR13, BP13b]. The work of Bitansky, Canetti, Paneth, and Rosen [BCPR13] shows that if iO exists then "extractability primitives" such as knowledge-types assumptions and extractable one-way functions [CD09] cannot exist in the presence of *arbitrary* auxiliary information. Boyle and Pass [BP13b] strengthen this result further by showing a pair of (universal) distributions $\mathcal{Z}$,$\mathcal{Z}$' on auxiliary information such that either extractable OWF w.r.t. $\mathcal{Z}$ do not exist or extractability-obfuscations w.r.t. $\mathcal{Z}$' do not exist.

2 Preliminaries

We use standard notations which are recalled here. This section can be skipped without affecting readability.

Notation. For a randomized algorithm A we write $A(x;r)$ the process of evaluating A on input x with random coins r. We write $A(x)$ the process of sampling a uniform r and then evaluating $A(x;r)$. We define $A(x,y;r)$ and $A(x,y)$ analogously. We denote by $\mathbb{N}$ and $\mathbb{R}$ the set of natural and real numbers respectively. The concatenation of two string a and b is denoted by $a \parallel b$.

We assume familiarity with interactive Turing machines (ITMs). For two randomized ITMs A and B, we denote by $[A(x,y) \leftrightarrow B(x,z)]$ the interactive computation between A and B, with A's inputs (x,y) and B's inputs (x,z), and uniform randomness; and $[A(x,y;r_A) \leftrightarrow B(x,z;r_B)]$ when we wish to specify randomness. We denote by $\mathsf{VIEW}_P[A(x,y) \leftrightarrow B(x,z)]$ and $\mathsf{OUT}_P[A(x,y) \leftrightarrow B(x,z)]$ the view and output of machine $P \in \{A,B\}$ in this computation. Finally, $\mathsf{TRANS}[A(x,y) \leftrightarrow B(x,z)]$ denotes the *transcript* of the interaction $[A(x,y) \leftrightarrow B(x,z)]$ which consists of all messages exchanged in the computation.

We also assume familiarity with *oracle* Turing machines, which are ordinary TMs with an extra tape called the *oracle communication tape*. An oracle TMs A will be written as $A^{(\cdot)}$ to insist that it is an oracle TM; in addition, we write $A^{\mathcal{I}}$ when A's oracle is fixed to $\mathcal{I}$. Recall that each query to $\mathcal{I}$ counts as one step towards the running time of $A^{\mathcal{I}}$.

Unless specified otherwise, all algorithms receive a parameter $n \in \mathbb{N}$, called the security parameter, as their first input. Often, the security parameter will not be mentioned explicitly and dropped from the notation. With some exceptions, all algorithms run in $\mathsf{poly}(n)$ steps and all inputs have $\mathsf{poly}(n)$ length. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if it approaches zero faster than every polynomial.

Two ensembles $\{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{Y}_n\}_{n\in\mathbb{N}}$ are said to be *computationally indistinguishable*, denoted $\{\mathcal{X}_n\} \stackrel{c}{\approx} \{\mathcal{Y}_n\}$, if for all non-uniform probabilistic polynomial time (PPT) distinguishers D, sufficiently large n, and every advice string z_n: $|\Pr_{x\leftarrow\mathcal{X}_n}[D_n(x)=1] - \Pr_{y\leftarrow\mathcal{Y}_n}[D_n(y)=1]| \leq \mathsf{negl}(n)$, where we write $D_n(a)$ to denoted $D(n, z_n, a)$, and negl is a negligible function. The statistical distance between two probability distributions $\mathcal{X}$ and $\mathcal{Y}$ over the same support S is denoted by $\Delta(X,Y) = \frac{1}{2}\sum_{a\in S} |\Pr[X=a] - \Pr[Y=a]|$. We say that ensembles $\{\mathcal{X}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{Y}_n\}_{n\in\mathbb{N}}$ are *statistically indistinguishable* (or statistically close), denoted $\{\mathcal{X}_n\} \stackrel{s}{\approx} \{\mathcal{Y}_n\}$, if there exists a negligible function negl such that $\Delta(\mathcal{X}_n, \mathcal{Y}_n) \leq \mathsf{negl}(n)$ for all sufficiently large n.

Standard primitives. In this work, we will be using a family of *injective* one-way functions. In addition, unless specified otherwise, we assume that all functions $f \in \mathcal{F}_n$ in the family have an *efficiently testable range membership*: i.e., there exists a polynomial time algorithm to test that $y \in \mathsf{Range}(f)$ where $\mathsf{Range}(f)$ denotes the range of f.

We will also be using a family of *collision resistant hash functions* (CRHF) $\{\mathcal{H}_n\}$ where $h : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(n)}$ for $h \in \mathcal{H}_n$; recall that $\{\mathcal{H}_n\}$ is a CRHF family if there exists a negligible function negl such that for every non-uniform PPT machines A, every sufficiently large n, and every advice string z_n: $\Pr_{h\leftarrow\mathcal{H}_n}[h(x) = h(y) : (x,y) \leftarrow A(z_n, h)] \leq \mathsf{negl}(n)$.

Finally, we will also be using a non-interactive, perfectly binding *commitment scheme* for committing strings of polynomial length. A commitment to a string m using randomness u will be denoted by $c = \mathsf{Com}(m; u)$. Without loss of generality, we assume that the message m committed to in c can be recovered given the randomness u and the string c. We assume perfectly binding schemes purely for the simplicity of exposition. One can replace Com by the 2-round statistically-binding commitment scheme of Naor [Nao89] without affecting our results.

2.1 Interactive Proofs, Proofs of Knowledge, and Witness Indistinguishability

We recall the standard definitions of interactive proofs [GMR85], witness indistinguishability [FS90], and proofs of knowledge [GMR85, TW87, FFS88, FS90, BG92, PR05b].

Definition 2.1 (Interactive Proofs)**.** A pair of probabilistic polynomial time interactive Turing machines $\langle P, V\rangle$ is called an interactive argument system for a language $\mathbf{L} \in \mathbf{NP}$ with witness relation $\mathbf{R}$ if there exists a negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ such that the following two conditions hold:

- *Completeness*: for every $x \in \mathbf{L}$, and every witness w such that $\mathbf{R}(x,w) = 1$, it holds that

$$\Pr[\mathsf{OUT}_V[P(x,w) \leftrightarrow V(x)] = 1] = 1.$$

- *Soundness*: for every $x \notin \mathbf{L}$, every interactive Turing machine P^* running in time at most $\mathsf{poly}(|x|)$, and every $y \in \{0,1\}^*$,

$$\Pr[\mathsf{OUT}_V[P^*(x,y) \leftrightarrow V(x)] = 1] \leq \mathsf{negl}(|x|).$$

If the soundness condition holds for every (not necessarily PPT) machine P^* then $\langle P, V\rangle$ is called an interactive *proof* system. □

The probability in the soundness condition is called the *soundness error* of the system, and we say that the system has *negligible* soundness error since this probability is at most $\mathsf{negl}(|x|)$. Although, traditionally

soundness error is defined in terms of the statement length $|x|$, in cryptographic contexts, it is convenient to define it in terms of the security parameter n, and write $\mathsf{negl}(n)$. This is without loss of generality, since in our setting since $|x| = \mathsf{poly}(n)$. Also, in this work, we will use words "argument" and "proof" interchangeably throughout the paper.

Definition 2.2 (Proof of Knowledge)**.** Let $\langle P, V\rangle$ be an interactive proof system for a language $\mathbf{L} \in \mathbf{NP}$ with witness relation $\mathbf{R}$. We say that $\langle P, V\rangle$ is a *proof of knowledge* (POK) for relation $\mathbf{R}$ if there exists a polynomial p and a probabilistic oracle machine E (called the *extractor*) such that for every PPT ITM P^*, there exists a negligible function negl such that for every $x \in \mathbf{L}$, and every $(y, r) \in \{0,1\}^*$ such that $q_{x,y,r} := \Pr[\mathsf{OUT}_V[P^*_{x,y,r} \leftrightarrow V(x)] = 1] > 0$ where $P^*_{x,y,r}$ denotes the machine P^* whose common input, auxiliary input, and randomness are fixed to x, y and r respectively and the probability is taken over the randomness of V, the following conditions holds:

- the expected number of steps taken by $E^{P^*_{x,y,r}}$ is bounded by $\frac{p(|x|)}{q_{x,y,r}}$, where $E^{P^*_{x,y,r}}$ is machine E with oracle access to $P^*_{x,y,r}$;
- except with negligible probability, $E^{P^*_{x,y,r}}$ outputs w^* such that $\mathbf{R}(x, w^*) = 1$. □

Definition 2.3 (Witness Indistinguishable Proofs)**.** Let $\langle P, V\rangle$ be an interactive proof system for a language $\mathbf{L} \in \mathbf{NP}$ with witness relation $\mathbf{R}$. We say that $\langle P, V\rangle$ is *witness indistinguishable* (WI) for relation $\mathbf{R}$ if for every PPT ITM V^*, every statement $x \in \mathbf{L}$, every pair of witnesses (w_1, w_2) such that $\mathbf{R}(x, w_i) = 1$ for every $i \in \{1, 2\}$, and every (advice) string $z \in \{0,1\}^*$, it holds that $\{\mathsf{VIEW}^{(1)}_{|x|}\} \stackrel{c}{\approx} \{\mathsf{VIEW}^{(2)}_{|x|}\}$ where $\{\mathsf{VIEW}^{(i)}_{|x|}\} := \mathsf{VIEW}_{V^*}[P(x, w_i) \leftrightarrow V^*(x, z)]$. □

As before, w.l.o.g., we can replace $|x|$ by the security parameter n in all definitions above. We remark that there exists a WIPOK with *strict* polynomial time extraction in *constant rounds* using non-black-box techniques [BL04] and in $\omega(1)$ rounds using black-box techniques [GMR85, Blu87].

Three round, public-coin WIPOK**and** ZAP**s.** The classical protocols of [GMR85, Blu87], based on the existence of non-interactive perfectly binding commitment schemes, are 3-round *witness indistinguishable, proof of knowledge* (WIPOK) protocols (for every language in $\mathbf{NP}$). We will use Blum's protocol [Blu87] as a building block and denote its three messages by $\langle \alpha, \beta, \gamma\rangle$, where β is random string of sufficient length.[7]

A ZAP for a language $\mathbf{L}$, introduced by Dwork and Naor [DN00], is a *two round witness indistinguishable* interactive proof for $\mathbf{L}$. ZAPs can be constructed from a variety of assumptions such as non-interactive zero-knowledge proofs [BFM88, BSMP91] (which in turn can be based on trapdoor permutations [FLS99]) and verifiable random functions [MRV99]. In fact, even *non-interarctive* (i.e., one round) constructions for ZAPs for all of $\mathbf{NP}$ exist based on bilinear pairings [GOS06] and derandomization techniques [BOV03].

We will use the two round construction of [DN00] based on NIZK as a building block and denote its two messages by $\langle \sigma, \pi\rangle$ where σ is a randomly string of sufficient length. An important property of this construction is *adaptive soundness*: the statement to be proven can be chosen *after* the string σ has been sent by the verifier. We will rely on this property in our security proofs.

[7] We remark that this protocol has a black-box extractor whose *expected* running time is proportional to the inverse of a cheating prover's success probability. However, there also exist WIPOK with *strict* polynomial time extraction in *constant rounds* using non-black-box techniques [BL04] and in $\omega(1)$ rounds using black-box techniques [GMR85, Blu87].

2.2 Concurrent Zero Knowledge

We now recall the notion of concurrent zero-knowledge [DNS98] in which one considers a "concurrent adversary" V^* who interacts in many copies of P, proving adaptively chosen, possibly correlated, polynomially many statements. We follow conventions established in [DNS98, PRS02, Ros04].

Concurrent attack. The *concurrent attack* on an interactive proof systems $\langle P, V\rangle$ for language $\mathbf{L} \in \mathbf{NP}$ with witness relation $\mathbf{R}$ considers an arbitrary interactive TM V^* which opens at most $m = m(n)$ sessions for an arbitrary polynomial m with arbitrary auxiliary input $z \in \{0,1\}^*$. Let $\vec{x} := \{x_i\} \in \mathbf{L}^m$ be set of statements in $\mathbf{L}$ of length at most $\mathsf{poly}(n)$, and $\vec{w} := \{w_i\}_{i\in[m]}$ be such that $\mathbf{R}(x_i, w_i) = 1$. The attack proceeds by uniformly fixing the random coins of V^* and initiating its execution on input the security parameter $n \in \mathbb{N}$ and auxiliary input z. At each step, V^* either initiates a new *session*—in which case a new prover instance $P(x_i, w_i)$ with fresh randomness is fixed who interacts with V^* in session i; or V^* schedules the delivery of a message of an existing session in which the corresponding prover instance responds with corresponding message. There is no restriction on how V^* schedules the messages of various sessions. We say that V^* *launches* m*-concurrent attack* on $\langle P, V\rangle$. The output of the attack consists of the view of V^*, denoted $\mathsf{VIEW}_{V^*}^{\langle P,V\rangle}(n, m, \vec{x}, \vec{w}, z)$.

Definition 2.4 (Concurrent Zero Knowledge)**.** We say that an interactive proof system $\langle P, V\rangle$ for a language $\mathbf{L} \in \mathbf{NP}$ (with witness relation $\mathbf{R}$) is *concurrent zero knowledge* if for every polynomial $m : \mathbb{N} \to \mathbb{N}$, every PPT ITM V^* launching a m-concurrent attack, there exists a PPT machine S_{V^*} such that for every set $\vec{x} := \{x_i\} \in \mathbf{L}^m$ of statements of length at most $\mathsf{poly}(n)$, every $\vec{w} := \{w_i\}_{i\in[m]}$ such that $\mathbf{R}(x_i, w_i) = 1$, and every auxiliary input $z \in \{0,1\}^*$ it holds that

$$\left\{S_{V^*}\big(n, \vec{x}, z\big)\right\}_{n\in\mathbb{N}} \stackrel{c}{\approx} \left\{\mathsf{VIEW}_{V^*}^{\langle P,V\rangle}\big(n, m, \vec{x}, \vec{w}, z\big)\right\}_{n\in\mathbb{N}}.$$

Machine S_{V^*} is called the *simulator*. □

In what follows, we will sometimes abuse the notation and write V^* to also mean the *description* of the Turing machine V^*. However, when we want to be explicit about the description of a Turing machine M (including V^*), we will actually write $\mathsf{desc}(M)$. For the simulator, we may sometimes write $S_{V^*}(\cdot) := S(V^*, \cdot)$ to insist that the program of V^* is given as an explicit input to the simulator (and drop n from the notation). Further, we will assume a (unique) session identifier for each session represented by a string of length n; this session identifier can be chosen by V^* so long as it is unique for every session. W.l.o.g. we assume that the all-ones string 1^n (not to be confused with the unary representation of the security parameter) is never used as a session identifier and denotes a special symbol.

3 Differing Input Obfuscation for Turing Machines

In this section, we recall the notion of differing input obfuscation (diO) for Turing machines. A weaker somewhat variant of this definition, namely *extractability obfuscation* ($e\mathcal{O}$) was defined and explored in [BCP14]. As mentioned earlier, we will actually need to work with what is called *distributional auxiliary infomration* diO (or $e\mathcal{O}$) where the auxiliary information will be sampled according a hard distribution, say $\mathcal{Z}$, over the statements of $\mathbf{L}^a_{\mathrm{sim}}$ for an arbitrary polynomial a. Recall that $\mathbf{L}^a_{\mathrm{sim}}$ is the language corresponding to relation $\mathbf{R}^a_{\mathrm{sim}}$ which are decidable in time at most $\mathsf{poly}(\alpha(n))$.

3.1 Definitions

Let $\mathsf{Steps}(M, x)$ denote the number of steps taken by a TM M on input x; we use the convention that if M does not halt on x then $\mathsf{Steps}(M, x)$ is defined to be the *special symbol* ∞. We define the notion of "compatible Turing machines" and "nice sampler." A pair of TMs (M_0, M_1) is said to be compatible if they have the same size, and more crucially, for every input x if M_0 halts on x then M_1 also halts on x in the *same* number of steps. I.e., for every x, $\mathsf{Steps}(M_0, x) = \mathsf{Steps}(M_1, x)$. We then consider sampling algorithms Samp which output a pair of compatible TMs (M_0, M_1), and say that Samp is "nice" if no PPT adversary A can produce an x such that: $M_0(x) \neq M_1(x)$ *and both* M_0, M_1 *halt within a polynomial number of steps* on input x. This requirement, or some variant of it, is necessary [ABG+13, BCP14].

Definition 3.1 (Compatible TMs)**.** A pair of Turing machines (M_0, M_1) is said to be *compatible* if $|M_0| = |M_1|$ and for every string $x \in \{0,1\}^*$ it holds that $\mathsf{Steps}(M_0, x) = \mathsf{Steps}(M_1, x)$.

By our convention, the second condition implies that M_0 halts on x *if and only if* M_1 halts on x.

Definition 3.2 (Nice TM Sampler)**.** We say that a (possibly non-uniform) PPT Turing machine Samp is a *nice sampler for Turing machines* if the following conditions hold:

1. the output of Samp is a triplet (z, M_0, M_1) such that (M_0, M_1) is *always* a pair of *compatible* TMs, and $z \in \{0,1\}^*$ is a string;
2. there exists a negligible function negl such that for every polynomial $a : \mathbb{N} \to \mathbb{N}$, every sufficiently large $n \in \mathbb{N}$, and every (possibly non-uniform) TM A running in time at most $a(n)$, it holds that:

$$\Pr\left[\begin{array}{l}(z, M_0, M_1) \leftarrow \mathsf{Samp}(1^n)\ ;\ A(z, M_0, M_1) = x;\\ \mathsf{Steps}(M_0, x) \leq a(n)\ ;\ M_0(x) \neq M_1(x).\end{array}\right] \leq \mathsf{negl}(n). \ \square$$

Some remarks are in order. First, note that since M_0, M_1 are always compatible and $\mathsf{Steps}(M_0, x) \leq a(n)$, we have that $\mathsf{Steps}(M_1, x) \leq a(n)$. Further, the "event" in the parentheses above can actually be tested in *polynomial* time. This is because every step defining this event can be performed in polynomial time. Finally, note that since the definition quantifies over all polynomials a, it allows A to produce *any* input x so long as M_0, M_1 halts on x within a polynomial number of steps.

The first output of Samp above will be used as auxiliary information in the definition below. We will denote the *distribution of first output* of Samp by $\mathcal{Z}$.

Differing input obfuscator. We now present the definition of a $\mathcal{Z}$*-auxiliary differing input obfuscator* for Turing machines. Roughly speaking, the notion states that a machine $\mathcal{O}$ is a $\mathcal{Z}$-auxiliary diO for (possibly non-uniform) efficiently samplable $\mathcal{Z} = \{\mathcal{Z}_n\}$ if the following holds: if there exists a PPT distinguisher D who distinguishes $\mathcal{O}(M_0)$ from $\mathcal{O}(M_1)$ when given auxiliary input $z \leftarrow \mathcal{Z}_n$, then it is easy to find an x (given z) such that $M_0(x) \neq M_1(x)$. In other words, if it is hard to find the "differing input" x then the two obfuscations are indistinguishable.

We now present the definition below, following [ABG+13]. We note that since we want to be explicit about the distribution of the auxiliary information (the first output of the sampling algorithm Samp), we will denote it by $\mathcal{Z}$.

Definition 3.3 ($\mathcal{Z}$-auxiliary Differing Input Obfuscator for Turing Machines)**.** A uniform PPT machine $\mathcal{O}$ is called a *differing input obfuscator* for a class of Turing machines $\{\mathcal{M}_n\}$ if the following conditions are satisfied:

1. *Polynomial slowdown and functionality:* there exists a polynomial a_{dio} such that for every $n \in \mathbb{N}$, every $M \in \mathcal{M}_n$, every input x such that M halts on x, and every $\widetilde{M} \leftarrow \mathcal{O}(n, M)$, the following conditions hold:

   - $\mathsf{Steps}(\widetilde{M}, x) \leq a_{\text{dio}}\Big(n,\ \mathsf{Steps}(M, x)\Big)$
   - $\widetilde{M}(x) = M(x)$

   Polynomial a_{dio} is called the *slowdown polynomial* of $\mathcal{O}$.

2. *Indistinguishability:* for every *nice* sampler Samp (i.e., satisfying definition 3.2) *whose first output is distributed according to* $\mathcal{Z}$, there exists a negligible function negl such that for every polynomial $a : \mathbb{N} \to \mathbb{N}$, every sufficiently large $n \in \mathbb{N}$, and every (possibly non-uniform) TM D running in time at most $a(n)$, it holds that:

$$\Big|\Pr\left[D\left(z, \mathcal{O}(n, M_0)\right) = 1 : (z, M_0, M_1) \leftarrow \mathsf{Samp}\left(1^n\right)\right] - \Pr\left[D\left(z, \mathcal{O}(n, M_1)\right) = 1 : (z, M_0, M_1) \leftarrow \mathsf{Samp}\left(1^n\right)\right]\Big| \ \leq\ \mathsf{negl}(n).$$

   Machine D is called the *distinguisher*. □

3.2 Candidate Constructions

As noted earlier, our reduction requires the existence of $\mathcal{Z}$-auxiliary diO for the class of all polynomial size Turing machines which accept inputs of arbitrary polynomial length (in n) and halt within polynomial steps with respect to all (possibly nonuniform) efficiently samplable distributions $\mathcal{Z}$ that are *hard over the statements of* $\mathbf{L}^a_{\text{sim}}$ for every polynomial a. A candidate construction for this primitive appears in the work of [ABG+13]. Their construction is based on diO of the class of all polynomial-size circuits (constructed in [GGH+13]), fully homomorphic encryption (e.g., [Gen09, BV11], and SNARKs [BCCT13] (which require knowledge-type assumptions). If an a-priori bound on the input is known, then comparatively better constructions are possible [ABG+13, BCP14].

Our requirements from obfuscation are actually weaker than stated above. We do not need obfuscation for the class of *all* (polynomial size and running time) Turing machines; instead we only require the obfuscation of the machine $\mathsf{SimLock}$ which (receive inputs of arbitrary, a-priori unknown, polynomial length and) halt within a polynomial number of steps. We also do not need security w.r.t. every hard distribution $\mathcal{Z}$ over $\mathbf{L}^a_{\text{sim}}$; instead, we only need to assume that it holds for the statements λ that are transcripts of the $\mathsf{GenStat}$ protocol (with an arbitrary cheating prover P^*_1). Interestingly, this kind of advice can be simulated using the distribution $\mathcal{Z}^*$ that simply outputs (h, r); therefore distribution $\mathcal{Z}'$ can actually be *uniform distribution* if h is a "public-coin" CRHF [HR04] making it a more plausible assumption.

As we have to come to learn [GK05, GK13, BCPR13, BP13b], security w.r.t. arbitrary auxiliary inputs might be too strong an assumption. Bitansky, Canetti, Paneth, and Rosen [BCPR13] show that either indistinguishability obfuscation does not exist for all circuits or for every OWF-family $\mathcal{F}$ there exists an auxiliary input distribution $\mathcal{Z}_{\mathcal{F}}$ w.r.t. which $\mathcal{F}$ is not an extractable OWF family [CD09]. Boyle and Pass [BP13b] further strengthen this result by showing a pair of distributions $\mathcal{Z}$,$\mathcal{Z}$' such that either extractable OWFs do not exist w.r.t. $\mathcal{Z}$ or diO for (the class of all PPT) Turing machines does not exist w.r.t. $\mathcal{Z}$'. Thus at least one of these assumptions must fall. However, as they further note, it does not invalidate assumptions for other distributions $\mathcal{Z}''$ (in particular when $\mathcal{Z}''$ can be uniform). These negative results also do not

necessarily contradict the conjectured security of candidate construction of [ABG+13] w.r.t. the auxiliary input distributions we need (namely transcripts of $\mathsf{GenStat}$ or $\mathcal{Z}^*$ mentioned above). Nevertheless, we hope that candidate constructions based on better complexity-theoretic assumptions will be discovered for this primitive in the future.

4 Relation $\mathbf{R}_{\mathrm{sim}}$ and A Nice Sampler

In this section, we define the preamble $\mathsf{GenStat}$, relations $\mathbf{R}_{\mathrm{sim}}$, $\mathbf{R}^a_{\mathrm{sim}}$, and prove that a randomly sampled transcript of $\mathsf{GenStat}$ is a hard distribution over the statements of language $\mathbf{L}_{\mathrm{sim}}$ (corresponding to relation $\mathbf{R}_{\mathrm{sim}}$). For convenience, we use a non-interactive perfectly binding commitment scheme; the two-round statistically-binding commitment scheme of [Nao89] also works.

4.1 Preamble $\mathsf{GenStat}$

Statement generation protocol. Let $\{\mathcal{H}_n\}$ be a family of collision-resistant hash functions (CRHF) $h \in \mathcal{H}_n$ such that $h : \{0,1\}^* \to \{0,1\}^n$ and Com be a non-interactive perfectly-binding commitment scheme for $\{0,1\}^n$. The statement generation protocol $\mathsf{GenStat} := \langle P_1, V_1 \rangle$ is a three round protocol between P_1 and V_1 which proceeds as follows:

Protocol $\mathsf{GenStat} := \langle P_1, V_1 \rangle$**:**

1. V_1 sends a random $h \leftarrow \mathcal{H}_n$
2. P_1 sends a commitment $c = \mathsf{Com}(0^n; u)$ where u is a randomly chosen
3. V_1 sends a random string $r \leftarrow \{0,1\}^n$

The transcript of the protocol is $\lambda := \langle h, c, r \rangle$. $\square$

4.2 Relation $\mathbf{R}_{\mathrm{sim}}$

We now define the relation $\mathbf{R}_{\mathrm{sim}}$. Let $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ and $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ denote the family of injective one-way functions and collision-resistant hash functions respectively. Let Com be a non-interactive, perfectly binding (string) commitment scheme. The relation is formally defined in figure 2. The statements for the relation $\mathbf{R}_{\mathrm{sim}}$ are the transcripts $\lambda := (h, c, r)$ and the witnesses are of the form $\omega := (u, \Pi^{\langle\cdot\rangle}, \mathcal{T})$ such that c is a commitment to the oracle-TM $\Pi^{\langle\cdot\rangle}$ using randomness u, $\mathcal{T}$ is a table containing answers to all inversion queries that $\Pi^{\langle\cdot\rangle}$ makes (for functions $f \in \mathcal{F}_n$), and $\Pi^{\mathcal{T}}$ outputs r.[8]

An important observation regrading $\mathbf{R}_{\mathrm{sim}}$ is that since table $\mathcal{T}$ is not a part of the commitment c (and it should not be), we must enforce that $\Pi^{\langle\cdot\rangle}$ *never makes any invalid queries to* $\mathcal{T}$. This is because after seeing r, it is easy to design a "bad" table $\mathcal{T}$ which will encode r by means of "bad" entries and "satisy" λ.

Relation $\mathbf{R}_{\mathrm{sim}}$ is undecidable in general. For convenience, we define a decidable, polynomial time, version of $\mathbf{R}_{\mathrm{sim}}$, denoted by $\mathbf{R}^a_{\mathrm{sim}}$ where $a : \mathbb{N} \to \mathbb{N}$ is a polynomial, as follows.

Relation $\mathbf{R}^a_{\mathrm{sim}}$ and language $\mathbf{L}^a_{\mathrm{sim}}$: Let $a : \mathbb{N} \to \mathbb{N}$ be a polynomial; relation $\mathbf{R}^a_{\mathrm{sim}}$ is identical to $\mathbf{R}_{\mathrm{sim}}$ except that the witness $(u, \Pi^{\langle\cdot\rangle}, \mathcal{T})$ satisfies following additional constraints:

[8] For simplicity, we assume that it easy to test the that functions f are injective and whether a given element is in the range of f for every $f \in \{\mathcal{F}_n\}$. Note that n is implicit in the definition of $\mathbf{R}_{\mathrm{sim}}$ and can, for example, be obtained from the description of λ—in particular, length of r or description of h.

Instance: A tuple $\langle h, c, r\rangle \in \mathcal{H}_n \times \{0,1\}^{\mathsf{poly}(n)} \times \{0,1\}^n$ where $h : \{0,1\}^* \to \{0,1\}^n$.

Witness: A tuple $\langle u, \Pi^{\langle\cdot\rangle}, \mathcal{T}\rangle \in \{0,1\}^{\mathsf{poly}(n)} \times \{0,1\}^* \times \{0,1\}^*$ where $\Pi^{\langle\cdot\rangle}$ is an *oracle* Turing machine, and $\mathcal{T}$ is a table containing entries of the form $(f, \widetilde{s}, s)$ such that when queried on $(f, \widetilde{s})$, $\mathcal{T}$ returns s, denoted $\mathcal{T}(f, \widetilde{s}) = s$.

Relation: $\mathbf{R}_{\text{sim}}\big(\langle h, c, r\rangle,\ \langle u, \Pi^{\langle\cdot\rangle}, \mathcal{T}\rangle\big) = 1$ if and only if all of the following conditions hold:

1. $c = \mathsf{Com}\left(h\left(\Pi^{\langle\cdot\rangle}\right)\ ;\ u\right)$
2. $\forall\ (f, \widetilde{s}, s) \in \mathcal{T}$ it holds that $f \in \mathcal{F}_n$ is an *injective* function and $f(s) = \widetilde{s}$
3. Program $\Pi^{\mathcal{T}}$, takes no input, outputs the string r, and halts
4. Program $\Pi^{\mathcal{T}}$ makes oracle queries of the form $(f, \widetilde{s})$ such that:

$$\forall \text{ queries } (f, \widetilde{s})\ \ \exists\, s \ \text{ s.t. } \ (f, \widetilde{s}, s) \in \mathcal{T}$$

Figure 2: Relation $\mathbf{R}_{\text{sim}}$ based on a perfectly binding commitment Com.

1. $\left|\mathcal{T}\right| \ \leq\ a(n)$,
2. $\Pi^{\mathcal{T}}$ halts in at most $a(n)$ steps.

Let $\mathbf{L}_{\text{sim}}$ (resp. $\mathbf{L}^a_{\text{sim}}$) be the language corresponding to relation $\mathbf{R}_{\text{sim}}$ (resp., $\mathbf{R}^a_{\text{sim}}$). Note that $\mathbf{L}^a_{\text{sim}} \in \mathbf{NP}$. Note that $\mathbf{R}^a_{\text{sim}}$ can be tested in time $\mathsf{poly}(a(n)) = \mathsf{poly}(n)$.

Hard distributions over $\mathbf{L}^a_{\text{sim}}$. We say that $\mathcal{Z} = \{\mathcal{Z}_n\}$ is a *hard distribution* over the statements of $\mathbf{L}^a_{\text{sim}}$ if there exists a negligible function negl such that for every non-uniform PPT algorithm A^* and every sufficiently large n it holds that

$$\Pr[\lambda \leftarrow \mathcal{Z}_n; \omega \leftarrow A^*(1^n, \lambda); \mathbf{R}^a_{\text{sim}}(\lambda, \omega) = 1] \leq \mathsf{negl}(n).$$

The following lemma states that the transcripts of $\mathsf{GenStat}$ form a hard distribution over $\mathbf{L}_{\text{sim}}$. That is, it is hard for any PPT machine P^*_1 to compute a witness ω to statements λ when λ is the transcript of $\mathsf{GenStat}$ between P^*_1 and an honest V_1. The proof follows [Bar01].

Lemma 4.1 (Hardness of $\mathsf{GenStat}$)**.** *Assume that $\{\mathcal{H}_n\}$ is a family of collision-resistant hash functions against (non-uniform) PPT algorithms. There exists a negligible function negl such that for every (non-uniform) PPT Turing machine P^*_1 , the probability that P^*_1, after interacting with an honest V_1 in protocol $\mathsf{GenStat}$, writes a string ω on its (private) output tape such that $\mathbf{R}_{\text{sim}}(\lambda, \omega) = 1$ is at most $\mathsf{negl}(n)$, where λ is the transcript of interaction between P^*_1 and V_1, and the probability is taken over the randomness of both P^*_1 and V_1.*

Remark. We note that, by definition of $\mathbf{R}_{\text{sim}}$, expression $\mathbf{R}_{\text{sim}}(\lambda, \omega) = 1$ is only defined for those tuples (λ, ω) for which $\mathbf{R}_{\text{sim}}$ is decidable (not necessarily in polynomial time). Therefore, the statement of the lemma, in particular, means that the prover *must* write a string ω for which it is possible to test (not necessarily in polynomial times) that $\mathbf{R}_{\text{sim}}(\lambda, \omega) = 1$. We can restate the lemma in terms of relations $\mathbf{R}^a_{\text{sim}}$ for

every polynomial a, but as we shall see in the proof, a does not actually play a role. Therefore, we have chosen to avoid the use of a and directly state the lemma in terms of $\mathbf{R}_{\mathrm{sim}}$.

Proof of lemma 4.1. Assume, on the contrary, that there exist polynomials p, q and a prover P_1^* such that P_1^* takes at most $p(n)$ steps and writes a string ω on its private output tape such that for infinitely many values of n, $\delta(n) \geq 1/q(n)$ where $\delta(n)$ is the probability that $\mathbf{R}_{\mathrm{sim}}(\lambda, \omega) = 1$ (where λ is sampled as defined in the lemma). Now consider the machine P_1^* in an execution of GenStat and let (h, c) be the first two messages in this interaction. Let the machine $P^*_{1,h,c}$ denote the machine P_1^* whose state has been frozen up to the point where c is sent in this execution. By a standard averaging argument, it follows that with probability at least $\delta/2$ over the sampling of (h, c) in this interaction, the probability that $P^*_{1,h,c}$ writes a valid witness ω at the end of the interaction is at least $\delta/2$. We call such (h, c) "good."

The following procedure finds collisions in h provided (h, c) are good: the procedure chooses two random strings r_1, r_2 each of length n, feeds P_1^* with r_1 and then with r_2 separately; let $\omega_i = (u_i, \Pi_i^{\langle \cdot, \cdot \rangle}, \mathcal{T}_i)$ be the contents of the private output tape of $P^*_{1,h,c}$ when fed with string r_i for $i \in \{1, 2\}$. The procedure outputs (Π_1, Π_2) as the potential collision on h.

We claim that the procedure finds collisions in h with noticeable probability as follows. Note that since (h, c) is good, with probability $\delta^2/4$, it holds that $\mathbf{R}_{\mathrm{sim}}(\lambda_i, \omega_i) = 1$ where $\lambda_i = (h, c, r_i)$. Hence, $\Pi_i^{\mathcal{T}_i} = r_i$ and $h(\Pi_1) = h(\Pi_2)$ w.h.p. since c is perfectly binding.

Now, define $\mathcal{I}$ to be an inversion oracle which on input a query of the form $(f, \widetilde{s})$ for $f \in \mathcal{F}_n$ and $\widetilde{s} \in \mathrm{Range}(f)$ outputs $s = f^{-1}(\widetilde{s})$. Then, by definition of $\mathbf{R}_{\mathrm{sim}}$ (in particular, due to condition 4 in figure 2), we have that the output of $\Pi_i^{\mathcal{T}_i}$ is the same as that of $\Pi_i^{\mathcal{I}}$. I.e., $\Pi_i^{\mathcal{I}}$ outputs r_i. Since $\Pi_i^{\mathcal{I}}$ is a *deterministic* computation, it holds that Π_1 and Π_2 are different programs whenever $r_1 \neq r_2$ (which happens with prob. $1 - 2^{-n}$). Further, since P_1^* runs in time at most $p(n)$, programs Π_1, Π_2 are of size at most $p(n)$. Therefore, Π_1 and Π_2 are collisions in h, found with probability at least $\frac{\delta^2}{4} \cdot (1 - 2^{-n}) \geq \delta^2/8$.

It follows that collisions can be found for a noticeable (specifically, at least $\delta/2$) fraction of functions in $\{\mathcal{H}_n\}$ with noticeable probability (specifically, $\delta^2/8$). This concludes the proof. ■

4.3 A Nice Sampler for TM

Protocol GenStat allows us to build a (non uniform) sampling algorithm Samp which will be nice according to definition 3.2. In addition, the distribution of first output of Samp will give us a hard distribution over the statement of $\mathbf{L}^a_{\mathrm{sim}}$ for every polynomial a.

Samp uses the following simple TM:[9]

SimLock(λ, ω, s):
Test if $\mathbf{R}_{\mathrm{sim}}(\lambda, \omega) = 1$, and if so output s;
Else, output the empty string 0^n.

Also, for a fixed (λ, s), define $\mathsf{SimLock}_{\lambda,s}(\cdot) := \mathsf{SimLock}(\lambda, \cdot, s)$. Machine $\mathsf{SimLock}_{\lambda,s}$ essentially tests whether the input is a valid witness to λ, and if so outputs the fixed value s, and nothing otherwise. Note that it is possible that SimLock does not halt on some inputs. Also, w.l.o.g., we assume that $\mathsf{Steps}(\mathsf{SimLock}_{\lambda,s_1}, \omega) = \mathsf{Steps}(\mathsf{SimLock}_{\lambda,s_2}, \omega)$ for every λ, ω and $(s_1, s_2)\{0,1\}^n \times \{0,1\}^n$.

[9] SimLock stands for "simulator's lock," i.e., only the simulator (having access to the code of the verifier) will be able to "unlock" the secret s. We note that when we write "Test if $\mathbf{R}_{\mathrm{sim}}(\lambda, \omega) = 1$", it means that the relation $\mathbf{R}_{\mathrm{sim}}$ is tested using a Turing machine. If this test does not halt, then SimLock also does not halt.

The sampler. The sampling algorithm, Samp_{s,P_1^*} is defined with respect to a string $s \in \{0,1\}^n$ and an arbitrary PPT interactive TM P_1^*. ITM P_1^* follows the instructions of GenStat protocol and interacts with algorithm V_1. The distribution of first output of Samp_{s,P_1^*} is independent of s, and captured by separately defining distribution $\mathcal{Z}_{P_1^*} := \{\mathcal{Z}_{n,P_1^*}\}$

$\mathsf{Samp}_{s,P_1^*}(1^n)$**.**

- Sample a random transcript λ of GenStat by interacting with P_1^* honestly according to V_1
- Output $\left(\lambda,\ \mathsf{SimLock}_{\lambda,s},\ \mathsf{SimLock}_{\lambda,0^n}\right)$

Distribution $\mathcal{Z}_{n,P_1^*}$**.**

- Output a randomly sampled transcript λ of GenStat, obtained by honestly interacting with P_1^*.

Note that the first output of $\mathsf{Samp}_{s,P_1^*}(1^n)$ and $\mathcal{Z}_{n,P*_1}$ are distributed identically for every (n,s). The following lemma is essentially a corollary of lemma 4.1.

Lemma 4.2. *For every non-uniform* PPT TM P_1^**, and every* $s \in \{0,1\}^n$*,* Samp_{s,P_1^*} *is a nice sampler for Turing machines (according to definition 3.2). Further, for every polynomial* $a : \mathbb{N} \to \mathbb{N}$*,* $\mathcal{Z}_{n,P_1^*}$ *is a hard distribution over* $\mathbf{L}^a_{\text{sim}}$.

Proof. Observe that the pair $(\mathsf{SimLock}_{\lambda,s},\ \mathsf{SimLock}_{\lambda,0^n})$ is *always* a pair of compatible TMs, by definition of SimLock. Now suppose that the second property of definition 3.2 is not satisfied. Then there exists an A, running in time at most $a(n)$ for some polynomial a, which outputs an x with noticeable probability such that $\mathsf{SimLock}_{\lambda,s}(x) \neq \mathsf{SimLock}_{\lambda,0^n}(x)$, and $\mathsf{Steps}(\mathsf{SimLock}_{\lambda,s}, x) \leq a(n)$; here the probability is taken over the sampling of λ (which in turn is distributed according to $\mathcal{Z}_{n,P_1^*}$). It follows, from the definition of $\mathsf{SimLock}_{\lambda,s}$, that x must be a witness to λ and therefore A is a PPT machine which finds witnesses to statements $\lambda \in \mathbf{L}^a_{\text{sim}}$ with noticeable probability. We can use A to violate lemma 4.1 as follows.

Consider the machine $B^*_{1,s}$ which incorporates P_1^* and A. It then samples λ by routing messages between P_1^* and an external (honest) V_1, and returns the output of $A\left(\lambda, \mathsf{SimLock}_{\lambda,s}, \mathsf{SimLock}_{\lambda,0^n}\right)$. It is straightforward to see that $B^*_{1,s}$ violates lemma 4.1 (for every fixed s). Further, λ is distributed according $\mathcal{Z}_{n,P_1^*}$ and a is an arbitrary polynomial, this also proves the second part of the lemma. ■

5 A Simpler Variant of Our Protocol

In this section, we describe the simpler version of our protocol, namely Simple-c$\mathcal{ZK}$; it is a (fully) concurrent zero-knowledge protocol in constant (but not four) rounds. Let P denote the prover algorithm and V denote the verifier algorithm. Informally, the protocol has three stages:

1. In stage 1, P and V sample a statement $\lambda = (h, c, r)$ for the relation $\mathbf{R}_{\text{sim}}$ using the protocol GenStat.
2. In stage 2, V sends the image $\widetilde{s}$ of a randomly chosen input s under an injective OWF f and also sends f. Additionally,
   (a) V also sends an obfuscation of the machine $\mathsf{SimLock}_{\lambda,s}$ which outputs s on every input ω for which $\mathbf{R}_{\text{sim}}(\lambda, \omega) = 1$.
   (b) V proves, using a ZK proof, that it computed all values in this stage honestly.

3. In stage 3, P proves that either it knows a witness w to x or it knows the pre-image $f^{-1}(\widetilde{s})$.

The formal description of protocol Simple-c$\mathcal{ZK}$ appears in figure 3. The main result of this section is the following theorem.

Theorem 5.1. *Assume the existence of collision-resistant hash functions and injective one-way functions. Further, for every polynomial $a : \mathbb{N} \to \mathbb{N}$, and every* hard distribution *$\mathcal{Z}$ over the statements of $\mathbf{L}^a_{\text{sim}}$, assume the existence of $\mathcal{Z}$-*auxiliary differing-input obfuscation *(diO) for the class of all polynomial-size Turing machines that halt in a polynomial number of steps.*[10] *Then, there exists a constant round, fully concurrent zero-knowledge protocol with negligible soundness, for all languages in* $\mathbf{NP}$.

We prove the above theorem by proving that protocol Simple-c$\mathcal{ZK}$ is a fully concurrent zero-knowledge protocol with negligible soundness error (Theorem 5.6). It is clear that the protocol has constant rounds and perfect completeness. The soundness and concurrent-ZK properties of this protocol are proven in next two sections.

Inputs. The common input to P and V is a statement $x \in \mathbf{L}$ where language $\mathbf{L} \in \mathbf{NP}$. The prover's auxiliary input is a witness w such that $\mathbf{R}(x, w) = 1$. The security parameter n is an implicit input to both parties.

Protocol. The protocol proceeds in three stages.

Stage 1: P and V execute the GenStat protocol in which V sends the first message $h \leftarrow \mathcal{H}_n$, P sends the second message $c = \mathsf{Com}(0^n; u)$ for a random u, and V sends the final message $r \leftarrow \{0,1\}^n$. Let $\lambda = \langle h, c, r \rangle$ be the transcript.

Stage 2: V samples an injective one-way functions $f \leftarrow \mathcal{F}_n$, a random input $s \in \{0,1\}^n$, and a sufficiently long random tape $\zeta \in \{0,1\}^{\mathsf{poly}(n)}$ and computes:

$$\widetilde{s} = f(s), \quad \widetilde{M}_{\lambda,s} \leftarrow \mathcal{O}\left(\mathsf{SimLock}_{\lambda,s}\,;\,\zeta\right) \tag{5.1}$$

V sends $(f, \widetilde{s}, \widetilde{M}_{\lambda,s})$, and proves using a constant round ZK protocol (say Π_{ZK}) that there exist (s, ζ) satisfying equation (5.1) above.

Stage 3: P proves to V, using a 3-round WIPOK (say Π_{WIPOK}) the knowledge of either:

- w such that $\mathbf{R}(x, w) = 1$; OR
- s such that $f(s) = \widetilde{s}$.

***Verifier's output*:** V accepts if the proof in stage 3 succeeds; otherwise, it rejects.

Figure 3: The simpler variant of our protocol: Simple-c$\mathcal{ZK}$.

5.1 Soundness

Lemma 5.2. Simple-c$\mathcal{ZK}$ *has negligible soundness error.*

[10] We note that we actually do not need obfuscation for the class of all PPT Turing machines. Instead, we only need obfuscation for those Turing machines of the form $\mathsf{SimLock}^a$ where a is a polynomial and $\mathsf{SimLock}^a$ is the same as SimLock except that it runs for at most $a(|x|)$ steps on input x.

Proof. Let P^* be a non-uniform cheating prover who succeeds in proving a false statement $x \notin \mathbf{L}$ with some non-negligible probability. There are two parts of the proof:

- first part shows that P^* cannot compute the secret s with noticeable probability,
- second part shows that if $x \notin \mathbf{L}$ and P^* convinces V with noticeable probability, then it can be used to compute s with noticeable probability—violating the first part.

We start with the first part of the proof. Let $P^*_{x,z,\rho}$ denote the prover algorithm P^* with non-uniform advice z and random tape fixed to ρ. Further define the following two machines:

Machine $P^*_{1,(x,z,\rho)}$: This machine is identical to $P^*_{x,z,\rho}$ except that it only executes *stage 1* of the protocol, i.e., the $\mathsf{GenStat}$ part, aborts the rest of the execution and halts. The transcripts of this machine's interactions are of the form $\lambda = (h, c, r)$.

Machine $P^*_{2,(x,z,\rho)}$: This machine is identical to $P^*_{x,z,\rho}$ except that it only executes first two stages of the protocol, namely *stage 1* and *stage 2*, aborts the rest of the execution and halts. The transcripts of this machine's interaction contain $\lambda = \langle h, c, r\rangle$, $(f, \widetilde{s}, \widetilde{M}_{\lambda,s})$), and the transcript of the ZK protocol.

Observe that the sampler $\mathsf{Samp}_{s,P^*_{1,(x,z,\rho)}}$, defined in section 4.3, is a well defined machine for every s with respect to our first algorithm $P^*_{1,(x,z,\rho)}$. Further, it is a nice sampler due to lemma 4.2. We now prove that $P^*_{2,(x,z,\rho)}$ cannot learn the inverse of $\widetilde{s}$. That is,

Claim 5.3. *The probability that $P^*_{2,(x,z,\rho)}$, after an interacting with the honest verifier V in protocol* Simple-c$\mathcal{ZK}$*, writes a string $s \in \{0,1\}^n$ on its private output tape such that*

$$\widetilde{s} = f(s)$$

is at most $\mathsf{negl}(n)$ *where $(\lambda,\ \ (f, \widetilde{s}, \widetilde{M}_{\lambda,s})\ )$ is the (partial) transcript of the interaction and the probability is taken over the randomness of V.*

Proof. Assume on the contrary that $P^*_{2,(x,z,\rho)}$ does write a string s satisfying the lemma with non-negligible probability $\delta = \delta(n)$. We show how to use this machine to invert the injective function f in polynomial time. We start by consider the following machine $B^*_{2,(x,z,\rho)}$ which takes no input.

Machine $B^*_{2,(x,z,\rho)}$:

1. The machine incorporates $P^*_{2,(x,z,\rho)}$ and interacts with it by playing the role of V honestly until the end of stage 1. Let λ be the transcript of this stage.
2. At the start of stage 2, $B^*_{2,(x,z,\rho)}$ generates $(f, \widetilde{s}, \widetilde{M}_{\lambda,s})$ honestly.
3. $B^*_{2,(x,z,\rho)}$ employs the simulator of the ZK protocol and samples a a view for $B^*_{2,(x,z,\rho)}$.
4. At the end of simulation, $B^*_{2,(x,z,\rho)}$ outputs the contents of $P^*_{2,(x,z,\rho)}$'s private output tape.

By construction, the view of $P^*_{2,(x,z,\rho)}$ is simulated perfectly by $B^*_{2,(x,z,\rho)}$ until the the ZK protocol begins. Therefore, from the properties of the ZK-simulator, it holds that the view of $P^*_{2,(x,z,\rho)}$ at the end of the simulation is computationally indistinguishable from its view in a real execution with V. It follows that the outputs of $B^*_{2,(x,z,\rho)}$ is a string s such that $f(s) = \widetilde{s}$ with probability $\delta' \geq \delta - \mathsf{negl}(n)$.

To build the inverter for f, the next step is to slightly modify $B^*_{2,(x,z,\rho)}$: instead of computing the obfuscated TM $\widetilde{M}_{\lambda,s}$ honestly, the new machine $B^{**}_{2,(x,z,\rho)}$ sends the machine which instead outputs 0^n. I.e.,

Machine $B^{**}_{2,(x,z,\rho)}$: This machine is identical to $B^{*}_{2,(x,z,\rho)}$ except at the start of stage 1 it sends $(f, \widetilde{s}, \widetilde{M}_{\lambda,0^n})$ where:

$$\widetilde{M}_{\lambda,0^n} \leftarrow \mathcal{O}\left(\mathsf{SimLock}_{\lambda,0^n}\ ;\ \zeta\right)$$

always outputs 0^n on all inputs and λ is the transcript of stage 1.

We claim that $B^{**}_{2,(x,z,\rho)}$ outputs s such that $f(s) = \widetilde{s}$ with probability $\delta'' \geq \delta' - \mathsf{negl}(n)$. To prove this, we construct a hybrid machine H^*_2. Machine H^*_2 violates the indistinguishability property of the obfuscator $\mathcal{O}$ with respect to the nice sampler $\mathsf{Samp}_{s,P^*_{1,(x,z,\rho)}}$ for every $s \in \{0,1\}^n$.

Machine H^*_2: The machine proceeds as follows:

1. It samples a random $f \in \mathcal{F}_n$, $s \in \{0,1\}^n$. Sends s to the challenger, who feeds H^*_2 with a challenge $(\lambda, \widetilde{M}_b)$ where b is a random bit and:

   $$(\lambda, \mathsf{SimLock}_{\lambda,s}, \mathsf{SimLock}_{\lambda,0^n}) \leftarrow \mathsf{Samp}_{s,P^*_{1,(x,z,\rho)}}$$
   $$\widetilde{M}_0 \leftarrow \mathcal{O}(n, \mathsf{SimLock}_{\lambda,s}), \widetilde{M}_1 \leftarrow \mathcal{O}(n, \mathsf{SimLock}_{\lambda,0^n})$$

   Note that the state of P^* at the end of stage-1 can be completely defined by specifying the transcript of stage-1 (and in particular, (h, r)). Let st_1 denote the state of the prover when the transcript is fixed to λ (sampled above).
2. Run the prover P^* from the state st_1 and complete stage 2 as follows: send the tuple $(f, \widetilde{s} = f(s), \widetilde{M}_b)$ at the start of the stage 2, and proceed exactly as $B^*_{2,(x,z,\rho)}$. I.e., use the simulator of the ZK protocol to complete stage 2 and output the contents of $P^*_{2,(x,z,\rho)}$'s private output tape.

Now observe that since the randomness ρ has been fixed, the state st_1 sampled by $\mathsf{Samp}_{s,P^*_{1,(x,z,\rho)}}$ is distributed identically to the state of $P^*_{x,z,\rho}$ at the end of stage 1. Further, when $b = 0$, the rest of the execution (and output) of H^*_2 is distributed identically to that of $B^*_{2,(x,z,\rho)}$, and when $b = 1$ it is identical to that of $B^{**}_{2,(x,z,\rho)}$. Due to lemma 4.2, $\mathsf{Samp}_{s,P^*_{1,(x,z,\rho)}}$ is a nice sampler. H^*_2 therefore violates the indistinguishability property of the obfuscator $\mathcal{O}$ unless $|\delta'' - \delta'| \leq \mathsf{negl}(n)$.

Therefore, the machine $B^{**}_{2,(x,z,\rho)}$ outputs s with probability δ''. However, note that $B^{**}_{2,(x,z,\rho)}$ does not need to know the value of s, and executes perfectly even if $(f, \widetilde{s})$ are given by an outside challenger. Therefore, $B^{**}_{2,(x,z,\rho)}$ is actually an inverter for $f \in \mathcal{F}_n$, succeeding with the final probability $\delta - \mathsf{negl}(n)$. It follows that δ must be negligible in n, establishing the claim. □

We now come to the second part of the proof. Suppose that $x \notin \mathbf{L}$ and the success probability of $P^*_{x,z,\rho}$ is not negligible. This means that there exists a polynomial q such that for infinitely many values of n, $P^*_{x,z,\rho}$ succeeds with probability $\delta_2(n) \geq 1/q(n)$. We show how to build a prover $P^*_{2,(x,z,\rho)}$ which violates claim 5.3.

Let $\mathsf{Ext}_{\mathsf{WIPOK}}$ be the knowledge extractor corresponding to the 3-round WIPOK. For concreteness, assume that $\mathsf{Ext}_{\mathsf{WIPOK}}$ is a black-box extractor which uses cheating prover P^*_{WIPOK} as an oracle. Let $p := p(n)$ be the polynomial associated with the running time of the extractor $\mathsf{Ext}_{\mathsf{WIPOK}}$; i.e., $\mathsf{Ext}_{\mathsf{WIPOK}}$ extracts the witness in expected time $\frac{p(n)}{\Pr[P^*_{\mathsf{WIPOK}} \text{ succeeds}]}$. The machine $P^{**}_{2,(x,z,\rho)}$ incorporates the original prover P^*_2 and proceeds as follows.

Cheating prover $P^{**}_{2,(x,z,\rho)}$:

1. Initiate an execution of Simple-c$\mathcal{ZK}$ with an external V, routing its messages to the internal prover P^*_2, up to stage 2. Let st_2 be the state of P^*_2 at the end of stage 2. Denote the residual prover by $P^*_{\mathsf{st}_2}$.
2. Stop the execution with outside V after stage 2, and apply the extractor $\mathsf{Ext}_{\mathsf{WIPOK}}$ to the machine $P^*_{\mathsf{st}_2}$.
3. If the extractor halts within $\frac{4p}{\delta_2}$ steps, output whatever it outputs. Otherwise, output a random string of length n and halt.

Clearly, the running time of $P^{**}_{2,(x,z,\rho)}$ is at most $\mathsf{poly}(n) + 4p/\delta_2 = \mathsf{poly}(n) + 4pq$, which is polynomial. Further, by a standard averaging argument, it holds that if we fix the prover's state to st_2, then with probability at least $\delta_2/2$ over the sampling of st_2, the success probability of the residual prove $P^*_{\mathsf{st}_2}$ is at least $\delta_2/2$ in the remainder execution (i.e., the WIPOK). Call such states st_2 "good".

For every good st_2, the expected running time of the $\mathsf{Ext}_{\mathsf{WIPOK}}$ is $2p/\delta_2$ and it outputs a valid witness—specifically the secret s (since is x is false)—with probability $1-\mathsf{negl}(n)$. Therefore, by Markov's inequality, if st_2 is good, stopping $\mathsf{Ext}_{\mathsf{WIPOK}}$ after $4p/\delta_2$ steps (twice the expectation), still produces s with probability at least $(1 - \mathsf{negl}(n))/2$. Since st_2 is good with probability at least $\delta_2/2$, we have that our extractor outputs s with probability at least $\delta_2/2 \times 1/2 - \mathsf{negl}(n) \geq \delta_2/8$. This contradicts claim 5.3 unless δ_2 is negligible. This completes the proof of soundness. ■

Proof of knowledge property of Simple-c$\mathcal{ZK}$. We wish to note that the current construction may not satisfy the proof (argument) of knowledge property since for many states st_2 $\mathsf{Ext}_{\mathsf{WIPOK}}$ might take too long. Nevertheless, by a simple and standard modification, it is possible to build a *witness extended emulator* [Lin01, BL04] for our protocol. This is usually sufficient for most applications of POK. Alternatively, we can use a WIPOK with strict polynomial time extraction, in which case protocol Simple-c$\mathcal{ZK}$ also becomes a POK.

5.2 The Simulator

The simulator is described in two parts. First we describe an "internal simulator" $S^{\langle\cdot\rangle}$, which requires access to an oracle $\mathcal{I}$ that inverts injective one-way functions. This internal simulator is invoked by a "main" simulator S_{main}, which is described later.

Before jumping into the full description of our simulators, we make a few remarks to aid our description:

1. The internal simulator is essentially a "light weight twin" of the main simulator. Meaning that it is *identical* to the main simulator in all respects except that it does not run the "heavy" computation for computing the simulation trapdoor (i.e., the secret s). The internal simulator simply makes queries to the inversion oracle, denoted $\mathcal{I}$.

2. The main simulator therefore commits to its "light weight twin" as the program which will *deterministically* predict the string r in every session opened by V^*.

3. Although this works, implementing this idea leads to small circularity. To correctly predict r, it is essential that the twin simulator use the *same* random tape. In particular, this means that the randomness to commit to the program (that predicts r) should *come from the program to be committed*. I.e., the randomness for the commitment is correlated to the message (i.e., the program) to be committed. This is also true for all subsequent messages which require randomness.

4. The above circularity can be avoided as follows. We allow the internal "twin" simulator to only have a *commitment* to each bit of the random tape. The twin can access the bit by making a query to the inversion oracle $\mathcal{I}$. Since $\mathcal{I}$ only inverts (injective) one-way functions, we implement "commitment to the random tape" as *hardcore bits* of injective functions. The "committed" tape will be denoted by $\widetilde{\rho} = (\widetilde{\rho}_1, \ldots, \widetilde{\rho}_{\mathsf{poly}(n)})$ and the i-th bit of the tape will be defined as:

$$\rho_i = \mathsf{hcb}_g(g^{-1}(\widetilde{\rho}_i))$$

where g is a global function fixed at the beginning by the main simulator. It will be convenient to define a procedure RandomBit which computes the bit ρ_i as defined here.

5. We note that only the randomness that is used to "interact" with V^* needs to be the same for both simulators. The main simulator can have additional randomness, which is not available to its twin, for other tasks.

We now describe the twin simulator S to be used internally by the main simulator. Without loss of generality, we assume that the string 1^n is never chosen as a session identifier for any session; it will be used as a special trigger during execution of the simulator. We will use the words "internal simulator" and "twin simulator" interchangeably to mean the simulator $S^{\langle\cdot\rangle}$ described below. For concreteness, we recall the inversion oracle $\mathcal{I}$ here and fix a procedure $\mathsf{RandomBit}^{\mathcal{I}}_{g,\widetilde{\rho}}$ as follows:

Inversion oracle $\mathcal{I}$. The *inversion* oracle $\mathcal{I}(\cdot)$ takes queries of the form $(f, \widetilde{s})$ where $f \in \mathcal{F}_n$ is a injective function from the family of (certified) injective functions $\{\mathcal{F}_n\}$, and $\widetilde{s}$ is an element in the range of f. The oracle returns the (unique) value $s = f^{-1}(\widetilde{s})$ if one exists; otherwise it returns a special symbol $\perp$.

Procedure $\mathsf{RandomBit}^{\mathcal{I}}_{g,\widetilde{\rho}}(k)$. The procedure is defined for an injective function $g \in \mathcal{F}_n$ and a string $\widetilde{\rho} = (\widetilde{\rho}_1, \widetilde{\rho}_2, \ldots)$ of arbitrary length such that every component $\widetilde{\rho}_k$ is in the range of g. On input an integer k, the procedure returns:

$$\rho_k = \mathsf{hcb}_g(\vec{\rho}_k) \;\text{ where }\; \vec{\rho}_k \leftarrow \mathcal{I}\,(g, \widetilde{\rho}_k)$$

Twin simulator $S^{\mathcal{I}}(i, A^{\langle\cdot\rangle}, B, z, g, \widetilde{\rho})$. Algorithm S is an oracle TM. The input to the algorithm consists of a "session identifier" $i \in \{0,1\}^n$, a string A interpreted as an (interactive) oracle TM, a string B interpreted as an interactive TM, a string $z \in \{0,1\}^*$ interpreted as an advice string, a string g describing an *injective* one-way function, and a sufficiently long string $\widetilde{\rho} = (\widetilde{\rho}_1, \widetilde{\rho}_2, \ldots) \in \{0,1\}^*$ such that every component $\widetilde{\rho}_i \in \mathrm{Range}(g)$.

Simulator S has access to an *inversion* oracle $\mathcal{I}(\cdot)$ as defined above. Inputs $(g, \widetilde{\rho})$ fix the *implicit random tape* of S which is the unique bit-string $\rho = (\rho_1, \rho_2 \ldots)$ where bit $\rho_k = \mathsf{RandomBit}^{\mathcal{I}}_{g,\widetilde{\rho}}(k)$.[11] If, during its execution, S needs to access ρ_k it calls $\mathsf{RandomBit}_{g,\widetilde{\rho}}(k)$.

The simulator $S^{\mathcal{I}}$ computes its output as described below. It writes z on the auxiliary input tape of B, and a sufficiently long random string (taken from ρ) on the random tape of B. It then initiates an execution of B. If TM B launches a concurrent attack w.r.t. the protocol Simple-c$\mathcal{ZK}$, S proceeds as described below. Otherwise, if B deviates from the concurrent attack, S aborts the execution, outputting a special symbol $\perp$.

[11] ρ contains random tapes for TM B, A *and* randomness for all other tasks to be performed by the S. Further, S accesses any given bit of ρ only when it is needed in the computation.

1. If B opens a new session j, let x_j be the statement to be proven and let h_j be the first message of stage-1 of this session. S responds by sending the second message c_j, which is computed as follows.

   (a) Define the program $\Pi^{\langle\cdot\rangle}_{A,B,j}$ to be the following oracle-TM:

   $$\Pi^{\langle\cdot\rangle}_{A,B,j} := A^{\langle\cdot\rangle}\big(\ j,\ \ \mathsf{desc}(A^{\langle\cdot\rangle}),\ \ \mathsf{desc}(B),\ \ z,\ \ g,\ \ \widetilde{\rho}\ \big).$$

   I.e., program $\Pi^{\langle\cdot\rangle}_{A,B,j}$ is an oracle-TM which takes no input and it is essentially the program $A^{\langle\cdot\rangle}$ whose first input has been fixed to j (a session identifier), second input has been fixed to the *description* of $A^{\langle\cdot\rangle}$, third input has been fixed to the description of machine B, and last three inputs are fixed to z, g, and $\widetilde{\rho}$.

   (b) Compute c_j to be commitment to $\Pi^{\langle\cdot\rangle}_{A,B,j}$:

   $$c_j = \mathsf{Com}\left(h_j\left(\mathsf{desc}\left(\Pi^{\langle\cdot\rangle}_{A,B,j}\right)\right)\ ;\ u_j\right),$$

   where u_j is a uniform string taken from the implicit randome tape ρ.

2. If B sends a string r_j, denoting the third message of stage 1 of session j, do the following: if $j = i$, **output** the string r_i and halt. Otherwise, enter stage 2 of session j.

3. If B initiates stage-2 of an *existing* session $j \neq i$ by sending:

   $$(f_j, \widetilde{s}_j, \widetilde{M}_j)$$

   as first message of stage-2 of this session. The simulator responds to each message of this stage honestly by playing the role of the honest verifier of ZK protocol. If B sends the final message of this stage and succeeds in the proof, it marks the start of stage-3.

4. If B initiates stage-3 of an *existing* session $j \neq i$, S sends the query $(f_j, \widetilde{s}_j)$ to the oracle $\mathcal{I}$, and learns the answer s_j. I.e.,

   $$s_j \leftarrow \mathcal{I}\left(f_j, \widetilde{s}_j\right).$$

   If $s_j = \perp$, S **outputs** a special symbol $\mathsf{sim\_fail}$ and halts. Otherwise, it uses s_j as the witness and completes the WIPOK honestly.

5. If B halts without sending the third message of stage-1 of session i, denoted r_i, **output** the *view* of B and halt.[12]

This completes the description of the twin simulator $S^{\langle\cdot\rangle}$. □

We now describe the main simulator S_{main}. This simulator essentially runs the twin simulator S by setting its input $A = S$ (i.e., the code of the twin simulator itself) and $B = V^*$ (i.e., the concurrent adversarial verifier). In addition, S_{main} also simulates the inversion oracle $\mathcal{I}$ for S by keeping a track of the queries and recovering s_i for every session i using the obfuscated TM $\widetilde{M}_{\lambda_i,s_i}$.

Intuitively, setting $B = V^*$ and $A = S$, results in an execution where the twin simulator commits to its own code. Further, since S_{main} only initiates an execution S on these inputs, the "simulated transcript" for

[12] In particular, this happens necessarily when $i = 1^n$.

both S_{main} and S become identical. As a result, this ensures that S_{main} does have a "fake witness," namely the program $\Pi_{S,V^*,i}$ (along with a partial table $\mathcal{T}_j$ containing inverses for various one-way functions), to successfully run $\widetilde{M}_{\lambda_i,s_i}$ and get s_i.

We need to fix a global polynomial bound, denoted $Q_{V^*}(n)$, defined w.r.t. a given PPT adversary V^*. This bound will be used in terminating certain computations in the main simulator, if they run for too long. Roughly speaking, it is the total time that the obfuscated code will take in outputting the secret s when it is fed with a valid witness ω (for some statement λ).

Polynomial $Q_{V^*}(n)$: The polynomial is defined as follows. Let $T_{V^*}(n)$ be the fixed polynomial bounding the running time of V^*. Let a_{dio} be the *slowdown polynomial* of the obfuscator $\mathcal{O}$.[13] Let T_{Π,V^*} be a polynomial bounding the running time of the program $\Pi^{\mathcal{I}}_{S,V^*,1^n} := S^{\mathcal{I}}\left(1^n, \mathsf{desc}(S^{\langle\cdot\rangle}), \mathsf{desc}(V^*), z, g, \widetilde{\rho}\right)$ (for all strings $z, g, \widetilde{\rho}$ which are always bounded by T_{V^*}).

Finally, recall that the program $\mathsf{SimLock}_{\lambda,s}(\omega)$ takes as input a witness $\omega = (u, \Pi^{\langle\cdot\rangle}, \mathcal{T})$ and halts in time that is polynomial in the running time of $\Pi^{\mathcal{T}}$ and the size of $\mathcal{T}$. Let a_{lock} denote this polynomial, i.e., $\mathsf{Steps}(\mathsf{SimLock}_{\lambda,s}, \omega) \leq a_{\text{lock}}(T_{\Pi,V^*}(n), |\mathcal{T}|) \leq a_{\text{lock}}(T_{\Pi,V^*}(n), T_{V^*}(n))$. Then,

$$Q_{V^*}(n) \;:=\; a_{\text{dio}}\Bigg(n,\; a_{\text{lock}}\big(\; T_{\Pi,V^*}(n)\;, T_{V^*}(n)\;\big)\;\Bigg) \tag{5.2}$$

The description of our main simulator follows.

The main simulator $S_{\text{main}}\big(\mathsf{desc}(V^*), z\;;\;\xi\big)$. The input to the main simulator is the description of the TM implementing the cheating verifier V^*, the auxiliary input string z, and a sufficiently large random tape ξ. S_{main} computes its output as follows:

1. Sample a (certified) injective OWF $g \in \mathcal{F}_n$, a sufficiently large bit-string $\rho = (\rho_1, \rho_2, \ldots)$, a collection of strings $(\vec{\rho}_1, \vec{\rho}_2, \ldots)$ such that for every valid k: $\rho_k = \mathsf{hcb}_g(\vec{\rho}_i)$. Finally, define

   $$\widetilde{\rho} = (\; g(\vec{\rho}_1),\; g(\vec{\rho}_2),\; \ldots\;)$$

   and we write $\widetilde{\rho}_k = g(\vec{\rho}_k)$. The size of ρ is at most (a known polynomial) T_{V^*}. Finally, initialize a global table $\mathcal{T}$ containing the inverses of all coordinates of $\widetilde{\rho}$:

   $$\mathcal{T} \;=\; \big\{\; \big(g,\; \widetilde{\rho}_k,\; \vec{\rho}_k\big)\;\big\}$$

2. Perform all steps of the program $\Pi^{\langle\cdot\rangle}_{S,V^*,1^n}$ answering its oracle queries using the table $\mathcal{T}$, which is updated dynamically as described below. If a query cannot be answered using $\mathcal{T}$, output $\mathsf{sim\_fail}$. Let us recall that,

   $$\Pi^{\mathcal{I}}_{S,V^*,1^n} := S^{\mathcal{I}}(1^n, \mathsf{desc}(S^{\langle\cdot\rangle}), \mathsf{desc}(V^*), z, g, \widetilde{\rho}). \tag{5.3}$$

   Note that since 1^n is never selected as a session identifier, the output of this simulation is always either a view of V^*, special symbol $\mathsf{sim\_fail}$, or abort symbol $\perp$.

3. Table $\mathcal{T}$ is updated dynamically as the execution of $\Pi^{\langle\cdot\rangle}_{S,V^*,1^n}$ proceeds. To describe how $\mathcal{T}$ is updated, define the following three quantities associated with a session whose identifier is $i \in \{0,1\}^n \setminus \{1^n\}$:

[13] That is, a_{dio} is such that for every TM M, and input x for which $\mathsf{Steps}(M, x)$ is defined, it holds that $\mathsf{Steps}(\mathcal{O}(M), x) \leq a_{\text{dio}}(n, \mathsf{Steps}(M, x))$.

(a) Statement $\lambda_i = (h_i, c_i, r_i)$

(b) Table $\mathcal{T}_i$ which is equal to the contents of $\mathcal{T}$ at the time when r_i is sent by V^*

(c) "Fake" witness $\omega_i = \left(u_i,\ \mathsf{desc}(\Pi^{\langle\cdot\rangle}_{S,V^*,i}),\ \mathcal{T}_i\right)$

Note that the value of ω_i also gets completely defined when r_i is received. Further, S_{main} has access to ω_i for every session i in the simulation.

4. When V^* completes stage-2 of some session i, it has successfully proven the validity of the statement defined by the tuple $\left(f_i, \widetilde{s}_i, \widetilde{M}_{\lambda_i,s_i}\right)$. Compute

$$s_i = \widetilde{M}_{\lambda_i,s_i}(\omega_i)$$

by running the obfuscated program $\widetilde{M}_{\lambda_i,s_i}$ for at most $Q_{V^*}(n)$ steps (see (5.2). If a valid s_i is received, append $(f_i, \widetilde{s}_i, s_i)$ to $\mathcal{T}$; otherwise, output $\mathsf{sim\_fail}$ and halt.[14]

5. When $\Pi^{\langle\cdot\rangle}_{S,V^*,1^n}$ sends a query $(f_i, \widetilde{s}_i)$, S_{main} locates the corresponding entry in $\mathcal{T}$. If a matching entry $(f_i, \widetilde{s}_i, s_i)$ is found, it returns s_i. If a matching entry is not found, S_{main} aborts the entire execution, **outputs** a special symbol $\mathsf{sim\_fail}$, and halts.

When the program $\Pi^{\langle\cdot\rangle}_{S,V^*,1^n}$ halts, S_{main} **outputs** whatever $\Pi^{\langle\cdot\rangle}_{S,V^*,1^n}$ outputs and halts. $\square$

5.3 Indistinguishability of Simulation

We argue that the output of S_{main}, on input the description of a cheating verifier V^*, is computationally indistinguishable from the output of V^* in a random execution of the concurrent attack experiment. We prove this by designing a several hybrid experiments, starting from the "real" experiment in which all witnesses are actually given to the experiment.

At a high level, there are three main steps in the proof:

1. The first step is to commit to the description of the twin simulator using fresh randomness u_i (that is not correlated to $\widetilde{\rho}$) for every session i.

2. The second step is to argue that even if S_{main} uses randomness derived from $\mathsf{RandomBit}_{g,\widetilde{\rho}}$ (which is correlated to $\widetilde{\rho}$), the simulated view is computationally indistinguishable.

3. The third step is to use the secret s_i instead of the witness w_i in the WIPOK part. However, this needs to be done one-by-one for each session *in the order in which r_i is received*. The order is important to ensure that all sessions whose stage-3 gets scheduled before a particular r_i, are actually using the "fake witness" (and not the real witnesses). This will ensure that the fake witness for session i is also available to the simulator.

We now describe our hybrids.

[14] Note that in general $\mathcal{T}_i$ does *not* contain s_i since s_i is computed after r_i was sent, which in particular fixes $\mathcal{T}_i$ before s_i is even defined.

Hybrid H_1. This experiment is identical to the concurrent attack experiment. The experiment incorporates the verifier program V^* and auxiliary input z. It initiates a concurrent execution with V^*. For every session i, H_1 is provided with a witness w_i to the statement x_i being proven. H_1 uses honest prover algorithms and fresh randomness for all tasks. When V^* halts, H_1 outputs the view of V^*, denoted ν_1.

Hybrid H_2. This hybrid is identical to H_1 except that at the start of the experiment, H_2 samples the function g, string $\widetilde{\rho}$, and table $\mathcal{T}$ exactly as in step 1 of S_{main}. Let ν_2 denote the output of H_2.

Hybrid H_3. This experiment is identical to H_2 except that in every session i, commitment c_i (i.e., the second message of stage 1) is computed as follows:

$$c_i = \mathsf{Com}\left(h_i\left(\mathsf{desc}\left(\Pi^{\langle\cdot\rangle}_{S,V^*,i}\right)\right)\ ;\ u_i\right),$$

where h_i is the first message of session i, program $\Pi^{\langle\cdot\rangle}_{S,V^*,i}$ is defined in (5.3),[15] and u_i is uniformly chosen. Let ν_3 denote the output of H_3.

Hybrid H_4. This experiment is identical to H_3 except that after sampling g, $\widetilde{\rho}$, and $\mathcal{T}$, the experiment does not sample any more random bits. Instead, H_4 keeps a counter k initialized to 0 and whenever a new random bit is needed, it does the following: increments k by 1, sets $\rho_k \leftarrow \mathsf{RandomBit}^{\mathcal{I}}_{g,\widetilde{\rho}}(k)$, and uses ρ_k as the random bit. Let ν_4 denote the output of H_4.

Before going further, observe the following (straightforward) claim:

Claim 5.4. *For every $i \in [4]$, hybrid H_i halts in polynomial time and outputs the symbol* sim_fail *with probability 0. Further, $\nu_1 \equiv \nu_2 \overset{c}{\approx} \nu_3 \overset{c}{\approx} \nu_4$.*

Proof. By construction, every H_i uses the real witnesses $\{w_j\}$ to succeed in WIPOK part of the protocol, and all other steps are polynomial time computations. Further, every H_i only access $\mathcal{T}$ via RandomBit procedure. By construction $\mathcal{T}$ contains answers to all queries of RandomBit, never leading to sim_fail.

The difference between H_1 and H_2 does not change the output, and hence $\nu_1 \equiv \nu_2$. By computational hiding of Com, $\nu_2 \overset{c}{\approx} \nu_3$ since the only difference between H_2 and H_3 is that they commit different, but known, strings to V^*. By applying the same argument to H_3 and H_4, and using the *pseudo-randomness* property of hard-core bits, we conclude that $\nu_3 \overset{c}{\approx} \nu_4$. □

For every partial transcript ν of the concurrent execution, we can define an ordering of the sessions that appear in ν. We order the session identifiers i according to the order in which strings r_i—which is the second message of stage-1 of session i—are received. I.e., identifier i_1 is ordered before identifier i_2 if r_{i_1} is scheduled before r_{i_2}. Note that ordering depends on the scheduling and hence the actual transcript ν. The ordering is well defined for the sessions which have appeared in ν even if ν is not completely fixed yet and this order does not change as new sessions are added to ν in course of a concurrent attack.

Define $\mathsf{id}_\nu(i)$ to be a number which tells us the position of session i in (possibly partial) transcript ν acceding to the above ordering. Note that $\mathsf{id}^{-1}_\nu(m)$ is also well defined and identifies which session's r was received in m-th position. The transcript ν will often be clear from the context, and we will drop it from the notation and simply write $\mathsf{id}(i)$ and $\mathsf{id}^{-1}(m)$.

For $m \in [T_{V^*}]$, we define our next set of hybrids.

[15] Note that $g, \widetilde{\rho}$ are already defined at this point and so this program is also well defined.

Hybrid $G_{0:2}$**.** This hybrid is same as H_4.

Hybrid $G_{m:1}$**.** This hybrid is same as $G_{m-1:2}$ except for the following difference. Let $i_m = \mathsf{id}^{-1}(m)$ be the session-id of a session such that r_{i_m} is the m-th such string scheduled/sent by V^*. Then, this experiment computes:

$$s_{i_m} \leftarrow \widetilde{M}_{\lambda_{i_m}, s_{i_m}}(\omega_{i_m})$$

for at most $Q_{V^*}(n)$ steps; (here all quantities are defined as before w.r.t. the session i_m). If $s_{i_m} \neq \perp$, $G_{m:1}$ appends $(f_{i_m}, \widetilde{s}_{i_m}, s_{i_m})$ to $\mathcal{T}$; otherwise it outputs a special symbol $\mathsf{sim\_fail}$, and halts. Let $\nu_{m:1}$ denote the output of $G_{m:1}$.

Hybrid $G_{m:2}$**.** This hybrid is same as $G_{m:1}$ except that when stage-3 of session i_m begins, it probes $\mathcal{T}$ on query $(f_{i_m}, \widetilde{s}_{i_m})$ to obtain s_{i_m}. If s_{i_m} is found, it is used as the witness to complete the WIPOK proof. If s_{i_m} is not found, $G_{m:2}$ outputs $\mathsf{sim\_fail}$. Let $\nu_{m:2}$ denote the output of this experiment.

It should be noted that the output of the final hybrid, namely $G_{T_{V^*}:2}$, is identical to that of S_{main}. We now show that:

Lemma 5.5 (Main Lemma)**.** *For every $m \in \{1, \ldots, T_{V^*}\}$, the following conditions hold:*

- $G_{m:1}$ *and* $G_{m:2}$ *are* PPT *experiments which output* $\mathsf{sim\_fail}$ *with only negligible probability, where the probability is taken over the sampling of tuple $(g, \widetilde{\rho})$ used by the procedure* $\mathsf{RandomBit}_{g,\widetilde{\rho}}$.
- $\nu_{m-1:2} \overset{s}{\approx} \nu_{m:1} \overset{c}{\approx} \nu_{m:2}$

Proof. We start by observing that $G_{0:2}$ is the same as H_4, it is a PPT experiment which outputs $\mathsf{sim\_fail}$ with negligible probability (by claim 5.4). Recall that program $\Pi^{\mathcal{I}}_{S,V^*,i}$ outputs $\mathsf{sim\_fail}$ if and only if there exists a session j in the execution such that $(f_j, \widetilde{s_j})$ is sent as a query but a unique $f^{-1}(\widetilde{s}_j)$ does not exist. However, since we assumed[16] that it is possible to efficiently test that f_j is injective and $\widetilde{s}_j$ is in the range of f_j, it follows that a unique inverse always exists, and hence $\Pi^{\mathcal{I}}_{S,V^*,i}$ never outputs $\mathsf{sim\_fail}$.

Let us compare the hybrids $G_{m-1:2}$ and $G_{m:1}$. If we fix the random tape of these hybrids to be the *same* string ξ then, by construction, the execution of the two experiments are *identical* up to the point where r_{i_m} is sent (where $i_m = \mathsf{id}^{-1}(m)$ is the identifier of the session in m-th position in the order). Let $\mathcal{T}_{i_m}$ be the contents of the table at this point. Further, let

$$c_{i_m} = \mathsf{Com}\left(h_{i_m}\left(\mathsf{desc}\left(\Pi^{\langle\cdot\rangle}_{S,V^*,i_m}\right)\right) \; ; \; u_{i_m}\right)$$

be the commitment sent as the second message of stage-1 of session i_m.[17] If $G_{m-1:2}$ does not output $\mathsf{sim\_fail}$ on randomness ξ, then the string:

$$\omega_{i_m} = \left(u_{i_m}, \; \mathsf{desc}(\Pi^{\langle\cdot\rangle}_{S,V^*,i_m}), \; \mathcal{T}_{i_m}\right)$$

[16] As noted earlier, these assumption are *only* for simplicity and can be removed because of the following. The proof Π_{ZK} given by the verifier already proves that $\widetilde{s}_j$ is in the range of f_j; we can further require it to also prove that f_j was sampled using the key-generation algorithm of $\{\mathcal{F}_n\}$ which is known to be injective. By soundness of ZK, it will follow that $\Pi^{\mathcal{I}}_{S,V^*,i}$ outputs $\mathsf{sim\_fail}$ with negligible probability.

[17] By construction, c_{i_m}, $\mathcal{T}_{i_m}$, and r_{i_m} are same in both experiments.

is a "fake witness" to the statement $\lambda_{i_m} = (h_{i_m}, c_{i_m}, r_{i_m})$. This is because by our ordering relation, every message between $r_{i_{m-1}}$ and r_{i_m} is either a stage-3 message of a session j such that $\mathsf{id}(j) < m$ or it is not a stage-3 message at all. Therefore, if V^* sends a TM $\widetilde{M}_{\lambda_{i_m}, s_{i_m}}$ which is indeed an obfuscation of $\mathsf{SimLock}_{\lambda_{i_m}, s_{i_m}}$, $G_{m:1}$ learns s_{i_m} successfully (since the obfuscated TM takes less than $Q_{V^*}(n)$ steps, by definition of Q_{V^*}). By the soundness of ZK protocol (stage 2), $\widetilde{M}_{\lambda_{i_m}, s_{i_m}}$ is indeed a correct obfuscation with $1 - \mathsf{negl}(n)$ probability, and hence outputs s_{i_m} within $Q_{V^*}(n)$ steps. As the only difference between $G_{m-1:2}$ and $G_{m:2}$ is in how they sample s_{i_m}, it follows that:

- $\nu_{m-1:2} \overset{s}{\approx} \nu_{m:1}$
- $G_{m:1}$ takes at most $a(Q_{V^*})$ steps more than the running time of $G_{m-1:2}$

Next, we compare $G_{m:1}$ and $G_{m:2}$. The only difference between two hybrids is that they use a different witness in WIPOK stage of session i_m. Clearly this does not affect the running time. This is because WIPOK proves *an* $\mathbf{NP}$ *statement* with a *known bound* on the witness size; therefore, w.l.o.g., the prover of WIPOK can be assumed to run in the same time for every witness (with all other things being equal).

Finally, we show that due to WI property of the WIPOK proof, it must be that $\nu_{m:1} \overset{c}{\approx} \nu_{m:2}$. If the two outputs are not computationally indistinguishable, the following hybrid machine G^*_m violates the WI property of stage 3: G^*_m is identical to $G_{m:1}$ except that instead of internally running the prover algorithm of WIPOK in stage 3 of session i_m, it receives these messages from an external prover.

Note that prior to the start of stage 3 of session i_m, G^*_m receives no external messages and therefore it successfully completes every session prior to stage of i_m. For all sessions after i_m, the hybrid uses the real witnesses to complete stage 3. Therefore, G^*_m is indeed PPT which is identical to either $G_{m:1}$ or $G_{m:2}$ depending upon which witness is used by the external prover. It follows that G^*_m violates WI unless $\nu_{m:1} \overset{c}{\approx} \nu_{m:2}$. ■

Theorem 5.6. *Assume that $\{\mathcal{H}_n\}$ is a family of collision-resistant hash function, $\{\mathcal{F}_n\}$ a family of injective one-way functions,* Com *a non-interactive perfectly binding (string) commitment scheme, and protocols Π_{ZK} and Π_{WIPOK} are (constant-round)* ZK *and* WIPOK *systems respectively for* $\mathbf{NP}$*. Further assume that for every polynomial $a : \mathbb{N} \to \mathbb{N}$, and every* hard distribution $\mathcal{Z}$ *over the statements of $\mathbf{L}^a_{\mathrm{sim}}$, $\mathcal{O}$ is a $\mathcal{Z}$-*auxiliary differing-input obfuscation *(*diO*) for the class of all polynomial-size Turing machines that halt in a polynomial number of steps. Then,* $\mathsf{Simple\text{-}c}\mathcal{ZK}$ *is a constant-round, fully concurrent zero-knowledge protocol for all languages in* $\mathbf{NP}$*, with perfect completeness and negligible soundness.*

Proof. The perfect completeness and constant-round claim is straightforward to see. The negligible soundness was proven in lemma 5.2. Concurrent ZK follows by observing that the output of S_{main} is indistinguishable from the view of V^* in a concurrent attack due to claim 5.4 and lemma 5.5 and by observing that m is bounded by the (polynomial) running time V^*. ■

6 The Four Round Protocol

In the previous section, we presented a reduction from *constant round, concurrent zero-knowledge* to diO based on standard cryptographic assumptions. In this section, we present a similar reduction for four message concurrent zero-knowledge.

Let us start by optimizing the number of rounds in our constant round protocol of previous section. The standalone ZK protocol used in stage 2 has at least four rounds.[18] Since the last message of this ZK protocol must come from the verifier, our resulting protocol will have at least five rounds even after optimizations.

We consider two approaches to obtain a four round protocol. First, we can use a two-round ZK protocol with *super polynomial time simulation*[Pas03]. This approach gives us a reduction where the soundness of the resulting protocol must assume sub-exponential hardness assumptions. The second approach is to use a WI protocol to prove the correctness of the obfuscated program. However, in typical applications of WI, to get any useful security we must somehow ensure that the statement being proven has at least two witnesses.

The standard approach in such cases is to consider two independently sampled statements, in this case, two obfuscated programs $\widetilde{M}_{\lambda,s}$ and $\widetilde{M}_{\lambda,s'}$; and prove that at least one of them is correctly constructed using a WI proof. However, this approach actually fails for a very interesting reason. Although it does hide one of the secrets s, s', it actually breaks the simulation. Indeed, the internal simulator committed to in the preamble, will have no efficient way of knowing which of these two programs is actually correctly prepared. In particular, it will have to ask for the inversion of *two* challenges per session but the main simulator might be able to return only one of them (since one of the obfuscated programs could have been maliciously prepared). Attempting to overcome this subtle issue actually breaks the hardness of $\mathbf{R}_{\mathrm{sim}}$.

We therefore use a different approach; we set up an "intermediate statement" which is selected by the prover, and require the prover to provide a WIPOK of its correctness. The verifier then proves that either this intermediate statement is true or the obfuscated program is correctly prepared. The intermediate statement is prepared in such a way that it is possible to make it false and succeed (using the real witness for x) without the verifier noticing. This allows us to ensure that the obfuscated program must be correctly prepared and simulation still continues to go through. For the soundness, roughly speaking, we can extract the witness corresponding to the "intermediate statement" by using the extractor of WIPOK; we then use it to simulate the WI proof that comes from verifier's side. This allows us to again enforce the ideas we developed to prove the soundness of the Simple-c$\mathcal{ZK}$ protocol.

To setup the "intermediate statement" we use perfectly binding commitments to specially prepared strings. In the final proof, we will need to actually extract the secret s to violate the hardness of one-way functions. We get around this difficulty by using a combination of the WIPOK used by the prover and a ZAP proof. We now present an overview of our four round protocol below. The formal description of the protocol appears in Section 6.1. A pictorial representation of the protocol appears in figure 4.

Four round protocol for concurrent zero-knowledge. The protocol has four components whose messages will be sent in parallel (as depicted in figure 4).

1. The first component is the GenStat protocol, producing statements of the form $\lambda = \langle h, c, r\rangle$.

2. The second component is a three round WIPOK given by the *prover to the verfier*. The prover prepares two commitments, namely $\widetilde{t}_1 = \mathsf{Com}(0 \parallel t_1; v_1)$ and $\widetilde{t}_2 = \mathsf{Com}(0 \parallel t_2; v_2)$ and proves that either $(\widetilde{t}_1, \widetilde{t}_2)$ are correctly prepared or x is true. The 3 messages of this WIPOK will be denoted by $\langle \alpha, \beta, \gamma\rangle$.

3. The final component is a ZAP for a specially prepared statement, which will let us extract either a witness to x or the secret s in the proof of soundness. The special statement is prepared as follows.

   The prover creates two commitments τ_1, τ_2 such that τ_1 uses string t_1 (defined above in item 2) as its randomness; likewise τ_2 uses t_2. Further, the value committed to in one of them is the witness w

[18] To keep our *reduction* from concurrent ZK to obfuscation free from "knowledge assumptions," we cannot use 3-round ZK protocols based on such assumptions.

for statement x. The prover then proves, using a ZAP, that there exists $i \in \{1,2\}$ such that τ_i is a commitment to w using t_i. The two messages of this ZAP are denoted by $\langle \sigma', \pi' \rangle$.

The main result of this section is the following theorem, which is proven by showing that protocol $\mathrm{c}\mathcal{ZK}$ is fully concurrent zero-knowledge (Theorem 6.3) shortly.

Theorem 6.1. *Assume the existence of collision-resistant hash functions and trapdoor one-way permutations (alternatively, injective one-way functions and* ZAP *proofs for* $\mathbf{NP}$*). Further, for every polynomial* $a : \mathbb{N} \to \mathbb{N}$*, and every* hard distribution $\mathcal{Z}$ *over the statements of* $\mathbf{L}^a_{\mathrm{sim}}$*, assume the existence of* $\mathcal{Z}$-auxiliary differing-input obfuscation *(*diO*) for the class of all polynomial-size Turing machines that halt in a polynomial number of steps. Then, there exists a* four message*, fully concurrent zero-knowledge protocol with negligible soundness, for all languages in* $\mathbf{NP}$.

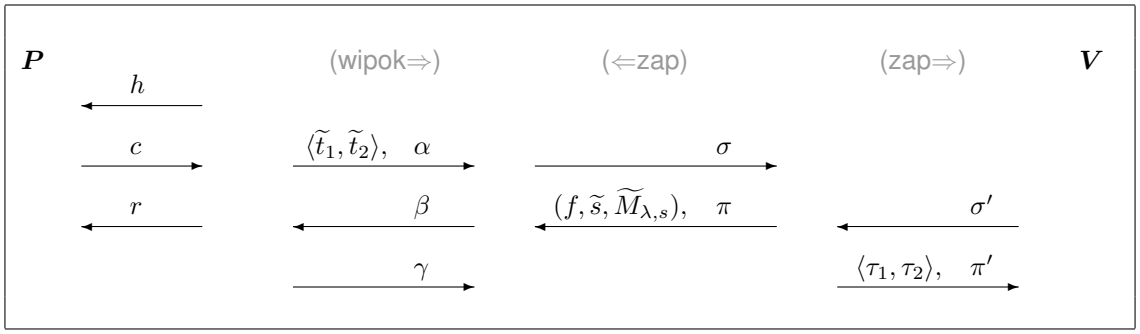

Figure 4: Schematic diagram of the four round protocol

6.1 Protocol $\mathrm{c}\mathcal{ZK}$

Inputs. The common input is a statement $x \in \mathbf{L}$ where language $\mathbf{L} \in \mathbf{NP}$. The prover's auxiliary input is a witness w such that $\mathbf{R}(x, w) = 1$. Let us assume that $\mathbf{L}$ is the $\mathbf{NP}$-complete language of graph Hamiltonicity.[19]

The security parameter n is an implicit input to both the prover as well as the verifier. Without loss of generality, we assume that $|w| = n$. Let $\{H_n\}$ be a family of CRHF, $\{F_n\}$ be a family of injective one-way functions with efficiently testable range membership, and Com be a non-interactive perfectly binding commitment scheme for committing strings of arbitrary polynomial length. For concreteness, let Π_{WIPOK} be a 3-round public-coin WI protocol with special properties (described in Section 2) and Π_{ZAP} be a ZAP proof system.

Rounds. The prover P and the verifier V interact as follows.

1. The verifier sends a CRHF function $h \leftarrow \mathcal{H}_n$.
2. The prover sends
$$\left(c,\ \langle \widetilde{t}_1, \widetilde{t}_2 \rangle,\ \alpha,\ \sigma\right)$$
where:

[19] The statements $x \in \mathbf{L}$ are graphs which contain a Hamiltonian cycle, and the witness w is a Hamiltonian cycle in x.

(a) $c = \mathsf{Com}(0^n; u)$ for a randomly chosen u;

(b) $\widetilde{t}_1, \widetilde{t}_2$ are commitments created as follows: $\widetilde{t}_i = \mathsf{Com}(0 \parallel t_i \; ; \; v_i)$ for every $i \in \{1, 2\}$ and t_i, v_i are random strings (of sufficient length);

(c) α is the first prover message of Π_{WIPOK} for the statement:

- $\exists\, w$ s.t. $\mathbf{R}(x, w) = 1$; OR
- $\exists\, (t_1, t_2, v_1, v_2)$ s.t. $\widetilde{t}_1 = \mathsf{Com}(0 \parallel t_1; v_1)$ and $\widetilde{t}_2 = \mathsf{Com}(0 \parallel t_2; v_2)$

(d) σ is the first message of the ZAP proof;

The prover uses witness of the *second part*, namely (t_1, t_2, v_1, v_2), for computing α and all subsequent messages corresponding to the prover of Π_{WIPOK}.

3. The verifier sends

$$\Big(r,\ \big(f, \widetilde{s}, \widetilde{M}_{\lambda,s}\big),\ \beta,\ \pi,\ \sigma'\Big)$$

where:

(a) $r \leftarrow \{0,1\}^n$ is a random string, and $\lambda := (h, c, r)$ is already defined by the transcript;

(b) $f \in \mathcal{F}_n$ is a (randomly chosen) injective one-way function, $\widetilde{s} = f(s)$ for a random string $s \in \{0,1\}^n$, and $\widetilde{M}_{\lambda,s}$ is the obfuscation of $\mathsf{SimLock}_{\lambda,s}$, generated using randomness (say) ζ.[20] I.e.,

$$\widetilde{M}_{\lambda,s} \leftarrow \mathcal{O}\big(\mathsf{SimLock}_{\lambda,s}\ ;\ \zeta\big).$$

(c) β is the second message of Π_{WIPOK} (a random string);

(d) π is the second message of ZAP proof (w.r.t. first message σ) proving the statement:

- $\exists\, (s, \zeta)$ s.t.: $\widetilde{s} = f(s)$ and $\widetilde{M}_{\lambda,s} = \mathcal{O}\big(\mathsf{SimLock}_{\lambda,s}\ ;\ \zeta\big)$; OR
- $\exists\, (t_1, t_2, v_1, v_2)$ s.t. $\widetilde{t}_1 = \mathsf{Com}(0 \parallel t_1; v_1)$ and $\widetilde{t}_2 = \mathsf{Com}(0 \parallel t_2; v_2)$

The verifier uses (s, ζ) as the witness to compute π.

(e) σ' is a freshly sampled first message of the ZAP proof.

4. The prover checks that (σ, π) is a valid proof. If so, it sends

$$\Big(\gamma,\ \ \langle \tau_1, \tau_2 \rangle,\ \ \pi'\Big)$$

where:

(a) γ is the last message of Π_{WIPOK} (computed using the witness (t_1, t_2, v_1, v_2)—see step 2).

(b) One of the strings τ_1, τ_2 is a commitment to the real witness w *using one of* t_1, t_2 *as randomness*. That is, P chooses a random $i \in \{1, 2\}$ and sets: $\tau_i = \mathsf{Com}(w; t_i)$. The other string is a commitment to 0^n. That is, $\tau_{i'} = \mathsf{Com}(0^n; t_{i'})$ where $i' = \{1, 2\} \setminus i$.

[20] Recall that $\mathsf{SimLock}_{\lambda,s}(\cdot) = \mathsf{SimLock}(\lambda, \cdot, s)$, defined earlier, is a TM which outputs s if and only if its input is ω s.t. $\mathbf{R}_{\mathrm{sim}}(\lambda, \omega)$=1; otherwise it outputs the string 0^n.

(c) π' is the the second message of the ZAP proof, proving that there exists an i such that either "τ_i commits to w OR it commits to the secret s using randomness t_i." Formally, π' proves that: $\exists\,(i, a, t_i, v_i)$ such that $\tau_i = \mathsf{Com}(a; t_i)$ where $i \in \{1, 2\}$, $a \in \{0,1\}^n$, $\widetilde{t_i} = \mathsf{Com}(0 \parallel t_i; v_i)$ and it holds that *either* $\mathbf{R}(x, a) = 1$ OR $f(a) = \widetilde{s}$. The prover computes π' using the knowledge of (i, w, t_i, v_i).

5. ***Verifier's output.*** The verifier accepts if and only if both (α, β, γ) and (σ', π') are accepted by the corresponding verifier algorithms. Otherwise it rejects.

This completes the description of our protocol. □

We now prove that c$\mathcal{ZK}$ is a concurrent zero-knowledge protocol with negligible soundness error.

6.2 Proof of Soundness

The proof of soundness is very similar to the soundness of Simple-c$\mathcal{ZK}$. We therefore only present a proof sketch.

Lemma 6.2. *Prototocol* c$\mathcal{ZK}$ *has negligible soundness error.*

Proof. The main idea of the proof is to extract the values t_1, t_2, v_1, v_2 from a cheating prover P^*, and use them in the ZAP proof given by the verifier. Thereafter, extract the value of s from τ_1, τ_2 using t_1, t_2 and invert the one-way function family. The details follow.

Suppose that there exists a polynomial q and a cheating prover P^* such that for infinitely many values of the security parameter n, P^* successfully proves a false $x \notin \mathbf{L}$ probability $\delta = 1/q(n)$. Further, let $p := p(n)$ be the polynomial associated with the running time of the extractor $\mathsf{Ext}_{\mathsf{WIPOK}}$; i.e., $\mathsf{Ext}_{\mathsf{WIPOK}}$ extracts the witness in expected time $\frac{p(n)}{\Pr[P^*_{\mathsf{WIPOK}} \text{ succeeds}]}$.

Consider the following prover machine M_1^* which behaves like P^* except that it also attempts to extract the value of the secret s.

Machine M_1^*: this machine incorporates the prover P^* and interacts with an external V of our c$\mathcal{ZK}$ protocol as follows:

1. Route the first two messages between P^* and V. Let P^*_{st} denote the state of P^* at this point and let $\langle \widetilde{t_1}, \widetilde{t_2} \rangle$ be the two commitments in the second message and α be the first message of the WIPOK protocol. Before proceeding further, do the following:
   (a) Consider the machine B^* which acts as prover of the Π_{WIPOK} protocol for the statement $\langle \widetilde{t_1}, \widetilde{t_2} \rangle$. B^* always sends α that has already been received. Upon receiving a challenge β, B^* samples the third message of our c$\mathcal{ZK}$ protocol honestly by following V's algorithm except that it uses β received from the outside party as the challenge for the internal WIPOK protocol. If internal P^*_{st} responds with a convincing last message, B^* forwards only the first component, namely γ to the outside verifier. This defines the machine B^* completely.
   (b) Apply the extractor $\mathsf{Ext}_{\mathsf{WIPOK}}$ to the machine B^*, running for at most $4p/\delta$ steps. If a correct witness to $(\widetilde{t_1}, \widetilde{t_2})$ is not extracted, abort the entire execution; otherwise continue to the next step with the extracted value, say (t_1, t_2, v_1, v_2).

2. Receive the third message from the outside verifier of our c$\mathcal{ZK}$ protocol, feed it to P^*_{st}, and forward the resulting response to V. In addition, if τ_1, τ_2 are the commitment strings in the last message, attempt to extract the values a_1, a_2 (using the randomness t_1, t_2) in these commitments. If $f(a_i) = \widetilde{s}$ for some $i \in \{1, 2\}$ (where $(f, \widetilde{s})$ are received from V in the third round) write a_i on its private output tape and halt.

Observe that the only difference between M_1^* and P^* is that the former employs a witness extraction strategy and aborts if it fails after $4p/\delta$ steps. Since there is at least $\delta/2$ fraction of states st such that P^*_{st} succeeds with probability at least $\delta/2$ in the remainder of the execution, we have that the extraction does succeed with probability at least $1/2 - \mathsf{negl}(n)$ for such a P^*_{st}. Since $x \notin \mathbf{L}$, this extraction results in $s = f^{-1}(\widetilde{s}$. Therefore, M_1^* writes s on its private output tape with at least $\frac{\delta}{2} \times (\frac{1}{2} - \mathsf{negl}(n)) \times \frac{\delta}{2} \geq \frac{\delta^2}{16}$.

Next, we consider the machine which uses the extracted values (t_1, t_2, v_1, v_2) as the witness in computing the verifier's ZAP (σ, π). Formally, consider the following machines:

Machine M_2^*: identical to M_1^* except that it internally simulates all the steps of our honest verifier V and does not communicate with an external V.

Machine M_3^*: identical to M_2^* except that it uses extracted values (t_1, t_2, v_1, v_2) in computing the ZAP proof π.

It is clear that the outputs of M_1^* and M_2^* are distributed identically; further due to the WI property, the output of M_3^* is computationally indistinguishable from that of M_2^*. It follows that M_3^* writes a string s on its private output tape such that $f^{-1}(s) = s$ with probability at least $\delta^2/16 - \mathsf{negl}(n)$.

Finally, consider the following machine M_4^*: this machine is identical to M_3^* except that instead of sending the obfuscation of $\mathsf{SimLock}_{\lambda,s}$, it sends obfuscation of $\mathsf{SimLock}_{\lambda,0^n}$ (in the internal simulation of honest V). We claim that the outputs of M_3^* and M_4^* are computationally indistinguishable. This is demonstrated by the following hybrid machine:

Machine H^*: this machine is identical to M_3^* except that it does not simulate the messages $\lambda := (h, c, r)$ and $\widetilde{M}_{\lambda,s}$ internally. Instead these messages are received from an external verifier V_1' which acts exactly like the $\mathsf{GenStat}$ verifier V_1; however, after the messages (h, c, r) are completed, H^* internally generates $(f, f(s))$ and then sends s to V_1'. Verifier V_1' responds by sending either an honestly generated obfuscation of $\mathsf{SimLock}_{\lambda,s}$ or that of $\mathsf{SimLock}_{\lambda,0^n}$, chosen randomly.

It is straightforward to see that machine H^* satisfies the conditions of a nice sampler; in particular, it receives the obfuscation of one of the two machines produced by the nice sampler Samp_{s,H^*}. H^* is identical to M_3^* (resp., M_4^*) if V_1' sends an obfuscation of $\mathsf{SimLock}_{\lambda,s}$ (resp., $\mathsf{SimLock}_{\lambda,0^n}$). Therefore, from the indistinguishability property of $\mathcal{O}$, the outputs of M_3^* and M_4^* are computationally indistinguishable. It follows that M_4^* also write the inverse s on its private output tape. However, note that the execution of M_4^* does not need to know the string s and runs perfectly even if only $(f, \widetilde{s} = f(s))$ are provided to it by an external challenger. Therefore M_4^* is an inverter for the one-way function. Hence the lemma. ■

6.3 The Simulator and Proof of Concurrent Zero-Knowledge

As before, the simulator is described in two parts: an internal/twin simulator which requires access to the oracle $\mathcal{I}$ and a main simulator which invokes the internal simulator. The description of these simulators is

really the same as for the Simple-c$\mathcal{ZK}$ protocol except that the messages are computed according to the four-round c$\mathcal{ZK}$ protocol.

To distinguish the simulators from previous sections, the internal and the main simulators for the four-round protocol will be denoted by $S_4^{\langle\cdot\rangle}$ and $S_{\text{main}}^{(4)}$ respectively.

Twin simulator $S_4^{\mathcal{I}}(i, A^{\langle\cdot\rangle}, B, z, g, \widetilde{\rho})$**.** This simulator proceeds identically to the twin simulator $S^{\langle\cdot\rangle}$ defined in section 5.2 except that it only has to send two messages in every session: the second and the fourth messages. We only mention how these messages are prepared:

1. When B opens a new session j, let x_j be the statement to be proven and h_j be the first verifier message. S_4 prepares the commitment c_j exactly as S does (by committing to the "hash" of program $\Pi_{A,B,j}^{\langle\cdot\rangle}$ under h_j. In addition, it creates the rest of the components, namely $\langle \widetilde{t}_{1,j}, \widetilde{t}_{2,j}\rangle$, α_j, and σ_j exactly as the honest prover algorithm does. It then sends this message to B.

2. If B sends the third message of an *existing* session $j \neq i$, say $\left(r_j,\ (f_j, \widetilde{s}_j, \widetilde{M}_{\lambda_j,s_j}),\ \beta_j,\ \pi_j,\ \sigma'_j\right)$, simulator verifies that (σ_j, π_j) is a valid ZAP proof and $f_j \in \mathcal{F}_n$ and $\widetilde{s}_j$ is in the range of f_j. If so, S_4 sends queries $(f_j, \widetilde{s}_j)$ to the oracle $\mathcal{I}$, and learns the inverse s_j. I.e.,

$$s_j \leftarrow \mathcal{I}\left(f_j, \widetilde{s}_j\right).$$

   S_4 sends the fourth message of session j, denoted $\left(\gamma_j, (\tau_{1,j}, \tau_{2,j}), \pi'_j\right)$ where γ_j is computed honestly, and the rest of the messages are computed using s as the witness as described below.

   (a) S_4 chooses a random $i' \in \{1, 2\}$ and sets: $\tau_{i',j} = \mathsf{Com}(s_j; t_{i',j})$. The other string is a commitment to 0^n. That is, $\tau_{i'',j} = \mathsf{Com}(0^n; t_{i'',j})$ where $i'' = \{1, 2\} \setminus i'$.

   (b) ZAP proof π'_j is computed honestly by using the knowledge of $(i', s_j, t_{i',j}, v_{i',j})$.

3. If the third message corresponds to session $j = i$, the simulator **outputs** the string r_i and halts. If, however, B halts without sending the third message of session i, S_4 **outputs** the *view* of B and halts.

This completes the description of the internal simulator $S_4^{\mathcal{I}}$. □

We note that the polynomials T_{V^*}, T_{Π,V^*}, and Q_{V^*} are all still well defined w.r.t. the four round protocol and simulator $S_4^{\langle\cdot\rangle}$.

The main simulator $S_{\text{main}}^{(4)}(\mathsf{desc}(V^*),\ z\ ;\ \xi)$**.** As before, the main simulator initializes the values $g, \widetilde{\rho}$, and $\mathcal{T}$, and then emulates actions of the twin simulator $S_4^{\langle\cdot\rangle}$. In particular, for a given verifier V^*, it emulates the actions of the program $\Pi_{S_{\text{main}}^{(4)}, V^*, 1^n}$ and computes the simulation trapdoor s_j for every session j by running the obfuscated program $\widetilde{M}_{\lambda_j,s_j}$ for every $j \in [T_{V^*}]$. The only difference from the main simulator S_{main} is that the messages are now prepared according to the specifications of the four round protocol c$\mathcal{ZK}$. We omit a complete description.

Theorem 6.3. *Assume that $\{\mathcal{H}_n\}$ is a family of collision-resistant hash function, $\{\mathcal{F}_n\}$ a family of injective one-way functions,* Com *a non-interactive perfectly binding (string) commitment scheme, and protocols* Π_{WIPOK} *and* Π_{ZAP} *are 3-round-*WIPOK *and* ZAP *systems respectively for* $\mathbf{NP}$*. Further assume that for*

every polynomial $a : \mathbb{N} \to \mathbb{N}$*, and every* hard distribution $\mathcal{Z}$ *over the statements of* $\mathbf{L}^a_{\text{sim}}$*,* $\mathcal{O}$ *is a* $\mathcal{Z}$-auxiliary differing-input obfuscation *(*diO*) for the class of all polynomial-size Turing machines that halt in a polynomial number of steps. Then,* c$\mathcal{ZK}$ *is a four message, fully concurrent zero-knowledge protocol for all languages in* **NP***, with perfect completeness and negligible soundness.*

Proof (sketch). The proof is obtained by following the proof of theorem 5.6. Namely, the same proof and the hybrid experiments are also valid for proving the computational indistinguishability of the output of the main simulator $S^{(4)}_{\text{main}}$ described above. However, one additional argument is required to prove zero-knowledge—in particular to show that in hybrid $G_{m:1}$, the extracted value is indeed the secret s_{i_m} (i.e., a correct inverse).

Let us highlight why this new argument is needed. In the earlier Simple-c$\mathcal{ZK}$ protocol, the soundness of ZK protocol ensured that the extracted value is indeed the correct secret s_{i_m}. However, since we are now using a ZAP proof in the four round protocol (c$\mathcal{ZK}$), we need to argue this claim separately. Below we argue (only) this claim, and the rest of the proof is the same as before.

Suppose that the claim is not true in the hybrid $G_{m:1}$. That is, with noticeable probability, the extracted value s_{i_m} is not a correct inverse of the corresponding values $(f_{i_m}, \widetilde{s}_{i_m})$ that appears in the transcript of session i_m. Recall that in hybrid $G_{m:1}$, all sessions $i_{m'}$ for $m' \geq m$ still use a valid witness in preparing the values $(\tau_{1,i_{m'}}, \tau_{2,i_{m'}})$ and $\pi'_{i_{m'}}$. We consider the following two hybrid experiments:

1. **Hybrid** $G'_{m:1}$**:** same as $G_{m:1}$ except that in the WIPOK proof of session i_m, instead of using values (t_1, t_2, v_1, v_2), the prover uses the real witness w to compute messages (α, β, γ).

2. **Hybrid** $G''_{m:1}$**:** same as $G'_{m:1}$ except that one of the two strings $(\widetilde{t}_1, \widetilde{t}_2)$, say $\widetilde{t}_{i'}$ for a randomly chosen $i' \in \{1, 2\}$ is *incorrectly prepared* by appending 1 (instead of 0) in front of $t_{i'}$; that is: $\widetilde{t}_{i'} = \mathsf{Com}(1 \parallel t_{i'}\ ;\ u_{i'})$. We note that i' is fixed at this step, and in the last step τ_i contains w such that $i \neq i'$. This is necessary since otherwise the hybrid cannot complete the proof successfully.[21]

Now, observe that the output of $G'_{m:1}$ is computationally indistinguishable from that of $G_{m:1}$ due to the WI property of Π_{WIPOK}. Further, $G''_{m:2}$ is computationally indistinguishable from $G'_{m:1}$ due to the security of commitment scheme. This can be seen in three steps as follows. In the first step, the randomness used in the committing $\tau_{i'}$ (in game $G'_{m:1}$ is changed from $t_{i'}$ to a random string; the output of the game even after this change remains computationally indistinguishable from its previous output because this change can be simulated perfectly by taking a commitment from outside challenger who either commits to $0 \parallel t_{i'}$ or $0 \parallel t^*$ for a random string t^*. In the second step, string $\widetilde{t}_{i'}$ is changed to a commitment to $1 \parallel t_{i'}$ instead of $0 \parallel t_{i'}$ and this keeps the output of the game computationally indistinguishable, since, as before this change can be perfectly simulated by receiving string $\widetilde{t}_{i'}$ from an outside challenger who either commits to $0 \parallel t_{i'}$ or $1 \parallel t_{i'}$. In the final step, the randomness of $\tau_{i'}$ is changed back from uniform (i.e., t^*) to $t_{i'}$ and indistinguishability is argued as in the first step. Hence, the outputs of $G'_{m:1}$ and $G''_{m:2}$ are computationally indistinguishable.

Therefore, it follows that if s_{i_m} is not a correct inverse in $G_{m:1}$, then the same holds for it in hybrid $G''_{m:1}$. However, the soundness of ZAP now implies that the statement represented by $(\widetilde{t}_1, \widetilde{t}_2)$ is false and thus the obfuscation must have been correctly computed. Therefore, in G''_{i_m}, the value computed by $S^{(4)}_{\text{main}}$ by running the obfuscated code must indeed yield a correct inverse s_{i_m}. This is a contradiction. We conclude that the outputs of the obfuscated programs in game $G_{m:1}$ must yield correct inverses as desired. ∎

[21] We also note that this is in fact identical to first choosing i randomly (to be used in the last step to set up τ_i) and then choosing (uniquely defined) $i' \in \{1, 2\} \setminus i$ (or equivalently $i \neq i'$) for defining $\widetilde{t}_{i'}$ as described above.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013, 2013.

[App13] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. Cryptology ePrint Archive, Report 2013/699, 2013. `http://eprint.iacr.org/2013/699.pdf`.

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *FOCS*, 2002.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *TCC*, 2014. Preliminary version on Eprint 2013: `http://eprint.iacr.org/2013/668.pdf`.

[BC10] Nir Bitansky and Ran Canetti. On strong simulation and composable point obfuscation. In *CRYPTO*, pages 520–537, 2010.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. Extractable obfuscation and applications. In *TCC*, 2014. Preliminary version on Eprint 2013: `http://eprint.iacr.org/2013/650.pdf`.

[BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. More on the impossibility of virtual-black-box obfuscation with auxiliary input. Cryptology ePrint Archive, Report 2013/701, 2013. `http://eprint.iacr.org/2013/701.pdf`.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1992.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Annual IEEE Conference on Computational Complexity (CCC)*, volume 17, 2002. Preliminary full version available as Cryptology ePrint Archive, Report 2001/105.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS 2001*, pages 116–125, 2001.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Crypto '01*, pages 1–18, 2001.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. *IACR Cryptology ePrint Archive*, 2013:631, 2013.

[BL04] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. *SIAM Journal on Computing*, 33(4):783–818, August 2004. Extended abstract appeared in STOC 2002.

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

[BOV03] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In *CRYPTO*, pages 299–315, 2003.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO*, pages 273–289, 2004.

[BP12a] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, pages 223–232, 2012.

[BP12b] Nir Bitansky and Omer Paneth. Point obfuscation and 3-round zero-knowledge. In *TCC*, pages 190–208, 2012.

[BP13a] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, pages 241–250, 2013.

[BP13b] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. Cryptology ePrint Archive, Report 2013/703, 2013. `http://eprint.iacr.org/2013/703.pdf`.

[BR13a] Zvika Brakerski and Guy N. Rothblum. Obfuscating conjunctions. In *CRYPTO (2)*, pages 416–434, 2013.

[BR13b] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. `http://eprint.iacr.org/2013/563.pdf`.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/642, 2013. `http://eprint.iacr.org/`.

[CD09] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In *TCC*, pages 595–613, 2009.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proc. 32th STOC*, pages 235–244, 2000.

[CKPR03] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. *SIAM Journal on Computing*, 32(1):1–47, February 2003. Preliminary version in STOC '01.

[CLP13a] Ran Canetti, Huijia Lin, and Omer Paneth. Public-coin concurrent zero-knowledge in the global hash model. In *TCC*, pages 80–99, 2013.

[CLP13b] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from p-certificates. In *FOCS*, 2013.

[COPV13] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, and Ivan Visconti. Simultaneous resettability from one-way functions. In *FOCS*, pages 231–240, 2013.

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In *STOC*, pages 231–240, 2013.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *TCC*, pages 72–89, 2010.

[CV13] Ran Canetti and Vinod Vaikuntanathan. Obfuscating branching programs using black-box pseudo-free groups. *IACR Cryptology ePrint Archive*, 2013:500, 2013.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, 2009.

[DL07] Yi Deng and Dongdai Lin. Instance-dependent verifiable random functions and their application to simultaneous resettability. In *EUROCRYPT*, pages 148–168, 2007.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *Proc.* 41*st FOCS*, pages 283–293, 2000.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc.* 30*th STOC*, pages 409–418, 1998.

[DS98] Cynthia Dwork and Amit Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In *CRYPTO*, pages 442–457, 1998.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988. Preliminary version in STOC 1987.

[FLS99] Feige, Lapidot, and Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29, 1999.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–545, 1989.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proc.* 22*nd STOC*, pages 416–426, 1990.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GJO+13] Vipul Goyal, Abhishek Jain, Rafail Ostrovsky, Silas Richelson, and Ivan Visconti. Concurrent zero knowledge in the bounded player model. In *TCC*, pages 60–79, 2013.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, February 1996. Preliminary version appeared in ICALP' 90.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GK13] Shafi Goldwasser and Yael Tauman Kalai. A note on the impossibility of obfuscation with auxiliary input. Cryptology ePrint Archive, Report 2013/665, 2013. `http://eprint.iacr.org/2013/665.pdf`.

[GM11] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In *FOCS*, pages 678–687, 2011.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc.* 17*th STOC*, pages 291–304, Providence, 1985. ACM.

[Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *Proc.* 34*th STOC*, pages 332–340, 2002.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, pages 97–111, 2006.

[Goy13] Vipul Goyal. Non-black-box simulation in the fully concurrent setting. In *STOC*, pages 221–230, 2013.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[GS12a] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *CRYPTO*, pages 105–123, 2012.

[GS12b] Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. *CoRR*, abs/1210.3719, 2012.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In *AsiaCrypt '00*, pages 443–457, 2000.

[Had10] Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT*, pages 92–112, 2010.

[HR04] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *CRYPTO*, pages 92–105, 2004.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC*, pages 233–252, 2007.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/509, 2013. `http://eprint.iacr.org/2013/509.pdf`.

[HT99] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. Cryptology ePrint Archive, Report 1999/009, 1999. `http://eprint.iacr.org/`.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc.* 24*th STOC*, pages 723–732, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In *Crypto '95*, pages 311–324, 1995.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *STOC*, pages 560–569, 2001.

[KPR98] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the Internet. In *Proc.* 39*th FOCS*, pages 484–492, 1998.

[KRW13] Venkata Koppula, Kim Ramchen, and Brent Waters. Separations in circular security for arbitrary length key cycles. Cryptology ePrint Archive, Report 2013/683, 2013. `http://eprint.iacr.org/2013/683.pdf`.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO*, pages 171–189, 2001.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.

[Mic94] S. Micali. CS proofs. In *Proc.* 35*th FOCS*, pages 436–453, 1994.

[MO13] Antonio Marcedone and Claudio Orlandi. Obfuscation ==¿ (ind-cpa security =/=¿ circular security). Cryptology ePrint Archive, Report 2013/690, 2013. `http://eprint.iacr.org/2013/690.pdf`.

[MR13] Tal Moran and Alon Rosen. There is no indistinguishability obfuscation in pessiland. Cryptology ePrint Archive, Report 2013/643, 2013. `http://eprint.iacr.org/2013/643.pdf`.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness (extended abstract). In *CRYPTO*, pages 128–136, 1989.

[Pan14] Omkant Pandey. Achieving constant round leakage-resilient zero-knowledge. In *TCC*, 2014. Preliminary version on Eprint 2012: `http://eprint.iacr.org/2012/362.pdf`.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *Eurocrypt '03*, 2003.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. 36th STOC*, pages 232–241, 2004.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *Proc. 44th FOCS*, 2003.

[PR05a] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, 2005.

[PR05b] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC*, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, 2002.

[PTV10] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Eye for an eye: Efficient concurrent zero-knowledge in the timing model. In *TCC*, pages 518–534, 2010.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *TCC*, pages 553–570, 2008.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.

[Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *Crypto '00*, pages 451–468, 2000.

[Ros04] Alon Rosen. *The Round-Complexity of Black-Box Concurrent Zero-Knowledge*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 2004.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *IACR Cryptology ePrint Archive*, 2013:454, 2013.

[TW87] M. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. In *Proc. 28th FOCS*, pages 472–482, 1987.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.