# Adaptive Zero-Knowledge Proofs and Adaptively Secure Oblivious Transfer*


Yehuda Lindell Hila Zarosim

Dept. of Computer Science
Bar-Ilan University, Israel
{lindell,zarosih}@cs.biu.ac.il


July 22, 2009


**Abstract**

In the setting of secure computation, a set of parties wish to securely compute some function of their inputs, in the presence of an adversary. The adversary in question may be *static* (meaning that it controls a predetermined subset of the parties) or *adaptive* (meaning that it can choose to corrupt parties during the protocol execution and based on what it sees). In this paper, we study two fundamental questions relating to the basic zero-knowledge and oblivious transfer protocol problems:

- *Adaptive zero-knowledge proofs:* We ask whether it is possible to construct adaptive zero-knowledge *proofs* (with unconditional soundness). Beaver (STOC 1996) showed that known zero-knowledge proofs are not adaptively secure, and in addition showed how to construct zero-knowledge arguments (with computational soundness).
- *Adaptively secure oblivious transfer:* All known protocols for adaptively secure oblivious transfer rely on seemingly stronger hardness assumptions than for the case of static adversaries. We ask whether this is inherent, and in particular, whether it is possible to construct adaptively secure oblivious transfer from enhanced trapdoor permutations alone.

We provide surprising answers to the above questions, showing that achieving adaptive security is sometimes harder than achieving static security, and sometimes not. First, we show that assuming the existence of one-way functions only, there exist adaptive zero-knowledge proofs for all languages in $\mathcal{NP}$. In order to prove this, we overcome the problem that all adaptive zero-knowledge protocols known until now used equivocal commitments (which would enable an all-powerful prover to cheat). Second, we prove a black-box separation between adaptively secure oblivious transfer and enhanced trapdoor permutations. As a corollary, we derive a black-box separation between adaptively and statically securely oblivious transfer. This is the first black-box separation to relate to adaptive security and thus the first evidence that it is indeed harder to achieve security in the presence of adaptive adversaries than in the presence of static adversaries.



*This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 781/07).

# 1 Introduction

In the setting of secure two-party and multiparty computation, parties with private inputs wish to securely compute some joint function of their inputs, where "security" must hold in the presence of adversarial behavior by some of the parties. An important parameter in any definition of security relates to the adversary's power. Is the adversary computationally bounded or all powerful? Is the adversary semi-honest (meaning that it follows all protocol instructions but tries to learn more than it's supposed to by analyzing the messages it receives) or is it malicious (meaning that it can arbitrarily deviate from the protocol specification)? Finally, are the adversarial corruptions *static* (meaning that the set of corrupted parties is fixed) or *adaptive* (meaning that the adversary can corrupt parties throughout the computation and the question of who to corrupt and when may depend on the adversary's view in the protocol execution). It is desirable to achieve security in the presence of adaptive adversaries where possible, since it models the real-world phenomenon of "hackers" actively breaking into computers, possibly while they are executing secure protocols. However, it seems to be technically harder to achieve security in the presence of adaptive adversaries. Among other things, it requires the ability to construct a simulator who can first generate a transcript blindly (without knowing any party's input) and then later, upon receiving inputs, "explain" the transcript as an execution of honest parties with those inputs.

In this paper, we ask two basic questions related to the feasibility of achieving security in the presence of adaptive adversaries. Our questions were borne out of the following two observations:

1. *Adaptive zero-knowledge proofs:* It has been shown that the zero-knowledge proof system of [23] (and all others known) is not secure in the presence of adaptive adversaries, or else the polynomial hierarchy collapses [1]. Due to this result, all known zero-knowledge protocols for $\mathcal{NP}$ in the adaptive setting are *arguments*, meaning that soundness only holds in the presence of a polynomial-time prover (adaptive zero-knowledge arguments were presented by [1] and later in the context of universal composability; e.g., see [8, 11]). However, the question of whether or not adaptive zero-knowledge *proofs* exist for all $\mathcal{NP}$ has not been addressed.

2. *Adaptively secure oblivious transfer:* One of the goals of the theory of cryptography is to understand what assumptions are necessary and sufficient for carrying out cryptographic tasks; see for example [24]. Despite this, no such study has been carried out regarding adaptively secure protocols. In particular, we do not know what assumptions are necessary for achieving adaptively secure oblivious transfer (since oblivious transfer is complete for secure computation, this question has important ramifications to adaptively secure computation in general). Currently, what is known is that although statically secure oblivious transfer can be constructed from enhanced trapdoor permutations [16, 22], all constructions for adaptively secure oblivious transfer use additional assumptions like the ability to sample a permutation without knowing its trapdoor [3, 11].

**Our results – adaptive zero-knowledge proofs.** All known zero-knowledge protocols for $\mathcal{NP}$ essentially follow the same paradigm: the prover sends the verifier commitments that are based on the statement being proved (and its witness), and the verifier then asks the prover to open part or all of the commitments. Based on the prover's answer, the verifier is either convinced that the statement is true or detects the prover cheating. It therefore follows that soundness only holds if the commitment scheme used is *binding*, and this is a problem in the setting of adaptive security. Consider an adversary that corrupts the verifier at the beginning of the execution and the prover at the end. In this case, the zero-knowledge simulator must generate a transcript without knowing the NP-witness. However, at the end, after the prover is corrupted (and the simulator then receives a witness), it must be able to show that the commitments were generated using that witness. Until now, this has been solved by using *equivocal* commitments that can be opened to any value desired (in order for soundness to hold, the ability to equivocate is given to the simulator and not the real prover). However, this means that the protocol has only computational soundness, because an all-powerful prover is able to equivocate like the simulator. Indeed, the above observation led us to initially conjecture that adaptive zero-knowledge proofs exist only for $\mathcal{SZK}$. However, our conjecture was wrong, and in this paper we prove the following theorem:

**Theorem 1** *Assuming the existence of one-way functions that are hard to invert for non-uniform adversaries, there exist adaptive zero-knowledge proofs for all* $\mathcal{NP}$.

We prove Theorem 1 by constructing a new type of *instance-dependent* commitment scheme. Instance-dependent commitment schemes are commitments whose properties depend on whether the instance (or statement) in question is in the language or not [4, 26]. Typically, they are defined for a language $L$ as follows. Let $x$ be a statement. If $x \in L$ then the commitment associated with $x$ is computationally hiding and if $x \notin L$ then the commitment associated with $x$ is perfectly binding. This has proven very useful in the context of zero-knowledge where hiding alone is needed for the case of $x \in L$, and binding alone is needed in the case of $x \notin L$; see for example [32, 40, 33]. We construct an instance-dependent commitment scheme with the additional property that if $x \in L$ then the commitment is *equivocal* and the simulator can open it to any value it wishes. To be more exact, we need the commitment itself to be *adaptively secure*, meaning that it must be possible to generate a commitment value $c$ and then later find "random coins" $r$ for any bit $b$ so that $c$ is a commitment string generated by an honest committer with input $b$ and random coins $r$.[1] In contrast to the above, if $x \notin L$ then the commitment is still perfectly binding. Given such a commitment (which is actually very similar to the commitment schemes presented in [17] and [11]) we are able to construct the first computational zero-knowledge *proof* for all $\mathcal{NP}$ that is secure also in the case of adaptive corruptions.[2]

**Our results – adaptively secure oblivious transfer.** As we have mentioned, all known protocols for adaptively secure oblivious transfer require assumptions of the flavor that it is possible to sample a permutation without its trapdoor. In contrast, standard trapdoor permutations do not have this property. We remark that enhanced trapdoor permutations do have the property that it is possible to sample an element in the domain of the permutation without knowing its preimage. This begs the question as to whether such "oblivious sampling" of the permutation's domain suffices for achieving adaptively secure oblivious transfer, or is something stronger needed (like oblivious sampling of permutations themselves). We remark that oblivious sampling is used in this context by having the simulator sample unobliviously and then "lie" in its final transcript by claiming to have sampled in the regular way. However, this strategy is problematic when the oblivious sampling is carried out on elements in the domain because if the trapdoor is known then it may be possible to see if the preimage of the sampled value appears implicitly in the protocol

[1] We stress that this is a strictly stronger requirement than equivocality. In most equivocal commitments, the committer reveals only some of its coins upon decommitting. This does not suffice for achieving adaptive commitments.

[2] In [33], adaptively secure commitment schemes were constructed for the languages of Graph Isomorphism and Quadratic Residuosity (although they were not presented in this way nor for this purpose). The constructions in [33] are incomparable to ours. On the one hand, they require no hardness assumptions whereas we use one-way functions. On the other hand, our construction is for all languages in $\mathcal{NP}$ whereas they are restricted to the above two specific languages (which are also in $\mathcal{SZK}$).

transcript. (For example, in the protocol of [16], the preimages fully define the sender's input and so if the trapdoor is known, the values can be checked.) Of course, such arguments do not constitute any form of evidence. In order to demonstrate hardness, we use the methodology of black-box separations, introduced by [25] and later used in [39, 29, 19, 38] amongst others. We prove the following informally stated theorem:

**Theorem 2** *There exists an oracle relative to which enhanced trapdoor permutations exist but adaptively secure oblivious transfer does not exist.*

Recalling that statically secure oblivious transfer can be constructed from any enhanced trapdoor permutation in a black-box way [16, 22], we obtain the following corollary:

**Corollary 3** *There exists an oracle relative to which statically secure oblivious transfer exists but adaptively secure oblivious transfer does not exist.*

This is the first evidence that it is strictly harder to achieve security in the presence of adaptive adversaries than to achieve security in the presence of static adversaries. We prove Theorem 2 by showing that if it is possible to achieve adaptively secure oblivious transfer using only enhanced trapdoor permutations, then it is possible to achieve statically secure oblivious transfer using only symmetric encryption (this is very inexact but sufficient for intuition). We then show that statically secure oblivious transfer does not exist relative to most symmetric encryption oracles. In order to prove this, we use the recent result of [12] that shows the equivalence of the random oracle and ideal cipher models, to replace a symmetric encryption oracle by a "plain" random oracle (using six rounds of the Luby-Rackoff construction [30])[3]. This enables us to extend the black-box separation of [25] to show that key agreement does not exist relative to most symmetric encryption oracles. (Indeed this is a novel interpretation of that result and it means that all black-box separations with random oracles hold also with symmetric encryption). We conclude the proof by recalling that key agreement can be constructed from oblivious transfer [19]. Thus, adaptively secure oblivious transfer cannot be constructed in a black-box way from enhanced trapdoor permutations. We remark that all of our results for oblivious transfer are proven for semi-honest adversaries (and thus hold also for malicious adversaries).

Our proof makes no explicit use of the fact that the functionality being computed is oblivious transfer and holds for any functionality. We conclude that either a given function can be securely computed statically assuming only the existence of one-way functions (or to be more exact, only given a "symmetric" random oracle), or enhanced trapdoor permutations do not suffice for computing it with adaptive security.

**Related Work.** Instance dependent commitment schemes were first implicitly used in [4] to construct a constant round zero-knowledge proof for the language of Graph Isomorphism. In [26] they follow these ideas and explicitly define instance-dependent commitment schemes[4] as commitment schemes where both the sender and the receiver receive an instance $x \in \{0,1\}^*$ and the binding and the hiding properties depend on whether $x \in L$ or not. Since then, a great deal of work has been carried out to investigate the relationship between zero-knowledge proofs and instance-dependent commitment schemes (see [40, 32, 35, 27, 13, 14]) and finally in the recent work of [36] it was shown

[3]The reason that 6 rounds are needed and not 4 (as in the construction of pseudorandom permutations from pseudorandom functions) is due to the fact that the distinguisher has access to the intermediate values

[4]Actually, in [26], they use the term "language-dependent cryptographic primitives".

that instance-dependent commitment schemes are necessary and sufficient for constructing zero-knowledge proofs. It is interesting to note that in contrast to regular commitment schemes, where the commitment cannot be both statistically hiding and statistically binding, instance-dependent commitment schemes that are statistically hiding when $x \in L$ and statistically binding when $x \notin L$ exist for certain languages (such as Graph Isomorphism).

The method of proving black box separations between cryptographic primitives, as a way to conclude that it is not likely that the existence of a certain cryptographic primitive implies the existence of another primitive, was first introduced in the seminal work of Impagliazzo and Rudich [25]. In their work, they consider a world where all parties have access to a random permutation oracle, which is provably one-way in the strongest sense. On the other hand, they show that if $\mathcal{P} = \mathcal{NP}$, there doesn't exists a secure key agreement protocol (even relative to such a random oracle). This result has the following consequence: There is an oracle relative to which one way permutation exists but key agreement does not exist (This oracle is constructed from the random oracle and a $\mathcal{PSPACE}$-complete problem). This method of proving black-box separation was subsequently used in many other works (see [37, 39, 29, 28, 21, 18, 31, 10, 20]).

Adaptive zero-knowledge arguments were constructed in [1] and later adaptively secure universally composable zero-knowledge arguments were constructed in [8, 11]. Adaptively secure oblivious transfer was constructed in [3] using trapdoor permutations with the additional property that it is possible to select a permutation without knowing its trapdoor, and in [11] from non-committing encryptions (see [9, 2, 15]). All constructions of adaptively secure oblivious transfer (and non-committing encryptions) require the possibility to sample a permutation without its trapdoor. In this sense, our result is a complementary result to these construction as we prove that enhanced trapdoor permutations alone are not sufficient for constructing an adaptively secure oblivious transfer in a black-box manner.

In [12] it is shown that the random oracle and the ideal cipher models are equivalent. Our techniques imply a new application for this result: all black-box separations that have been proven relative to a random oracle also hold relative to an ideal cipher (or a symmetric encryption oracle).

# 2 Preliminaries and Definitions

## 2.1 Preliminaries and Notations

We let $n$ denote the security parameter. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large $n$ it holds that $\mu(n) < \frac{1}{p(n)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. For an $\mathcal{NP}$ relation $R$, we denote by $R_x$ the set of witnesses of $x$ and by $L_R$ its associated language. That is, $R_x = \{w | \ (x, w) \in R\}$ and $L_R = \{x | \ \exists w \ \ (x, w) \in R\}$.

Let $\langle A, B\rangle$ be an interactive protocol. $\langle A(x_A; r_A), B(x_B; r_B)\rangle$ denotes the joint output of $A$ and $B$, where the input of $A$ is $x_A$ and its random tape is $r_A$ and the input of $B$ is $x_B$ and its random tape is $r_B$. $\langle A(x_A), B(x_B)\rangle$ denotes the random variable describing $\langle A(x_A; r_A), B(x_B; r_B)\rangle$ where the random tapes of the parties are chosen uniformly. The view here contains the oracle replies as well.

$VIEW_A^{\Pi}(x_A, x_B)$ is a random variable describing the view of $A$ in an execution of protocol $\Pi$ on inputs $x_A$ and $x_B$, where the random tapes are chosen uniformly. The view of a party includes its input, random tape and the messages the party received. For an oracle $O$, $VIEW_A^{\Pi,O}(x_A, x_B)$ describes the view of $A$ in an execution of protocol $\Pi$ on inputs $x_A$ and $x_B$ with oracle access to $O$, where the random tapes are chosen uniformly. For a distribution $\mathcal{D}$ on oracles, $VIEW_A^{\Pi,\mathcal{D}}(x_A, x_B)$

describes the view of $A$ in an execution of protocol $\Pi$ on inputs $x_A$ and $x_B$ where the random tapes are chosen uniformly and the parties have access to an oracle that was chosen according to $\mathcal{D}$.

**Definition 2.1** *Let $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ be two distribution ensembles. We say that $X$ and $Y$ are* computationally indistinguishable, *denoted $X \stackrel{c}{\equiv} Y$, if for every PPT machine $D$, every positive polynomial $p(\cdot)$ and all sufficiently large $n$:*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

**Definition 2.2** *Let $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ be two distribution ensembles. We say that $X$ and $Y$ are* computationally indistinguishable relative to an oracle $O$, *denoted $X \stackrel{cO}{\equiv} Y$, if for every PPT oracle machine $D$, every positive polynomial $p(\cdot)$ and all sufficiently large $n$:*

$$\left|\Pr\left[D^O(X_n, 1^n) = 1\right] - \Pr\left[D^O(Y_n, 1^n) = 1\right]\right| < \frac{1}{p(n)}$$

We sometimes use the same notation when we refer to a distribution $\mathcal{D}$ on oracles rather than a single oracle $O$. We now present definitions for semi-honest security for static adversaries. This will be needed in our proof in Section 4.

**Definition 2.3** *Let $\langle S, R\rangle$ be an interactive protocol, where the input of $S$ is a pair of strings $s_0, s_1 \in \{0,1\}^k$ (where $k = poly(n)$) and the input of $R$ is a bit $\sigma$, and let $n$ be the security parameter. We say that $\langle S, R\rangle$* computes the $OT_1^2$ functionality[5] *if there exists a negligible function $neg(\cdot)$ such that for all $n$, for every $s_0, s_1 \in \{0,1\}^k, \sigma \in \{0,1\}$:*

$$\Pr[\langle S(1^n, s_0, s_1), R(1^n, \sigma)\rangle = (\lambda, s_\sigma)] \geq 1 - neg(n)$$

*We say that $\langle S^O, R^O\rangle$ computes the $OT_1^2$ functionality relative to an oracle $O$ if there exists a negligible function $neg(\cdot)$ such that for every $n$ and every $s_0, s_1, \sigma$:*

$$\Pr\left[\langle S^O(1^n, s_0, s_1), R^O(1^n, \sigma)\rangle = (\lambda, s_\sigma)\right] \geq 1 - neg(n)$$

**Definition 2.4** *Let $\Pi = \langle S, R\rangle$ be a protocol for computing the $OT_1^2$ functionality. We say that $\Pi$* securely computes the $OT_1^2$ functionality in the presence of static semi-honest adversaries *if there exist two probabilistic polynomial-time algorithms $\mathcal{S}_S$ and $\mathcal{S}_R$ such that:*

$$\{\mathcal{S}_S(1^n, s_0, s_1)\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}} \stackrel{c}{\equiv} \left\{VIEW_S^{\Pi}(1^n, s_0, s_1, \sigma)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}}$$

$$\{\mathcal{S}_R(1^n, \sigma, s_\sigma)\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}} \stackrel{c}{\equiv} \left\{VIEW_R^{\Pi}(1^n, s_0, s_1, \sigma)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}}$$

*Let $\Pi = \langle S^O, R^O\rangle$ be a protocol for computing the $OT_1^2$ functionality relative to an oracle $O$. We say that $\Pi$* securely computes the $OT_1^2$ functionality relative to an oracle $O$ in the presence of static semi-honest adversaries *if there exist two PPT oracle machines $\mathcal{S}_S$ and $\mathcal{S}_R$ such that:*

$$\left\{\mathcal{S}_S^O(1^n, s_0, s_1)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}} \stackrel{cO}{\equiv} \left\{VIEW_S^{\Pi,O}(1^n, s_0, s_1, \sigma)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}}$$

$$\left\{\mathcal{S}_R^O(1^n, \sigma, s_\sigma)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}} \stackrel{cO}{\equiv} \left\{VIEW_R^{\Pi,O}(1^n, s_0, s_1, \sigma)\right\}_{s_0,s_1\in\{0,1\}^k,\sigma\in\{0,1\}}$$

[5]For $s_0, s_1 \in \{0,1\}^k$ and $\sigma \in \{0,1\}$, the functionality *1-out-of-2 Oblivious Transfer*, denoted $OT_1^2$, is defined as: $OT_1^2((s_0, s_1), \sigma) = (\lambda, s_\sigma)$.

## 2.2 Adaptive Security- Definitions

In this section we provide a definition for adaptive security of two party protocols (for a deterministic functionality $f$). The definition is taken from [7]. Adjustments of the definition for the special cases of adaptive zero-knowledge proofs (for malicious adversaries) and adaptively secure oblivious transfer (for semi-honest adversaries) will be described afterwards.

**The real-life model:** Each party $P_i$ begins with an input $x_i \in \{0,1\}^*$, a random tape $r_i$ and the security parameter $n$. An *adaptive real-life adversary* $\mathcal{A}$ is a probabilistic polynomial-time interactive Turing machine that starts with a random tape $r_A$ and security parameter $n$. The environment $\mathcal{Z}$ is another probabilistic polynomial-time interactive Turing machine that starts with an input $z$, a random tape $r_Z$ and the security parameter $n$.

At the outset of the protocol, $\mathcal{A}$ receives some initial information from $\mathcal{Z}$. Next the computation continues in rounds. Before each round, if there exists an uncorrupted party, the adversary $\mathcal{A}$ might choose to corrupt one of the parties or both. Next, $\mathcal{A}$ activates the party that is supposed to be active in this round according to the protocol. At each round, $\mathcal{A}$ sees all messages sent by the parties (that is, the conversation between the parties is visible to the adversary).

Upon corrupting a party, the adversary learns its input and its random tape. In addition, $\mathcal{Z}$ learns the identity of the corrupted party and hands some auxiliary information to $\mathcal{A}$. If the adversary is malicious, once a party is corrupted, it follows the adversary's instructions from this point. If the adversary is semi-honest, the corrupted party continues following the protocol.

At the end of the computation, the parties locally generate their outputs. Uncorrupted parties output their output as specified by the protocol and corrupted parties output a special symbol $\perp$. In addition the adversary outputs an arbitrary function of its internal state. (Without loss of generality, this output consists of all the information seen in the execution: the random tape $r_A$, the information received from the environment and the corrupted parties' views of the execution.)

Next, a postexecution corruption process begins. $\mathcal{Z}$ learns the outputs. Next, $\mathcal{Z}$ and $\mathcal{A}$ interact in at most two rounds, where in each round $\mathcal{Z}$ can generate a "corrupt $P_1$" or "corrupt $P_2$" message and hand it to $\mathcal{A}$. Upon receipt of this message, $\mathcal{A}$ hands $\mathcal{Z}$ the internal state of the party.At the end of this process, $\mathcal{Z}$ outputs its entire view of the interaction with the parties and $\mathcal{A}$.

Let $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(n, x_1, x_2, z, \overrightarrow{r})$, where $\overrightarrow{r} = (r_Z, r_A, r_1, r_2)$, denote $\mathcal{Z}$'s output on input $z$, random tape $r_Z$ and security parameter $n$ after interacting with adversary $\mathcal{A}$ and parties $P_1$ and $P_2$ running protocol $\Pi$ on inputs $x_1, x_2$, random input $\overrightarrow{r}$ and security parameter $n$. Let $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(n, x_1, x_2, z)$ denote the random variable describing $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(n, x_1, x_2, z, \overrightarrow{r})$ when the random tapes of the parties $r_Z, r_A, r_1, r_2$ are chosen uniformly. Let $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble

$$\{\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(n, x_1, x_2, z)\}_{x_1,x_2,z\in\{0,1\}^*,n\in\mathbb{N}}$$

**The ideal process-adaptive model:** Each party $P_i$ has input $x_i$ and no random tape is needed. An *adaptive ideal-process adversary* $\mathcal{SIM}$ is a probabilistic polynomial-time interactive Turing machine that starts with a random tape $r_{\mathcal{SIM}}$ and the security parameter $n$. The environment $\mathcal{Z}$ is another probabilistic polynomial-time interactive Turing machine that starts with an input $z$, a random tape $r_Z$ and the security parameter $n$. In addition, there is an incorruptible trusted party $\mathcal{T}$. The ideal process proceeds as follows:

**First corruption stage:** First, $\mathcal{SIM}$ receives some auxiliary information from $\mathcal{Z}$. Next, $\mathcal{SIM}$ proceeds in at most two iterations, where in each iteration $\mathcal{SIM}$ may decide to corrupt one of the parties. Once a party is corrupted, its input becomes known to $\mathcal{SIM}$. In addition, $\mathcal{Z}$ learns the identity of the corrupted party and hands some auxiliary information to $\mathcal{SIM}$.

**Computation stage:** Uncorrupted parties hand the inputs to $\mathcal{T}$. In the malicious settings, corrupted parties hand values chosen by $\mathcal{SIM}$ to $\mathcal{T}$. In the semi-honest setting, corrupted parties hand their inputs to $\mathcal{T}$. Let $y_1, y_2$ be the values handed to $\mathcal{T}$. $\mathcal{T}$ computes $f(y_1, y_2)$ and hands $P_1$ the value $f(y_1, y_1)_1$ and $P_2$ the value $f(y_1, y_2)_2$.

**Second corruption stage:** $\mathcal{SIM}$ continues in another sequence of at most two iterations, where in each iteration, $\mathcal{SIM}$ might choose to corrupt one of the parties based on its random tape and the information gathered so far. Once a party is corrupted, $\mathcal{SIM}$ learns its input, $\mathcal{Z}$ learns the identity of the corrupted party and hands $\mathcal{SIM}$ some auxiliary information.

**Output:** Each uncorrupted party $P_i$ outputs $f(y_1, y_2)_i$. Corrupted parties output a special symbol $\perp$. The adversary $\mathcal{SIM}$ outputs an arbitrary function of its internal state. $\mathcal{Z}$ learns all outputs.

**Postexecution corruption:** After the outputs are generated, $\mathcal{SIM}$ proceeds in at most two rounds with $\mathcal{Z}$, where in each round, $\mathcal{Z}$ can generate a "corrupt $P_i$" message and hand it to $\mathcal{SIM}$. For any such request, $\mathcal{SIM}$ generates some arbitrary answer and it might choose to corrupt any of the parties. The interaction continues until $\mathcal{Z}$ halts with an output.

Let $\text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}(n, x_1, x_2, z, \overrightarrow{r})$, where $\overrightarrow{r} = (r_Z, r_{\mathcal{SIM}})$, denote the output of $\mathcal{Z}$ on input $z$, random input $r_Z$ and security parameter $n$ after interacting with an ideal-process adversary $\mathcal{SIM}$ with random input $r_{\mathcal{SIM}}$, with parties having inputs $(x_1, x_2)$ and with a trusted party $\mathcal{T}$ for evaluating the functionality $f$. Let $\text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}(n, x_1, x_2, z)$ denote the random variable describing $\text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}(n, x_1, x_2, z, \overrightarrow{r})$ where $r_Z, r_{\mathcal{SIM}}$ are chosen uniformly. Let $\text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}$ denote the distribution ensemble

$$\{\text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}(n, x_1, x_2, z)\}_{x_1,x_2,z\in\{0,1\}^*,n\in\mathbb{N}}$$

**Definition 2.5** *Let* $\Pi$ *be a protocol for computing a functionality* $f$*. We say that* $\Pi$ securely computes the functionality $f$ in the presence of adaptive adversaries *if for every probabilistic polynomial-time adaptive real-life adversary* $\mathcal{A}$ *and every environment* $\mathcal{Z}$*, there exists a probabilistic polynomial-time adaptive ideal-process adversary* $\mathcal{SIM}$*, such that:*

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\equiv} \text{IDEAL}_{f,\mathcal{SIM},\mathcal{Z}}$$

*If the adversary* $\mathcal{A}$ *and the simulator* $\mathcal{SIM}$ *are restricted to semi-honest behavior, then we say that* $\Pi$ securely computes the functionality $f$ in the presence of semi-honest adaptive adversaries.

**A special case – adaptive zero-knowledge:** When considering zero-knowledge as a special case of secure computation, it is most natural to define an adaptive zero knowledge *proof of knowledge* functionality of the form $f_R((x, w), \lambda) = (\lambda, (x, b))$ where $b = 1$ if $R(x, w) = 1$ and $b = 0$ if $R(x, w) = 0$. However, since the goal of our work is to deal with the fundamental question of the feasibility or infeasibility of adaptive zero-knowledge proofs (as asked by Beaver [1]), we present an alternative definition that is more in line with the standard setting of zero-knowledge proof systems (that are not necessarily proofs of knowledge). The advantage of this approach is that it simplifies the proof and allows us to focus on the main issue of constructing an adaptive proof (rather than an argument), without dealing with knowledge extraction.

Recall that in the standard setting of zero-knowledge, indistinguishability of the real world from the ideal world is only required for instances $x \in L$. For these instances the trusted party always returns 1, and we can therefore omit the trusted party altogether from the ideal world.

In this case the real-life model is as defined above where the input of the verifier is an instance $x \in \{0,1\}^n$ (where $n$ is the security parameter) and the input of the prover is a pair $(x, w) \in \{0,1\}^n \times \{0,1\}^{p(n)}$ for a polynomial $p(\cdot)$. The output of the uncorrupted prover is the empty string $\lambda$ and the output of the uncorrupted verifier is a bit specified by the protocol.

In the ideal process, the ideal process adversary $\mathcal{SIM}$ receives the instance $x$ that is guaranteed to be in the language as input and interacts with the environment and corrupted parties. Thus, we only need 3 stages: first corruption stage, output stage and postexecution corruption stage (since there is no computation stage, there is also no need for a second corruption stage). We also allow the simulator to run in expected polynomial time rather than strict polynomial time (we do not know how to construct a strict polynomial-time simulator for our protocol even though it has a nonconstant number of rounds).

The distribution $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denotes the distribution ensemble

$$\{\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(x, w, z)\}_{x \in L, w \in R_x, z \in \{0,1\}^*}$$

and $\text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}$ denotes the distribution ensemble

$$\{\text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}(x, w, z)\}_{x \in L, w \in R_x, z \in \{0,1\}^*}$$

.

**Definition 2.6** *Let $L$ be a language. We say that $\langle P, V \rangle$ is an* adaptive zero-knowledge proof system (AZK) *for $L$ if $\langle P, V \rangle$ is an interactive proof system for $L$ and for any PPT real-life adversary $\mathcal{A}$ and any PPT environment $\mathcal{Z}$, there exists an expected polynomial-time probabilistic adaptive ideal-process adversary $\mathcal{SIM}$, such that*

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\equiv} \text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}$$

**Adaptive security relative to oracles:** In this work we consider security relative to oracles. We therefore consider two distributions describing the real-life and the ideal-life process relative to an oracle. In this case, all parties as well as the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ are PPT oracle machines with access to the oracle $O$. Note that in the ideal-process the environment $\mathcal{Z}$ has access to the oracle $O$ as well (otherwise, it is easy to distinguish the real-life model from the ideal-process) and therefore $\mathcal{SIM}$ must as well have access to the oracle $O$. While considering security relative to oracles, an oracle $O$ is added to all notations (random variables, distributions, etc.).

We present the definition of security only for the semi-honest case as this suffices for our separation.

**Definition 2.7** *Let $\Pi$ be a protocol for computing a functionality $f$. We say that $\Pi$* securely computes the functionality $f$ in the presence of adaptive semi-honest adversaries relative to an oracle $O$ *if for every PPT real-life adversary $\mathcal{A}$ and every PPT environment $\mathcal{Z}$, there exists a PPT adaptive ideal-process adversary $\mathcal{SIM}$, where all are given oracle access to $O$, such that:*

$$\text{REAL}^{O}_{\Pi,\mathcal{A},\mathcal{Z}} \stackrel{c^O}{\equiv} \text{IDEAL}^{O}_{f,\mathcal{SIM},\mathcal{Z}}$$

# 3 Adaptive Zero-Knowledge Proofs

In this section we prove the following theorem:

**Theorem 3.1** *Assuming the existence of one-way functions that are hard to invert for non-uniform adversaries, there exist adaptive zero-knowledge proofs for all $\mathcal{NP}$.*

Specifically, we show how to construct an adaptive zero-knowledge proof for the language of Hamiltonicity ($HC$). Our construction is based on Blum's zero-knowledge proof for Hamiltonicity [6]. In this protocol, the prover first commits to a random permutation of the input graph $G$, and the verifier then chooses randomly whether to verify that the committed graph is indeed a permutation of $G$ or that the committed graph contains a Hamiltonian cycle. Soundness holds because a non-Hamiltonian graph cannot simultaneously be a permutation of $G$ and contain a Hamiltonian cycle. The simulator for this proof system does not know the witness and so cannot decommit to a Hamiltonian cycle after committing to a permutation of $G$. Therefore, it works by randomly choosing whether to send commitments to a permutation of $G$ or to a graph containing only a random cycle of length $n$.

Observe that in the latter case, the commitments generated by the simulator are to different values than those generated by the real prover. This is not a problem when considering static corruptions because the hiding property of the commitments means that this cannot be distinguished. However, in the setting of adaptive corruptions, the prover can be corrupted after the simulation ends. In this case, the simulator must be able to provide random coins that demonstrate that the commitments sent initially are those that an honest prover would have sent. However, when the simulator commits to a graph containing only a Hamiltonian cycle, it cannot do this (because an honest prover never sends such a commitment). Thus, the commitment scheme used must be such that the simulator can explain commitments to 0 as commitments to 1 and vice versa (actually it suffices that commitments to 0 be explainable as commitments to 1). One way of solving this problem is to use *equivocal* or *trapdoor* commitments. Loosely speaking, these are commitments that can be decommitted to both 0 and 1 given an appropriate trapdoor. As we have discussed, however, if we use this type of commitment scheme, then we can no longer achieve statistical soundness (since an all-powerful cheating prover can find the trapdoor and use the equivocality of the commitment scheme to fool the verifier).

We bypass this problem by constructing a new type of *instance-dependent* commitment scheme [26, 4]. Roughly speaking, these are commitments whose properties depend on whether the instance in question is in the language or not. Typically, they are defined for a language $L$ as follows. Let $x$ be a statement. If $x \in L$ then the commitment associated with $x$ is computationally hiding and if $x \notin L$ then the commitment associated with $x$ is perfectly binding. In order to achieve adaptive security, we extend this to an *adaptive* instance-dependent commitment scheme, where for $x \in L$ we have the additional property that commitments are equivocal (but for $x \notin L$ the commitments are still perfectly binding). Note that this type of commitment scheme is enough for constructing zero-knowledge proofs because the hiding and the adaptivity are essential only for proving zero-knowledge (which is needed only for $x \in L$) and the binding property is only essential for proving soundness (in the case of $x \notin L$). In Section 3.1 we provide a formal definition of *adaptive instance-dependent commitment schemes* along with a construction and a proof of security. In Section 3.2 we prove Theorem 3.1 by constructing an adaptive zero-knowledge proof system for Hamiltonicity and proving its security.

## 3.1 Adaptive Instance-Dependent Commitment Schemes

### 3.1.1 Definition

**Syntax.** Let $R$ be an $\mathcal{NP}$ relation and $L$ be the language associated with $R$. A (non-interactive) adaptive instance dependent commitment scheme (AIDCS) for $L$ is a tuple of probabilistic polynomial-time algorithms ($\mathsf{Com}$, $\mathsf{Com}'$, $\mathsf{Adapt}$), where:

- $\mathsf{Com}$ is the bit commitment algorithm: For a bit $b \in \{0,1\}$, an instance $x \in \{0,1\}^*$ and a random string $r \in \{0,1\}^{p(|x|)}$ (where $p(\cdot)$ is a polynomial), $\mathsf{Com}(x,b;r)$ returns a commitment value $c$.

- $\mathsf{Com}'$ is a "fake" commitment algorithm: For an instance $x \in \{0,1\}^*$ and a random string $r \in \{0,1\}^{p(|x|)}$, $\mathsf{Com}'(x;r)$ returns a commitment value $c$.

- $\mathsf{Adapt}$ is an adaptive opening algorithm: Let $x \in L$ and $w \in R_x$. For all $c$ and $r \in \{0,1\}^{p(|x|)}$ such that $\mathsf{Com}'(x;r) = c$, and for all $b \in \{0,1\}$, $\mathsf{Adapt}(x,w,c,b,r)$ returns a pair $(b,r')$ such that $c = \mathsf{Com}(x,b;r')$. (In other worlds, $\mathsf{Adapt}$ receives a "fake" commitment $c$ and a bit $b$, and provides an explanation for $c$ as a commitment to the bit $b$.)

A *decommitment* to a commitment $c$ is a pair $(b,r)$ such that $c = \mathsf{Com}(x,b;r)$.

Note the difference between $\mathsf{Com}$ and $\mathsf{Com}'$: $\mathsf{Com}$ is an ordinary committing algorithm (creating a commitment value for a given bit), while for $x \in L$ algorithm $\mathsf{Com}'$ creates commitment values that are not associated to any specific bit. However, given a witness attesting to the fact that $x \in L$, these commitments can later be claimed to be commitments to 0 or to 1 by using algorithm $\mathsf{Adapt}$. We stress that without such a witness, a commitment generated by $\mathsf{Com}'$ cannot necessarily be decommitted to any bit.

**Security.** We now define the notion of security for our commitment scheme. Recall that our goal is eventually designing an adaptive zero-knowledge proof and our definition of security is oriented towards this goal.

Let $C_0^x = \{c \mid \exists\, r \; s.t. \; c = \mathsf{Com}(x,0;r)\}$ and $C_1^x = \{c \mid \exists\, r \; s.t. \; c = \mathsf{Com}(x,1;r)\}$. That is, $C_0^x$ is the set of commitment values that can be decommitted to 0 and $C_1^x$ the set of commitment values that can be decommitted to 1.

**Definition 3.2** *Let $R$ be an $\mathcal{NP}$ relation and $L = L_R$. We say that* $(\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$ *is a secure AIDCS for L if the following hold:*

1. Computational hiding*: The ensembles* $\{\mathsf{Com}(x,0)\}_{x\in L}$*,* $\{\mathsf{Com}(x,1)\}_{x\in L}$ *and* $\{\mathsf{Com}'(x)\}_{x\in L}$ *are computationally indistinguishable.*

2. Adaptivity*: For all $b \in \{0,1\}$, the distributions* $\left\{(\mathsf{Com}(x,b;U_{p(|x|)}), b, U_{p(|x|)})\right\}_{x\in L, w\in R_x}$ *and* $\left\{(\mathsf{Com}'(x;U_{p(|x|)}), \mathsf{Adapt}(x,w,\mathsf{Com}'(x;U_{p(|x|)}), b, U_{p(|x|)}))\right\}_{x\in L, w\in R_x}$ *are computationally indistinguishable (that is, the random coins that are generated by* $\mathsf{Adapt}$ *are indistinguishable from real random coins used by the committing algorithm* $\mathsf{Com}$*).*

3. Perfect binding*: For all $x \notin L$, The sets $C_0^x$ and $C_1^x$ are disjoint.*

### 3.1.2 String commitments

We now argue that an adaptive instance-dependent commitment scheme remains secure even when we concatenate bit commitments. First, it is easy to see that the binding and the hiding properties are preserved when concatenating bit commitments. To prove that adaptivity is preserved as well we consider the following experiment between an adversary and a commitment oracle. In this experiment, the adversary outputs a message $m$ (a sequence of bits) and the commitment oracle chooses either to use the committing algorithm $\mathsf{Com}$ to commit to each of the bits of the message and output the sequence of the commitments together with the random coins used by $\mathsf{Com}$ or to generate fake commitments using algorithm $\mathsf{Com}'$, use algorithm $\mathsf{Adapt}$ to get random coins that are consistent with the message $m$ and output the fake commitments together with the random coins generated by $\mathsf{Adapt}$. The adversary succeeds in the game if it can distinguish between the choices of the commitment oracle with non-negligible probability. Formally, for a commitment scheme $C = (\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$, an adversary $\mathcal{A}$, an instance $x \in L$, a witness $w \in R_x$ and a value $b \in \{0,1\}$, consider the following experiment:

**The string commitment adaptivity experiment** $\mathsf{StrExpAdapt}^b_{\mathcal{A},C}(x,w)$**:**

1. *Upon input $x \in L$ and $w \in R_x$, where $|x| = n$, the adversary $\mathcal{A}$ outputs a message $m \in \{0,1\}^{q(n)}$.*
2. *Let $q = q(n)$. If $b = 0$ the commitment $c = \mathsf{Com}(x, m_1; r_1), ..., \mathsf{Com}(x, m_q, r_q)$ is computed where $r = r_1, ...r_q$ is uniformly chosen and $(c, m, r)$ is given to $\mathcal{A}$.*
   *If $b = 1$, the commitment $c = \mathsf{Com}'(x; r_1), ..., \mathsf{Com}'(x; r_q)$ is computed where $r = r_1, ..., r_q$ is uniformly chosen. Then for all $1 \le i \le q$, $(m_i, r'_i) = \mathsf{Adapt}(x, w, \mathsf{Com}'(x; r_i), m_i, r_i)$ is computed. Finally $(c, m, r')$ is given to $\mathcal{A}$, where $r' = r'_1, ..., r'_q$.*
3. *$\mathcal{A}$ outputs a value $b'$ and this is the output of the experiment.*

The following proposition follows from the adaptivity of the bit commitment scheme and can be proved by a simple hybrid argument:

**Proposition 3.3** *Let $C = (\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$ be an AIDCS. For every PPT machine $\mathcal{A}$, every positive polynomial $p(\cdot)$ and all sufficiently large $x \in L$ and $w \in R_x$:*

$$\left|\Pr\left[\mathsf{StrExpAdapt}^0_{\mathcal{A},C}(x,w) = 1\right] - \Pr\left[\mathsf{StrExpAdapt}^1_{\mathcal{A},C}(x,w) = 1\right]\right| < \frac{1}{p(|x|)}$$

### 3.1.3 The Construction

Our construction is almost identical to the trapdoor commitment of [17] (as adapted by [11]), with one small but crucial difference. We begin by describing the adaptation of [11] of the trapdoor commitment of [17]. Let $\mathsf{C}$ be a perfectly binding commitment scheme with pseudorandom range and let $G$ be a graph (in [17] $G$ is a Hamiltonian graph generated by the receiver whereas in [11] it is a Hamiltonian graph that is placed in the common reference string). Then, in order to commit to 0, the committer chooses a random permutation $\pi$ of the vertices of $G$ and commits to the adjacency matrix of $\pi(G)$ using $\mathsf{C}$. To decommit, it opens all entries and sends $\pi$. To commit to 1, the committer chooses a random $n$-cycle and for all entries in the adjacency matrix corresponding to the edges of the $n$-cycle, it uses $\mathsf{C}$ to commit to 1. In contrast, all other entries are set to a random string (recall that the commitment scheme has a pseudorandom range and thus these values are indistinguishable from commitments using $\mathsf{C}$). To decommit, it opens only the entries corresponding to the edges of the $n$-cycle. As stated, this scheme is computationally hiding due to the underlying commitment scheme $\mathsf{C}$. In addition, it is computational binding as long as the sender does not know the Hamiltonian cycle in $G$. We stress that the scheme is not perfectly binding because an all-powerful corrupted committer can find the Hamiltonian cycle in $G$ and send commitments that it can later open to both 0 and 1.

*Our key observation is that in the setting of zero-knowledge we can use the graph $G$ that is the statement being proven as the graph in the above commitment scheme.* This implies that if $G \in HC$, then the commitment scheme is computationally hiding and if $G \notin HC$ then it is perfectly binding, as required. (As an added bonus, the graph need not be generated by the protocol.) Regarding adaptivity, when $G \in HC$ a commitment to 0 can be opened as a 0 or 1 given a cycle in $G$.

Formally, we define the commitment scheme ($\mathsf{Com}$, $\mathsf{Com}'$, $\mathsf{Adapt}$) as follows:

- $\mathsf{Com}(G, 0)$ chooses a random permutation $\pi$ of the vertices of $G$, and sets $H = \pi(G)$. The output of the algorithm is a series of commitments (using the commitment scheme $\mathsf{C}$) to the adjacency matrix of $H$. Namely, a matrix of size $n \times n$, where the entry $(i, j)$ contains a commitment to 1 if $(i, j) \in E(H)$ and to 0 if $(i, j) \notin E(H)$, where $E(H)$ denotes the set of edges in $H$.

  $\mathsf{Com}(G, 1)$ chooses a random cycle of size $n$. The algorithm outputs a matrix of size $n \times n$, where the entries corresponding to the cycle contain a commitment to 1 (using commitment scheme $\mathsf{C}$). All other entries are set to random strings of the same length as the output length of $\mathsf{C}$.

- $\mathsf{Com}'(G)$ acts exactly as $\mathsf{Com}(G, 0)$; that is, $\mathsf{Com}'(G; r) = \mathsf{Com}(G, 0; r)$.

- $\mathsf{Adapt}(G, w, c, 0, r)$: If $\mathsf{Com}'(G; r) \neq c$, then it returns $\perp$. Otherwise, it outputs the bit 0 and $r$ (recall that $\mathsf{Com}'(G; r) = \mathsf{Com}(G, 0; r)$ and therefore $\mathsf{Com}(G, 0; r) = c$).

  $\mathsf{Adapt}(G, w, c, 1, r)$: If $\mathsf{Com}'(G; r) \neq c$, then it returns $\perp$. Otherwise, it outputs the bit 1, the cycle of length $n$ obtained by applying the permutation $\pi$ (that is a part of $r$) on the Hamiltonian cycle $w$, and $n^2$ strings $\{r_{i,j}\}_{i,j=1}^{n}$ where if the edge $(i, j) \in \pi(w)$, then $r_{i,j}$ is the randomness used by $\mathsf{C}$ to commit to 1 and if $(i, j) \notin \pi(w)$, then $r_{i,j}$ is the commitment value that appears in the corresponding entry in $c$ (recall that $\mathsf{C}$ has pseudorandom range; therefore these values "look" random).

**Proposition 3.4** *Assuming the existence of one way permutations,* $(\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$ *is a secure non-interactive adaptive instance-dependent commitment scheme for the language of Hamiltonicity.*

**Proof:** We show that $(\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$ fulfills Definition 3.2.

*Computational Hiding*: The computational hiding of the scheme has been proven in [11] and therefore we omit the proof.

*Perfect Binding*: Let $G \notin HC$. By the definition, the set $C_0^G$ contains commitments to isomorphic graphs to $G$ and the set $C_1^G$ contains commitments to Hamiltonian cycles. If $G \notin HC$, then a graph cannot be simultaneously isomorphic to $G$ and contain a Hamiltonian cycle and therefore the sets $C_0^G$ and $C_1^G$ are disjoint.

*Adaptivity*: We begin by arguing that for all graphs $G \in HC$ and all $w \in R_G$, the distributions $\{(\mathsf{Com}(G, 0; U_{p(n)}), 0, U_{p(n)}\}$ and $\{(\mathsf{Com}'(G; U_{p(n)}), \mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 0, U_{p(n)}))\}$ are computationally indistinguishable. This is easy to verify, since as we have mentioned in the construction of our AIDCS, $\mathsf{Com}'(G; U_{p(n)})$ is in fact an execution of $\mathsf{Com}(G, 0; U_{p(n)})$ and in this case

$\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 0, U_{p(n)})$ returns a bit 0 and the random coins used by $\mathsf{Com}'(G)$. We conclude that $\{(\mathsf{Com}(G, 0; U_{p(n)}), 0, U_{p(n)}\}$ and $\{(\mathsf{Com}'(G; U_{p(n)}), \mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 0, U_{p(n)}))\}$ are identically distributed.

Now, we argue that for all $G \in HC$ and $w \in R_G$, the distributions $\{(\mathsf{Com}(G, 1; U_{p(n)}), 1, U_{p(n)})\}$ and $\{(\mathsf{Com}'(G; U_{p(n)}), \mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)}))\}$ are computationally indistinguishable. Consider the following experiment: A random $r \in \{0,1\}^{p(n)}$ is chosen and $c = \mathsf{Com}'(G; r)$ is computed. Then, $(1, r') = \mathsf{Adapt}(G, w, c, 1, r)$ is computed and the output of the experiment is $(\mathsf{Com}(G, 1, ; r'), 1, r')$. We denote the distribution induced by this experiment by $H_{\mathsf{Com}}$. It is not difficult to see that the distributions $\{(\mathsf{Com}'(G; U_{p(n)}), \mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)}))\}$ and $H_{\mathsf{Com}}$ are identically distributed. We will show that $\{(\mathsf{Com}(G, 1; U_{p(n)}), 1, U_{p(n)}\}$ and $H_{\mathsf{Com}}$ are computationally indistinguishable. Recall that $\mathsf{Adapt}(G, w, c, 1, r)$ outputs $n^2$ strings $\{r_{i,j}\}_{i,j=1}^n$ where if the edge $(i, j) \in \pi(w)$, then $r_{i,j}$ is the random string used by $\mathsf{C}$ to commit to 1 and if $(i, j) \notin \pi(w)$, then $r_{i,j}$ is the value that appears in the corresponding entry in $c$. That is, if $(i, j) \in \pi(w)$, the corresponding string $r_{i,j}$ is a truly random string and if $(i, j) \notin \pi(w)$, then the corresponding string $r_{i,j}$ is an output of the commitment scheme $\mathsf{C}$. We have taken $\mathsf{C}$ to have a pseudorandom range, and this implies that the distribution $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$ is computationally indistinguishable from $\{U_{p(n)}\}$.

Now, assume that $\{(\mathsf{Com}(G, 1; U_{p(n)}), 1, U_{p(n)}\}$ and $H_{\mathsf{Com}}$ are not computationally indistinguishable. Informally speaking, there exists a PPT machine $D$ that can distinguish between $\{(\mathsf{Com}(G, 1; U_{p(n)}), 1, U_{p(n)})\}$ and $H_{\mathsf{Com}}$ with a non-negligible probability. We construct a distinguisher $D'$ that can distinguish $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$ and $\{U_{p(n)}\}$. $D'$ gets an input an $r$ that is distributed according to $\{U_{p(n)}\}$ or $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$. $D'$ then computes $c = \mathsf{Com}(G, 1, r)$, hands $(c, 1, r)$ to $D$ and outputs the output of $D$. Now, if $r$ is distributed according to $\{U_{p(n)}\}$, then the input of $D$ is distributed according to $\{(\mathsf{Com}(G, 1; U_{p(n)}), 1, U_{p(n)}\}$. On the other hand, if $r$ is distributed according to $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$, then the input of $D$ is distributed as $H_{\mathsf{Com}}$ and therefore $D'$ distinguishes $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$ and $\{U_{p(n)}\}$ with a non-negligible probability, contradicting the computational indistinguishability of $\{\mathsf{Adapt}(G, w, \mathsf{Com}'(x; U_{p(n)}), 1, U_{p(n)})\}$ and $\{U_{p(n)}\}$. ■

**Remark - one-way functions.** Note that we can reduce our computational assumptions to the existence of one-way functions (rather than one-way permutations) by replacing the commitment scheme $\mathsf{C}$ by the scheme of [34] (which has a pseudorandom range as well). The resulting scheme will be an *interactive* Adaptive Instance Dependent Commitment Scheme with computational hiding and statistical binding. Also note that all proofs can be extended to the case of non-uniform adversaries assuming the existence of one-way functions that are hard to invert for non-uniform adversaries.

### 3.2 Adaptive Zero Knowledge Proofs for all NP

In this section, we show that the proof system for Hamiltonicity presented in [6] is an adaptive zero-knowledge proof system when the ordinary commitment scheme is replaced by an adaptive instance-dependent commitment scheme.

**Theorem 3.5** *If there exists an AIDCS for an $\mathcal{NP}$-complete problem with computational hiding and adaptivity against non-uniform adversaries, then every NP language has an adaptive zero knowledge proof.*

**Proof:** We present an adaptive zero knowledge proof for Hamiltonicity; the generalization to any language in $\mathcal{NP}$ is achieved by reducing the language to Hamiltonicity. Let $C = (\mathsf{Com}, \mathsf{Com}', \mathsf{Adapt})$ be an AIDCS for Hamiltonicity.

**Protocol 1 (Adaptive zero-knowledge proof for $HC$)** *:*

- **The verifier's input:** *A graph $G = (V, E)$ where $|V| = n$.*
- **The prover's input:** *A graph $G = (V, E)$ where $|V| = n$ and a hamiltonian cycle $w$ in $G$.*
- **The protocol:**
  1. *The prover chooses a random permutation $\pi$ of the vertices and commits to the adjacency matrix of $\pi(G)$ using algorithm* $\mathsf{Com}$*.*
  2. *The verifier sends a random bit $\sigma$ to the prover.*
  3. *If $\sigma = 0$ the prover sends $\pi$ to the verifier and decommits to all entries of the adjacency matrix.*

     *If $\sigma = 1$, the prover decommits to the entries of the Hamiltonian cycle in $\pi(G)$.*
  4. *If $\sigma = 0$ the verifier checks that the decommitted graph is indeed $\pi(G)$.*

     *If $\sigma = 1$ the verifier checks that the decommitted entries form a hamiltonian cycle of size $n$.*

**Claim 3.6** *Protocol 1 is an efficient-prover interactive proof for the language of Hamiltonicity.*

**Proof:** It is easy to see that the strategy of the verifier can be implemented in probabilistic polynomial-time and that given a Hamiltonian cycle in $G$, the strategy of the prover can be implemented in probabilistic polynomial-time as well.

Proving the completeness property of the protocol is straightforward. We will prove soundness of $\frac{1}{2}$. Let $G \notin HC$, and assume there exists a prover strategy $P^*$ such that $\Pr[\langle P^*, V\rangle(G) = 1] > \frac{1}{2}$. Let $p_i$ be $P^*$ $i$'th message to $V$. Assuming $P^*$ convinces $V$ with probability that is greater than $\frac{1}{2}$, there exists messages $p'_1$ and $p_2^0$ and $p_2^1$ such both $V(p'_1, 0, p_2^0)$ and $V(p'_1, 1, p_2^1)$ accept. Namely, $p_2^0$ is a decommitment of $p'_1$ to an isomorphic graph of $G$ and $p_2^1$ is a decommitment of some entries in the adjacency matrix that is represented by $p'_1$ that form a Hamiltonian cycle. Since the commitment scheme $C$ is perfectly binding when $G \notin HC$, this implies that $G$ contains a Hamiltonian cycle and that is a contradiction to $G \notin HC$. ∎

**Claim 3.7** *Protocol 1 is secure against adaptive adversaries.*

**Proof:** Let $\mathcal{A}$ be a PPT adaptive adversary that interacts with the prover and the verifier in the real-life run of the protocol and let $\mathcal{Z}$ be a PPT environment. We construct an ideal-process adversary(simulator) $\mathcal{SIM}$ such that

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\equiv} \text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}$$

$\mathcal{SIM}$ starts with an input $G$ and a random tape $r_{\mathcal{SIM}}$. $\mathcal{SIM}$ invokes a simulated copy of $\mathcal{A}$ on a uniform random tape. $\mathcal{SIM}$ simulates the interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ as follows: Every input $\mathcal{SIM}$ receives from the environment $\mathcal{Z}$ is written on the input tape of $\mathcal{A}$, as if it came from $\mathcal{A}$'s environment. Every output written by $\mathcal{A}$ on its output tape, is written by $\mathcal{SIM}$ on its own output tape. If $\mathcal{SIM}$ receives some auxiliary input from $\mathcal{Z}$ at the outset, then $\mathcal{SIM}$ hands it to $\mathcal{A}$. To simplify our analysis, we divide our protocol into 3 rounds: The prover's first message, the verifier's first message and the prover's second message, where the adversary might corrupt any of the parties at the onset of each round or in the postexecution corruption phase.

Our description of $\mathcal{SIM}$ consists of 2 parts: simulating corruptions of the parties and simulating the run of the protocol.

- **Simulating the run of the protocol:** We divide the description of the simulation into cases as following.

  1. *The prover is uncorrupted in the first round.* $\mathcal{SIM}$ generates the prover's first message as follows. $\mathcal{SIM}$ chooses a random bit $b \in \{0,1\}$. If $b = 0$, it chooses a random permutation $\pi$ and uses algorithm $\mathsf{Com}$ to commit to the adjacency matrix of $\pi(G)$.

     If $b = 1$, $\mathcal{SIM}$ creates an adjacency matrix of a graph containing only a random cycle of length $n$ (all other entries are set to 0). It then uses $\mathsf{Com}$ to commit to all 1's in the adjacency matrix. All other entries are filled with commitment values created by algorithm $\mathsf{Com}'$.

     We proceed to the second round.

     (a) *The Verifier is corrupted in the second round.* $\mathcal{A}$ generates a bit $\sigma$ and $\mathcal{SIM}$ sets $\sigma$ as the verifier's message. We move to the third round.

        i. *The prover is uncorrupted in the third round.* $\mathcal{SIM}$ generates the prover's second message as follows: If $b = \sigma = 0$ then $\mathcal{SIM}$ sets the prover's second message to be decommitments to all entries at the adjacency matrix and $\pi$. If $b = \sigma = 1$, $\mathcal{SIM}$ decommits to all 1's in the adjacency matrix (and since theses entries were created using $\mathsf{Com}$, $\mathcal{SIM}$ can decommit). In both cases, the run of the protocol terminates. The corrupted verifier's output is a special symbol $\perp$ and the output of the prover is the empty string $\lambda$. $\mathcal{SIM}$ outputs the simulated adversary's internal state. A postexecution corruption step starts and at the end $\mathcal{Z}$ halts with an output.

           If $b \neq \sigma$, $\mathcal{SIM}$ rewinds to the beginning of the first corruption stage (that is, the outset of the first round) and proceeds as above. In this case if the corruptions made by $\mathcal{A}$ do not lead $\mathcal{SIM}$ to the same scenario (that is, the verifier is corrupted before the second round and the prover is uncorrupted in the third round), $\mathcal{SIM}$ keeps rewinding to the outset of the first round until $\mathcal{SIM}$ ends up in the same scenario as the current one (of course, if again $b \neq \sigma$, $\mathcal{SIM}$ has to rewind again).

        ii. *The prover is corrupted in the third round.* The simulated copy of $\mathcal{A}$ generates the prover's second message. The output of the corrupted parties is the special symbol $\perp$ and $\mathcal{SIM}$ outputs the simulated adversary's internal state. Since both parties are corrupted, no postexecution corruption step takes place and $\mathcal{Z}$ halts with an output.

     (b) *The verifier is uncorrupted in the second round.* In this case, $\mathcal{SIM}$ sets $\sigma = b$ to be the verifier's first message (where $b$ is as chosen by $\mathcal{SIM}$ above).

        i. *The prover is uncorrupted in the third round.* Since $\sigma = b$, the simulator simply decommits appropriately to the commitment it sent and the third round terminates. At the end of the protocol, $\mathcal{SIM}$ outputs the simulated adversary's internal state. A postexecution corruption stage starts and at the end $\mathcal{Z}$ halts with an output.

ii. *The prover is corrupted in the third round.* The simulated copy of $\mathcal{A}$ generates the prover's second message and the protocol ends. $\mathcal{SIM}$ outputs the simulated adversary's internal state. If the verifier is uncorrupted, a postexecution corruption stage starts and at the end $\mathcal{Z}$ halts with an output.

2. *The prover is corrupted in the first round.* The simulated copy of $\mathcal{A}$ generates the prover's first message and the first round ends.
   (a) *The Verifier is corrupted in the second round.* The simulated copy of $\mathcal{A}$ generates the verifier's first message and the prover's second message. At the end, both parties output the special symbol $\perp$ and $\mathcal{SIM}$ outputs the simulated adversary's internal state. Since both parties are corrupted, no postexecution corruption stage is taken place and $\mathcal{Z}$ halts with an output.
   (b) *The verifier is uncorrupted in the second round.* $\mathcal{SIM}$ chooses a random bit $\sigma$ and sets it as the verifier's first message. The simulated copy of $\mathcal{A}$ generates the prover's second message and the protocol ends. $\mathcal{SIM}$ outputs the simulated adversary's internal state. If the verifier is uncorrupted, a postexecution corruption stage starts and at the end $\mathcal{Z}$ halts with an output.

- **Simulating Corruptions:**
  1. *$\mathcal{A}$ corrupts the verifier at the outset of the first or second rounds:* $\mathcal{SIM}$ corrupts the verifier and hands a truly random string to $\mathcal{A}$ as the verifier's random tape $r_V$.
  2. *$\mathcal{A}$ corrupts the verifier at the outset of the third round or in the postexecution corruption step:* In this case, the verifier's first message has been set by $\mathcal{SIM}$ to a random bit $\sigma$. $\mathcal{SIM}$ corrupts the verifier and provides $\mathcal{A}$ with a random tape $r_V$ that is consistent with $\sigma$.
  3. *$\mathcal{A}$ corrupts the prover at the outset of the first round:* $\mathcal{SIM}$ corrupts the prover, learns its input tape that includes the witness $w$, sets $r_P$ to be a truly random string and provides $\mathcal{A}$ with $w$ and $r_P$.
  4. *$\mathcal{A}$ corrupts the prover at the outset of the second or third round or at the postexecution corruption step:* $\mathcal{SIM}$ corrupts the prover in the ideal-process run of the protocol and learns the witness $w$ which is a part of its input tape. At this stage, $\mathcal{A}$ expects to learn the input and the random tape of $P$, and $\mathcal{SIM}$ has to generate a random tape for $P$ that is consistent with the prover's first message. If $b = 0$, the prover's first message has been created exactly as in a real run of the protocol and the simulator can provide $\mathcal{A}$ with the random coins used to create the commitment. If $b = 1$, the prover's first message is a commitment to a graph that contains a single cycle of length $n$; we denote this graph by $C_n$. The simulator finds an isomorphism $\pi$ between the Hamiltonian cycle $w$ in $G$ and between $C_n$. It then computes $H = \pi(G)$. For every edge $(u, v) \in C_n$, $\mathcal{SIM}$ provides $\mathcal{A}$ with the randomness used by $\mathsf{Com}$ to commit to 1. For every edge $(u, v) \in H$ and $(u, v) \notin C_n$, $\mathcal{SIM}$ uses algorithm $\mathsf{Adapt}$ to obtain random coins such that the appropriate commitment value in the adjacency matrix is a commitment to 1. For every $(u, v) \notin H$, $\mathcal{SIM}$ uses algorithm $\mathsf{Adapt}$ to get random coins such that the commitment value in the adjacency matrix is a commitment to 0. $\mathcal{SIM}$ provides the simulated copy of $\mathcal{A}$ with the input $w$ and the set of random coins described above as well as with $\pi$.

We begin by showing that the running time of $\mathcal{SIM}$ is expected polynomial time. We use the enumeration of the cases above (in the simulation of the run of the protocol) to indicate which parties have been corrupted and when in a specific run of the simulation. For example, when we refer to case 2, we refer to the case where the prover is corrupted at the outset of the first run. Similarly, when we refer to case 1a, we refer to the case where the prover is uncorrupted at the outset of the first round and the verifier is corrupted at the outset of the second round and when we refer to case 1(a)ii, we refer to the case where the prover is uncorrupted at the outset of the first round, the verifier is corrupted at the outset of the second round and the prover is corrupted at the outset of the third round.

It is easy to see that a single iteration of $\mathcal{SIM}$ (where no rewinding is carried out) is polynomial time (recall that the real-life adversary $\mathcal{A}$ is a PPT machine and therefore invoking a simulated copy of $\mathcal{A}$ is also polynomial time) and that $\mathcal{SIM}$ rewinds only if it reaches to case 1(a)i (when the prover is uncorrupted at the third round and the verifier is corrupted before the second round) and its random bit ($b$) does not equal the verifier's first message. In this case, $\mathcal{SIM}$ rewinds until an iteration where it reaches again to case 1(a)i but its random bit equals the verifier's first message. We denote by $\hat{b}$ the random bit of $\mathcal{SIM}$ in its first iteration and by $\hat{\sigma}$ the verifier's message in the first iteration. We obtain that in any case other than the one where $\mathcal{SIM}$ reaches to case 1(a)i and $\hat{b} \neq \hat{\sigma}$ the running time of $\mathcal{SIM}$ is polynomial and therefore:

$$
\begin{aligned}
E\left[\mathrm{TIME}\right] &= E\left[\mathrm{TIME} \mid \mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma}\right] \cdot \Pr\left[\mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma}\right] \\
&+ E\left[\mathrm{TIME} \mid \neg(\mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma})\right] \cdot \Pr\left[\neg(\mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma})\right] \\
&\leq E\left[\mathrm{TIME} \mid \mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma}\right] \cdot \Pr\left[\mathit{case}\ 1(a)i\ \wedge\ \hat{b} \neq \hat{\sigma}\right] + \mathsf{poly(n)}
\end{aligned}
$$

By the hiding property of our commitment scheme, we obtain the following lemma:

**Lemma 3.8** *In each iteration of case 1(a)i, the probability that $\mathcal{A}$ replies with $\sigma = b$ is greater than $\frac{1}{4}$. Formally, in each iteration:*

$$\Pr[\sigma = b \mid \mathit{case}\ 1(a)i] > \frac{1}{4}$$

The proof of this lemma is identical to the case of Hamiltonicity for static adversaries and regular commitments (intuitively, it is due to the hiding of the commitment scheme) and therefore we omit it.

We obtain that in each iteration:

$$\Pr[\mathit{case}\ 1(a)i\ \wedge\ \sigma = b] = \Pr[\sigma = b \mid \mathit{case}\ 1(a)i] \cdot \Pr[\mathit{case}\ 1(a)i] > \frac{1}{4} \cdot \Pr[\mathit{case}\ 1(a)i]$$

For any iteration of $\mathcal{SIM}$ we denote by $p$ the probability that $\mathcal{SIM}$ reaches case 1(a)i and obtain:

$$\Pr[\mathit{case}\ 1(a)i\ \wedge\ \sigma = b] > \frac{1}{4} \cdot p \tag{1}$$

We turn now to computing $E\left[\mathrm{TIME} \mid \mathit{case}\ 1(a)i\ \wedge\ \hat{\sigma} \neq \hat{b}\right]$. Given that $\mathcal{SIM}$ rewinds, let $T$ be the random variable denoting the number of iterations of $\mathcal{SIM}$ until the simulation ends; that is, the number of iterations until $\mathcal{SIM}$ ends up again in case 1(a)i and $\sigma = b$. Since the running time of each such iteration is polynomial time, we obtain:

$$E\left[\mathrm{TIME} \mid \mathit{case}\ 1(a)i\ \wedge\ \hat{\sigma} \neq \hat{b}\right] \leq \mathsf{poly(n)} \cdot E\left[T\right]$$

Now, note that $T$ is distributed according a geometric distribution with success probability at least $\frac{1}{4}p$ (Eq. 1). Therefore $E[T] \leq \frac{4}{p}$ and

$$E\left[\text{TIME} \mid \textit{case } 1(a)i \ \wedge\ \hat{\sigma} \neq \hat{b}\right] \leq \mathsf{poly(n)} \cdot \frac{4}{p}$$

And finally we obtain:

$$\begin{aligned} E[\text{TIME}] &\leq E\left[\text{TIME} \mid \textit{case } 1(a)i \ \wedge\ \hat{\sigma} \neq \hat{b}\right] \cdot \Pr[\textit{case } 1(a)i \ \wedge\ \hat{\sigma} \neq \hat{b}] + \mathsf{poly(n)} \\ &\leq (\mathsf{poly(n)} \cdot \frac{4}{p}) \cdot \frac{3}{4} \cdot p + \mathsf{poly(n)} \end{aligned}$$

which is still polynomial.

Now we show that the output of $\mathcal{Z}$ in the real-process is computationally indistinguishable from the output of $\mathcal{Z}$ in the ideal-process. Namely, we show that

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\equiv} \text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}$$

We first consider a modified simulator $\widetilde{\mathcal{SIM}}$ that receives as input the Hamiltonian cycle in $G$ (of course, $\widetilde{\mathcal{SIM}}$ is not a valid simulator). Then, $\widetilde{\mathcal{SIM}}$ works in the same way as $\mathcal{SIM}$ except that if the prover is uncorrupted in the first round, it chooses a random permutation $\pi$ on the vertices of $G$ and commits to $\pi(G)$ using algorithm $\mathsf{Com}$ (as the honest prover would) regardless of the value of $b$. If the prover is corrupted afterwards, $\widetilde{\mathcal{SIM}}$ sends the real random coins it used to commit to $\pi(G)$ to the simulated copy of $\mathcal{A}$. It is not difficult to see that the output distribution of $\mathcal{Z}$ in a real run of the protocol with adversary $\mathcal{A}$ interacting with the prover and the verifier is identical to that of $\mathcal{Z}$ in an ideal-process run with $\widetilde{\mathcal{SIM}}$. First, consider any of the cases except 1(a)i. In any of these cases, $\widetilde{\mathcal{SIM}}$ acts exactly as the honest real-process parties. Therefore, when $\widetilde{\mathcal{SIM}}$ invokes a simulated copy of $\mathcal{A}$, it gives a perfect simulation of the real-process interaction between the honest parties and $\mathcal{A}$. We now focus on case 1(a)i. In this case, whenever $\sigma \neq b$, $\widetilde{\mathcal{SIM}}$ rewinds whereas in a real-process run of the protocol, the prover decommits appropriately and the execution proceeds. However, since $\widetilde{\mathcal{SIM}}$ does not proceed with the simulation until it reaches again to case 1(a)i (recall that $\widetilde{\mathcal{SIM}}$ rewinds until it reaches case 1(a)i), we obtain the same distribution on the outputs of $\mathcal{Z}$ also in case 1(a)i. Thus, we obtain:

$$\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}^{ZK}_{L,\widetilde{\mathcal{SIM}},\mathcal{Z}}$$

We proceed to show that the output distribution of $\mathcal{Z}$ in an ideal-process run with $\widetilde{\mathcal{SIM}}$ is computationally indistinguishable from the output distribution of $\mathcal{Z}$ in an ideal-process run with $\mathcal{SIM}$. Intuitively, this is true because the only difference between $\mathcal{SIM}$ and $\widetilde{\mathcal{SIM}}$ is the values they commit to and the adaptivity of the commitment scheme implies that the commitments and their decommitments are computationally indistinguishable.

Assume by contradiction that there exists a (non-uniform) polynomial-time distinguisher $D$ and a polynomial $p$ such that for infinitely many graphs $G \in HC$, where $n = |G|$, there exist $w \in R_G$ and $z \in \{0,1\}^*$, such that:

$$\left|\Pr[D(G,z,\text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}(G,w,z)) = 1] - \Pr[D(G,z,\text{IDEAL}^{ZK}_{L,\widetilde{\mathcal{SIM}},\mathcal{Z}}(G,w,z)) = 1]\right| \geq \frac{1}{p(n)}$$

We use $D$ to construct a non-uniform PPT distinguisher $D'$ for the commitment scheme ($\mathsf{Com}$,$\mathsf{Com'}$,$\mathsf{Adapt}$). Formally, we construct a PPT machine $D'$ such that for infinitely many $G \in HC$ there exists a $w \in R_G$ such that:

$$\left|\Pr\left[\mathsf{StrExpAdapt}^0_{D',C}(G,w)=1\right]-\Pr\left[\mathsf{StrExpAdapt}^1_{D',C}(G,w)=1\right]\right|\geq\frac{1}{p(n)}$$

Distinguisher $D'$ is given a graph $G=(V,E)\in HC$ along with its Hamiltonian cycle $w$ as an input and $z$ as an auxiliary input and works as follows.

$D'$ fixes the random tape of $D$ to a uniformly distributed string. $D'$ then chooses a random bit $b\in\{0,1\}$ and prepares commitments as follows. $D'$ chooses a random permutation $\pi$ of the vertices of $G$. If $b=0$, then $D'$ uses algorithm $\mathsf{Com}$ to commit to the adjacency matrix of $\pi(G)$; we denote this commitment by $c'$ and by $r'$ the random coins used by $\mathsf{Com}$. If $b=1$, then $D'$ computes $\pi(w)$ (recall that $w$ is a Hamiltonian cycle in $G$ and therefore $\pi(w)$ is a Hamiltonian cycle in $\pi(G)$). $D'$ then creates a partial adjacency matrix $M$ of the graph $\pi(G)$ except for entries corresponding to the Hamiltonian cycle $\pi(w)$ in the graph. $D'$ hands $M$ to its commitment oracle and receives back a tuple $(c,M,r)$, that includes either real commitments $c$ to $M$ using algorithm $\mathsf{Com}$ and the random coins $r$ used by $\mathsf{Com}$ or fake commitments created by $\mathsf{Com}'$ and random coins generated by algorithm $\mathsf{Adapt}$. $D'$ uses algorithm $\mathsf{Com}$ to compute commitments $c_1$ to 1 for every $(u,v)\in\pi(w)$, let $r_1$ be the random coins used by algorithm $\mathsf{Com}$. Let $c'$ be the commitment to a complete adjacency matrix obtained from $c$ and $c_1$ and let $r'$ be the random coins obtained from $r$ and $r_1$. $D'$ then starts to simulate an execution of $\mathcal{Z}$ on input $z$ interacting with $\mathcal{SIM}$ with the following exceptions:

- If the prover is uncorrupted in the first round, $D'$ sets $c'$ as the prover's first message.
- If the prover is corrupted after the first round and $b=1$, $D'$ hands $r'$ to the simulated copy of $\mathcal{A}$ as the prover's random coins.

In all other cases, $D'$ works exactly as $\mathcal{Z}$ and $\mathcal{SIM}$. Also note that if $D'$ reaches case 1(a)i, then if $b=\sigma$, $D'$ can decommit because it generated the appropriate commitments itself.

At the end of the simulation of the interaction between $\mathcal{Z}$ and $\mathcal{SIM}$, $D'$ invokes $D$ on $G$, $z$ and the output of $\mathcal{Z}$ and outputs whatever $D$ outputs.

It is not difficult to ascertain that when $D'$ is interacting in experiment $\mathsf{StrExpAdapt}^0_{D',C}(G,w)$, the distribution generated by $D'$ is exactly the same as $\text{IDEAL}^{ZK}_{L,\widetilde{\mathcal{SIM}},\mathcal{Z}}(G,w,z)$. Thus,

$$\Pr[\mathsf{StrExpAdapt}^0_{D'(z),C}(G,w)=1]=\Pr[D(G,z,\text{IDEAL}^{ZK}_{L,\widetilde{\mathcal{SIM}},\mathcal{Z}}(G,w,z))=1]$$

On the other hand, when $D'$ is interacting in experiment $\mathsf{StrExpAdapt}^1_{D',C}(G,w)$, the distribution generated by $D'$ is exactly the same as $\text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}(G,w,z)$. Thus,

$$\Pr[\mathsf{StrExpAdapt}^1_{D'(z),C}(G,w)=1]=\Pr[D(G,z,\text{IDEAL}^{ZK}_{L,\mathcal{SIM},\mathcal{Z}}(G,w,z))=1]$$

Combining the above, we have that for infinitely many $G\in HC$ there exists a $w\in R_G$, such that:

$$\left|\Pr\left[\mathsf{StrExpAdapt}^0_{D'(z),C}(G,w)=1\right]-\Pr\left[\mathsf{StrExpAdapt}^1_{D'(z),C}(G,w)=1\right]\right|\geq\frac{1}{p(n)}$$

in contradiction to the adaptivity property of the commitment scheme. ■

This concludes the proof of Theorem 3.5. ■

# 4 Adaptive Oblivious Transfer

In this section we prove a black-box separation of adaptively secure oblivious transfer from enhanced trapdoor permutations. We prove our black-box separation in the following steps. First, in Section 4.1 we define $\Gamma$ and $\Delta$ oracles, where a $\Gamma$-oracle essentially represents an enhanced trapdoor permutation and a $\Delta$-oracle represents a type of symmetric encryption scheme. In Section 4.2 we show that enhanced trapdoor permutations exist relative to most $\Gamma$-oracles. Then, in Section 4.3 we show that if there exists a protocol for securely computing any functionality in the presence of *adaptive* adversaries relative to $\Gamma$-oracles, then there exists a protocol for securely computing the same functionality in the presence of *static* adversaries relative to $\Delta$-oracles. The next step of the proof is to then show that for measure 1 of random $\Delta$-oracles no statically secure $OT_1^2$ exists (we use $OT_1^2$ as a shorthand for 1-out-of-2 oblivious transfer). This is done by using the original black-box separation of key agreement from one-way functions [25], and the fact that key agreement can be obtained from statically secure oblivious transfer; see Section 4.4. We conclude that for measure 1 of random $\Gamma$-oracles no adaptively secure $OT_1^2$ exists (see Section 4.5).

## 4.1 Oracle Definitions

We begin by defining (asymmetric) $\Gamma$ and (symmetric) $\Delta$ oracles which are used in our proof.

**$\Gamma$-oracles.** Informally speaking, a $\Gamma$ oracle is supposed to model an enhanced trapdoor permutation. Thus, it has an oracle for specifying a function and its trapdoor, and an oracle for computing the function (given the function identifier) and inverting it (given the trapdoor). The functions themselves are over all of $\{0,1\}^n$ and thus it is trivial to sample an element without knowing its inverse (as is required for enhanced trapdoor permutations). Formally, we define a $\Gamma$-oracle to be an oracle containing the following functions:

- $G_\Gamma(\cdot) = (G_\Gamma^1, G_\Gamma^2)$ is a pair of injective functions such that on an input $r \in \{0,1\}^n$, $G_\Gamma(r) = (G_\Gamma^1(r), G_\Gamma^2(r)) = (fid, tid) \in \{0,1\}^{2n} \times \{0,1\}^{2n}$. Note that a party can query only $G_\Gamma$ and cannot query one of its components separately.

- A function $F(\cdot,\cdot)$, such that for every $fid \in Range(G_\Gamma^1)$, $F(fid,\cdot)$ is a permutation over $\{0,1\}^n$ and for every $fid \notin Range(G_\Gamma^1)$ and every $x \in \{0,1\}^n$, $F(fid,x) = \perp$.

- A function $F^{-1}$ satisfying $F^{-1}(tid, F(fid,x)) = x$ for every $x \in \{0,1\}^n$ and every $(fid,tid) \in Range(G_\Gamma)$. If $tid$ is not in the range of $G_\Gamma^2(\cdot)$, then $F^{-1}$ returns $\perp$. Note that since $G_\Gamma^1$ and $G_\Gamma^2$ are injective functions, pairs of the form $(fid,tid)$ and $(fid',tid)$, where $fid \neq fid'$ do not exist and $F^{-1}$ is well defined.

**Uniform distribution over oracles – notation:** We denote by $\mathcal{U}_\Gamma$ the uniform distribution over $\Gamma$-oracles. Namely, an oracle $O_\Gamma = (G_\Gamma, F, F^{-1})$ is distributed according to $\mathcal{U}_\Gamma$ if $G_\Gamma^1$ and $G_\Gamma^2$ are two uniformly distributed injective functions from $\{0,1\}^n$ to $\{0,1\}^{2n}$ and for every $fid \in Range(G_\Gamma^1)$, $F(fid,\cdot)$ is a uniformly distributed permutation over $\{0,1\}^n$. We write "$O_\Gamma$ is a random $\Gamma$-oracle" as shorthand for "$O_\Gamma$ is distributed according to $\mathcal{U}_\Gamma$".

**$\Delta$-oracles.** Informally, a $\Delta$ oracle is a symmetric oracle, meaning that anyone with the ability to compute the function also has the ability to invert it. Specifically, we define a function $P$ and its inverse that is analogous to $F$ and $F^{-1}$ in a $\Gamma$ oracle. For reasons that will become apparent later, we also define a function $Q$ and its inverse (this has no analogue in a $\Gamma$ oracle). Formally, we define a "$\Delta$-oracle" to be an oracle containing the following functions:

- $G_\Delta$ is an injective function from $\{0,1\}^n$ to $\{0,1\}^{2n}$.
- A function $P(\cdot,\cdot)$ such that for every $fid \in \mathit{Range}(G_\Delta)$, $P(fid,\cdot)$ is a permutation over $\{0,1\}^n$. For $fid \notin \mathit{Range}(G_\Delta)$ and every $x \in \{0,1\}^n$, $P(fid,x) = \perp$.
- $P^{-1}$ is the inversion algorithm of $P$. Namely for every $fid \in \mathit{Range}(G_\Delta)$ and $x \in \{0,1\}^n$, $P^{-1}(fid, P(fid,x)) = x$. For $fid \notin \mathit{Range}(G_\Delta)$ and every $x \in \{0,1\}^n$, $P^{-1}(fid,x) = \perp$.
- $Q$ is an injective function from the range of $G_\Delta$ to $\{0,1\}^{2n}$. Namely, for every $fid \in \mathit{Range}(G_\Delta)$, $Q(fid) \in \{0,1\}^{2n}$, for every $fid \neq fid' \in \mathit{Range}(G_\Delta)$, $Q(fid) \neq Q(fid')$ and for every $fid \notin \mathit{Range}(G_\Delta)$, $Q(fid) = \perp$.
- $Q^{-1}$ is the inversion algorithm of $Q$. Namely, for every $fid \in \mathit{Range}(G_\Delta)$, $Q^{-1}(Q(fid)) = fid$, and for every $y \notin \mathit{Range}(Q)$, $Q^{-1}(y) = \perp$.

We denote by $\mathcal{U}_\Delta$ the uniform distribution over $\Delta$-oracles: The oracle $O_\Delta = (G_\Delta, P, P^{-1}, Q, Q^{-1})$ is distributed according to $\mathcal{U}_\Delta$, if $G_\Delta$ is a uniformly distributed injective function from $\{0,1\}^n$ to $\{0,1\}^{2n}$, for every $fid \in \mathit{Range}(G_\Delta)$, $P(fid,\cdot)$ is a uniformly distributed permutation over $\{0,1\}^n$ and $Q$ is a uniformly distributed injective function from the range of $G_\Delta$ to $\{0,1\}^{2n}$. We sometimes write "$O_\Delta$ is a random $\Delta$-oracle" instead of "$O_\Delta$ is distributed according to $\mathcal{U}_\Delta$".

Note the difference between $\Gamma$-oracles and $\Delta$-oracles. $\Gamma$-oracle have an asymmetric nature: $F$ and its inversion oracle $F^{-1}$ use different keys. On the contrary, $\Delta$-oracles have a symmetric nature: identical keys are used by $P$ and its inversion oracle $P^{-1}$. (For this reason, we used a "symmetric" character $\Delta$ for $\Delta$-oracles and an "asymmetric" character $\Gamma$ for $\Gamma$-oracles.)

**$\Gamma$-oracles versus $\Delta$-oracles.** We now show a bijection mapping $\phi$ that maps every $\Gamma$-oracle to a corresponding $\Delta$-oracle. Let $O_\Gamma = (G_\Gamma, F, F^{-1})$ be a $\Gamma$-oracle. $\phi(O_\Gamma)$ is the tuple of functions $(G_\Delta, P, P^{-1}, Q, Q^{-1})$ satisfying:

- For every $r \in \{0,1\}^n$, we define $G_\Delta(r) = G^1_\Gamma(r)$.
- For every $r \in \{0,1\}^n$, $Q(G_\Delta(r)) = G^2_\Gamma(r)$, and for every $fid \notin \mathit{Range}(G_\Delta)$, $Q(fid) = \perp$.
- For every $fid \in \{0,1\}^{2n}$ and $x \in \{0,1\}^n$, we define $P(fid,x) = F(fid,x)$.
- $P^{-1}$ and $Q^{-1}$ are the inversion algorithms of $P$ and $Q$.

**Claim 4.1** *The mapping $\phi$ is a bijection from the set of $\Gamma$-oracles to the set of $\Delta$-oracles.*

**Proof:** We first show that for every $O_\Gamma$, $\phi(O_\Gamma) = (G_\Delta, P, P^{-1}, Q, Q^{-1})$ is indeed a $\Delta$-oracle:

- $G_\Delta$ is an injective function because $G^1_\Gamma$ is an injective function.
- For $fid \in \mathit{Range}(G_\Delta)$ it holds that $fid \in \mathit{Range}(G^1_\Gamma)$ and since $F(fid,\cdot)$ is a permutation over $\{0,1\}^n$, $P(fid,\cdot)$ is a permutation over $\{0,1\}^n$ as well.
- For $fid \notin \mathit{Range}(G_\Delta)$ it holds that $fid \notin \mathit{Range}(G^1_\Gamma)$. For every $x \in \{0,1\}^n$, it holds that $F(fid,x) = \perp$ and thus $P(fid,x) = \perp$ as desired.

- $Q$ is injective because $G^2_\Gamma$ is injective and therefore for every $fid_1 \neq fid_2 \in Range(G_\Delta)$, it holds that $Q(fid_1) \neq Q(fid_2)$.

- For every $fid \notin Range(G_\Delta)$, $Q(fid) = \perp$ as desired.

We show that the mapping is injective. Let $(G_\Gamma, F, F^{-1})$ and $(\widehat{G}_\Gamma, \widehat{F}, \widehat{F}^{-1})$ be two different $\Gamma - oracles$. That is, there exists an $r$ such $G_\Gamma(r) \neq \widehat{G}_\Gamma(r)$ or a pair $fid, x$ such that $F(fid, x) \neq \widehat{F}(fid, x)$. Let $(G_\Delta, P, P^{-1}, Q, Q^{-1}) = \phi(G, F, F^{-1})$ and $(\widehat{G}_\Delta, \widehat{P}, \widehat{P}^{-1}, \widehat{Q}, \widehat{Q}^{-1}) = \phi(\widehat{G}, \widehat{F}, \widehat{F}^{-1})$. If there exists an $r$ such $G_\Gamma(r) \neq \widehat{G}_\Gamma(r)$, then there are two cases:

- If $G^1_\Gamma(r) \neq \widehat{G}^1_\Gamma(r)$, then $G_\Delta(r) \neq \widehat{G}_\Delta(r)$.

- If $G^1_\Gamma(r) = \widehat{G}^1_\Gamma(r)$, it implies that $G^2_\Gamma(r) \neq \widehat{G}^2_\Gamma(r)$, and then $Q(G_\Delta(r)) \neq \widehat{Q}(G_\Delta(r))$.

Thus, in both cases, $\phi(G, F, F^{-1}) \neq \phi(\widehat{G}, \widehat{F}, \widehat{F}^{-1})$. Next, if there exist a pair $fid, x$ such that $F(fid, x) \neq \widehat{F}(fid, x)$, then $P(fid, x) \neq \widehat{P}(fid, x)$. Thus, as before $\phi(G, F, F^{-1}) \neq \phi(\widehat{G}, \widehat{F}, \widehat{F}^{-1})$ and the mapping is injective.

It remains to show that the mapping is onto. Let $O_\Delta = (G_\Delta, P, P^{-1}, Q, Q^{-1})$ be a $\Delta$-oracle. We show there exists a $\Gamma$-oracle $O_\Gamma$ that fulfills $\phi(O_\Gamma) = O_\Delta$. Let $O_\Gamma$ be the following oracle:

- For every $r \in \{0,1\}^n$, it holds that $G_\Gamma(r) = (G_\Delta(r), Q(G_\Delta(r))$.

- For every $fid \in \{0,1\}^{2n}$ and $x \in \{0,1\}^n$, it holds that $F(fid, x) = P(fid, x)$.

- For every $tid \in \{0,1\}^{2n}$ and $y \in \{0,1\}^n$, it holds that $F^{-1}(tid, y) = P^{-1}(Q^{-1}(tid), y)$

First, observe that $O_\Gamma$ is indeed a $\Gamma$-oracle:

- Since $G_\Delta$ and $Q$ are injective, it holds that $G^1_\Gamma$ and $G^2_\Gamma$ are injective.

- For every $fid \in Range(G^1_\Gamma)$, since $fid \in Range(G_\Delta)$ and $P(fid, \cdot)$ is a permutation over $\{0,1\}^n$, $F(fid, \cdot)$ is a permutation over $\{0,1\}^n$.

- For every $fid \notin Range(G^1_\Gamma)$, it holds that $fid \notin Range(G_\Delta)$ and for every $x \in \{0,1\}^n$, $P(fid, x) = \perp$ and thus $F(fid, x) = \perp$ as desired.

- For every $(fid, tid) \in Range(G_\Gamma)$ and for every $x \in \{0,1\}^n$,

$$F^{-1}(tid, F(fid, x)) = P^{-1}(Q^{-1}(tid), P(fid, x)) = P^{-1}(fid, P(fid, x)) = x$$

  as desired.

- For every $tid \notin Range(G^2_\Gamma)$ and every $y \in \{0,1\}^n$, it holds that $Q^{-1}(tid) = \perp$ and thus $F^{-1}(tid, y) = \perp$.

Verifying that $\phi(O_\Gamma) = O_\Delta$ is easy. ■

The above claim immediately implies the following:

**Corollary 4.2** *The random variables $\mathcal{U}_\Delta$ and $\phi(\mathcal{U}_\Gamma)$ are identically distributed.*

## 4.2 Enhanced TDP Relative to $\Gamma$-Oracles

In this section we prove that enhanced trapdoor permutations exist relative to $\Gamma$-oracles. With every $\Gamma$-oracle $O_\Gamma = (G_\Gamma, F, F^{-1})$, we associate the following tuple of algorithms $(I, S', F, B)$[6]:

- On input $1^n$, $I$ selects a random $r \in \{0,1\}^n$ and sets $(\alpha, \tau) = G_\Gamma(r)$.
- On input $\alpha$ and a random $r \in \{0,1\}^n$, $S'$ returns $r$ (which is a uniformly distributed element in the domain of $f_\alpha$).
- For a given $\alpha$ and $x$, algorithm $F$ returns $f_\alpha(x) = F(\alpha, x)$.
- For a given $\tau$ and $y \in \{0,1\}^n$, algorithm $B$ returns $F^{-1}(\tau, y)$. Note that if there exists an $\alpha$ such that $(\alpha, \tau)$ is in the range of $I(1^n)$, then $B(\tau, y) = f_\alpha^{-1}(y)$.

We prove that for almost all $\Gamma$-oracles, the tuple of algorithms defined above is an enhanced trapdoor permutation. We first show that $f_\alpha$ is hard to invert.

**Claim 4.3 (Adapted from [25])** *Let $M$ be an oracle PPT machine. Then, there exists a negligible function $neg(\cdot)$ such that for any $n$ and any input $y$ of length $n$, and for every $\alpha$ in the range of $I_1(1^n)$:*

$$\Pr\left[f_\alpha(M^{\mathcal{U}_\Gamma}(\alpha, y)) = y\right] < neg(n)$$

*where the probability is taken over random $\Gamma$-oracles and over the random tape of $M$.*

**Proof:** Machine $M$ gets as input $\alpha$ and $y$ and tries to invert $f_\alpha$ on $y$. Fix any random tape for $M$. First, we note that if $M$ never queries $G_\Gamma$ on an $r$ such that $G_\Gamma^1(r) = \alpha$, then it can guess the corresponding $\tau$ with probability at most $\frac{1}{2^{2n}}$ (since $\tau$ is distributed uniformly on $\{0,1\}^{2n}$). Now, note that every time $M$ makes a query to $G_\Gamma(r)$, the probability that $G_\Gamma^1(r) = \alpha$ is equal to $\frac{1}{2^n}$. $M$ makes only a polynomial number of queries to $G_\Gamma$ and therefore $M$ can find the corresponding $\tau$ for $\alpha$ with only a negligible probability.

Now, we argue that if $M$ does not possess the corresponding $\tau$ for $\alpha$, then it can invert $f_\alpha$ on $y$ with only a negligible probability. First, note that if $M$ never queries $F(\alpha, \cdot)$ on an $x$ such that $F(\alpha, x) = y$, then the probability that it outputs such an $x$ is bounded by $\frac{1}{2^n}$ (since $x$ is distributed uniformly on $\{0,1\}^n$). Every time $M$ makes a query $F(\alpha, x)$ the probability that $F(\alpha, x) = y$ is equal to $\frac{1}{2^n}$. $M$ makes only a polynomial number of queries to its oracle and therefore the probability that $f_\alpha(M(\alpha, y)) = y$ is a negligible function of $n$. ■

The following lemma is proven in [25, section 4.2] using Markov's inequality and the Borel-Cantelli lemma.

**Lemma 4.4 (From [25])** *Let $R$ be a set of oracles. If for every PPT oracle machine $M$ there exists a negligible function $neg(\cdot)$ such that for every $n$ and every input corresponding to $n$, $M$ succeeds in a given task with probability less than $neg(n)$ (when the probability is taken over random oracles in $R$ and the random tape of $M$), then for measure 1 of oracles in $R$, for every PPT machine $M$ there exists a negligible function $neg(\cdot)$ such that for all sufficiently large $n$'s, $M$ succeeds in the given task with probability no more than $neg(n)$.*

[6] We use the definition of enhanced TDP that appears in [22], where $I$ is a function that generates a random permutation index with a corresponding trapdoor, $S$ is a function that given a permutation index samples a random element in its domain, $F$ is a function that given a permutation index computes it on a given element in its domain, and $B$ is the function that given a permutation trapdoor, returns the preimage of a given element in its domain.

Combining Claim 4.3 and Lemma 4.4, we obtain the following theorem:

**Theorem 4.5** *With probability 1 over random $\Gamma$-oracles, the algorithms $(I, S', F, B)$ associated with the oracle form an enhanced trapdoor permutation. In particular, for every PPT machine $M$, every positive polynomial $p(\cdot)$ and for all sufficiently large $n$'s:*

$$\Pr[f_{I_1(1^n)}(M(I_1(1^n), \mathcal{U}_n)) = \mathcal{U}_n] < \frac{1}{p(n)}$$

We remark also that statical semi-honest oblivious transfer can be constructed from any enhanced trapdoor permutation [16] and thus exists relative to $\Gamma$-oracles.

### 4.3 Static $OT_1^2$ Relative to $\Delta$-Oracles from Adaptive $OT_1^2$

In this section we prove that if there exists an adaptively secure $OT_1^2$ relative to random $\Gamma$-oracles, then there exists a statically secure $OT_1^2$ relative to random $\Delta$-oracles. We actually prove a more general theorem that if there exists a protocol for securely computing any functionality $f$ in the presence of adaptive adversaries relative to a random $\Gamma$-oracle, then there exists a protocol for securely computing $f$ in the presence of static adversaries relative to a random $\Delta$-oracle. We restrict our proof to two-party protocols only, but stress that the claim can be proved similarly for multiparty protocols as well.

Let $\Pi_1 = \langle Alice_1, Bob_1 \rangle$ be a protocol for securely computing a functionality $f$ in the presence of *adaptive* adversaries relative to a $\Gamma$-oracle. We use $\Pi_1$ to construct a new protocol $\Pi_2 = \langle Alice_2, Bob_2 \rangle$ for securely computing $f$ in the presence of *static* adversaries relative to a $\Delta$-oracle.

Recall that the parties $Alice_2$ and $Bob_2$ have access to a $\Delta$-oracle, while in the original protocol, $Alice_1$ and $Bob_1$ have access to a $\Gamma$-oracle. There is a fundamental difference between these two cases because a $\Gamma$-oracle is inherently *asymmetric* (it is possible to send a party $fid$ while keeping $tid$ secret, thereby enabling them to compute the permutation but not invert it), while a $\Delta$-oracle is inherently *symmetric* (the same $fid$ is used to compute and invert the permutation). The idea behind our proof is to eliminate the asymmetric nature of the $\Gamma$-oracle by using the fact that in the adaptive setting (e.g., in the post-execution corruption phase), the distinguisher can ultimately corrupt all parties. If it does so, it obtains the entire view of all parties and in particular the view of any party who samples a permutation using $G_\Gamma$. The critical observation is that the probability of a party finding an $fid$ in the range of $G_\Gamma$ without explicitly querying it is negligible. However, if it does make such a query, then its view contains *both $fid$ and $tid$* and this will be obtained by the distinguisher upon corrupting the parties. Thus, the distinguisher is able to compute and invert the permutation, just like in the case of a $\Delta$-oracle. The fact that the adaptive simulator must simulate well even when the distinguisher works in this way (learning all $fid, tid$ pairs) is the basis for constructing a simulator for the static case when using a $\Delta$-oracle.

We begin by defining $\Pi_2 = \langle Alice_2, Bob_2 \rangle$ which is constructed from $\Pi_1$ by replacing the $\Gamma$-oracle with a $\Delta$-oracle:

**Protocol $\Pi_2$:** *On input $x_A$, $Alice_2$ invokes $Alice_1$ on $x_A$. On input $x_B$, $Bob_2$ invokes $Bob_1$ on $x_B$. The execution is described below for a party $\mathcal{P}_2$ emulating $\mathcal{P}_1$, and is the same for both $Alice_2$ and $Bob_2$. In each round:*

- *When $\mathcal{P}_2$ gets the message sent by the other party in the previous round, it hands it to $\mathcal{P}_1$.*

- *If $\mathcal{P}_1$ makes a query $r$ to the oracle $G_\Gamma$, $\mathcal{P}_2$ first queries $G_\Delta(r)$ and gets an output $fid$. Then, $\mathcal{P}_2$ queries $Q(fid)$ and gets an output $tid$. $\mathcal{P}_2$ hands the pair $(fid, tid)$ to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $(fid, x)$ to $F$, $\mathcal{P}_2$ queries $P(fid, x)$, receives an output $y$ and hands $y$ to $\mathcal{P}_1$ (note that $y$ may equal $\perp$).*
- *If $\mathcal{P}_1$ makes a query $(tid, y)$ to $F^{-1}$, $\mathcal{P}_2$ first queries its oracle $Q^{-1}$ on $tid$ and receives an output $fid$. If the outputs is $\perp$, it hands $\perp$ to $\mathcal{P}_1$. Otherwise, $\mathcal{P}_2$ queries $P^{-1}(fid, y)$, obtains an output $x$ and hands $x$ to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ writes a string $m$ on its outgoing communication tape, $\mathcal{P}_2$ sends $m$ to the other party.*
- *At the end of the simulation, $\mathcal{P}_2$ outputs the output of $\mathcal{P}_1$.*

We now prove that $\Pi_2$ securely computes the functionality $f$ in the presence of semi-honest static adversaries.

**Theorem 4.6** *If $\Pi_1$ securely computes a functionality $f$ in the presence of adaptive adversaries relative to a random $\Gamma$-oracle $O_\Gamma$, then $\Pi_2$ securely computes $f$ in the presence of static semi-honest adversaries relative to the $\Delta$-oracle $\phi(O_\Gamma)$.*

**Proof:** The intuition has already been described above and we therefore proceed directly to the proof. Let $O_\Gamma$ be an oracle that is distributed according to $\mathcal{U}_\Gamma$. We show that if $\Pi_1$ is a secure adaptive protocol for computing $f$ relative to $O_\Gamma$, then $\Pi_2$ is a secure static semi-honest protocol for computing $f$ relative to $O_\Delta = \phi(O_\Gamma)$. It is easy to see that $\Pi_2$ computes $f$ relative to $O_\Delta$ because an execution of $\Pi_2$ is, in fact, an execution of $\Pi_1$ with a simulated $\Gamma$-oracle which is exactly $\phi^{-1}(O_\Delta) = O_\Gamma$.

Next we turn to the security of $\Pi_2$. We show that $\Pi_2 = \langle Alice_2, Bob_2 \rangle$ securely computes the functionality $f$ in the presence of static semi-honest adversaries relative to a $O_\Delta = \phi(O_\Gamma)$. We use the ideal-process simulator $\mathcal{SIM}$ of $\Pi_1$ for the adaptive setting to construct two probabilistic polynomial-time simulators $\mathcal{S}_{Alice_2}$ and $\mathcal{S}_{Bob_2}$ for $\Pi_2$ in the static setting, such that

$$\left\{\mathcal{S}^{O_\Delta}_{Alice_2}(1^n, x_A, y_A)\right\}_{x_A, x_B \in \{0,1\}^*} \stackrel{\text{cO}}{\equiv} \left\{VIEW^{\Pi_2, O_\Delta}_{Alice_2}(1^n, x_A, x_B)\right\}_{x_A, x_B \in \{0,1\}^*}$$

$$\left\{\mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, y_B)\right\}_{x_A, x_B \in \{0,1\}^*} \stackrel{\text{cO}}{\equiv} \left\{VIEW^{\Pi_2, O_\Delta}_{Bob_2}(1^n, x_A, x_B)\right\}_{x_A, x_B \in \{0,1\}^*}$$

We use the ideal-process simulator $\mathcal{SIM}$ of $\Pi_1$ for the adaptive setting (see Section 2.2) to construct two probabilistic polynomial-time simulators $\mathcal{S}_{Alice_2}$ and $\mathcal{S}_{Bob_2}$. Since the constructions of $\mathcal{S}_{Alice_2}$ and $\mathcal{S}_{Bob_2}$ and the proofs of indistinguishability are very similar, we present only a construction and a proof for $\mathcal{S}_{Bob_2}$.

Let $\mathcal{A}$ and $\mathcal{Z}$ be the following adversary strategy and environment: $\mathcal{Z}$ starts with an input $z \in \{0, 1\}$. At the onset of the run of $\Pi_1$, $\mathcal{A}$ corrupts $Bob_1$ and at the end of the computation outputs the entire view of $Bob_1$. In the postexecution phase, if $z = 0$, no corruptions are made and if $z = 1$, $\mathcal{Z}$ creates a "corrupt $Alice_1$" message, hands it to $\mathcal{A}$ who corrupts Alice. Eventually $\mathcal{Z}$ outputs the entire view of the corrupted parties (that is: if $z = 0$, the view of Bob alone and if $z = 1$, the view of both parties). No auxiliary information is sent by $\mathcal{Z}$ to $\mathcal{A}$. Let $\mathcal{SIM}$ be the ideal-process adversary guaranteed to exist for $\mathcal{A}$ and $\mathcal{Z}$ by the adaptive security of $\Pi_1$. We now use $\mathcal{A}$, $\mathcal{Z}$ and $\mathcal{SIM}$ to define $\mathcal{S}_{Bob_2}$ (the static simulator for the case that Bob is corrupted). $\mathcal{S}_{Bob_2}$ receives the input $x_B$ and output $y_B$ of Bob as defined by the functionality $f$ and emulates the run of $\mathcal{SIM}$ in the adaptive ideal model with environment $\mathcal{Z}$ with input $z = 0$. Note that $\mathcal{SIM}$ must corrupt only Bob, because in the real world only Bob is corrupted when $z = 0$. We also can assume, w.l.o.g. that $\mathcal{SIM}$ corrupts Bob in the first corruption phase.

$\mathcal{S}_{Bob_2}$ receives input $(x_B, y_B)$ and works as follows, simulating a $\Gamma$-oracle for $\mathcal{SIM}$ using its $\Delta$-oracle:

- If $\mathcal{SIM}$ makes a query $r$ to the oracle $G_\Gamma$, $\mathcal{S}_{Bob_2}$ queries its oracle $G_\Delta(r)$ and receives an output $fid$. It then queries $Q$ on $fid$, gets an output $tid$ and hands the pair $(fid, tid)$ to $\mathcal{SIM}$.
- If $\mathcal{SIM}$ makes a query $(fid, x)$ to $F$, $\mathcal{S}_{Bob_2}$ queries it oracle $P(fid, x)$, gets an output $y$ and hands it to $\mathcal{SIM}$.
- If $\mathcal{SIM}$ makes a query $(tid, y)$ to $F^{-1}$, $\mathcal{S}_{Bob_2}$ first queries its oracle $Q^{-1}$ on $tid$, gets an output $fid$. If the outputs is $\perp$, it hands $\perp$ to $\mathcal{SIM}$. Otherwise, $\mathcal{S}_{Bob_2}$ queries $P^{-1}(fid, y)$, gets an output $x$ and hands $x$ to $\mathcal{SIM}$.
- When $\mathcal{SIM}$ decides to corrupt $Bob_1$, $\mathcal{S}_{Bob_2}$ plays the role of $\mathcal{Z}$ by sending $x_B$ to $\mathcal{SIM}$.
- In the computation phase, $\mathcal{S}_{Bob_2}$ plays the role of the trusted party and sends $y_B$ to $\mathcal{SIM}$ (recall that $\mathcal{S}_{Bob_2}$ gets $y_B$ as input).
- At the end of the simulation, $\mathcal{S}_{Bob_2}$ outputs the output of $\mathcal{SIM}$.

Informally speaking, we show that a distinguisher $D_2$ for $\Pi_2$ and $\mathcal{S}_{Bob_2}$ (relative to $O_\Delta$) implies the existence of a distinguisher $D_1$ for $\Pi_1$ and $\mathcal{SIM}$ (relative to $O_\Gamma$). The idea is to have $D_1$ simulate the run of $D_2$ on the view of Bob. However, $D_2$ has oracle access to a $\Delta$-oracle $O_\Delta$, while $D_1$ has oracle access to a $\Gamma$-oracle $O_\Gamma$. This might be problematic for example if $D_2$ wishes to compute $P^{-1}(fid, y)$ but $D_1$ doesn't know the corresponding $tid$ (recall that $D_1$ can only invert $y$ in the $\Gamma$-oracle world if it holds the trapdoor $tid$ whereas $D_2$ can invert $y$ given $fid$ only). Despite the above, we use the fact that the range of $G_\Gamma$ is a negligible fraction of $\{0,1\}^{2n} \times \{0,1\}^{2n}$, and therefore any $fid$ used in the protocol (except with negligible probability) must have been generated via a query to $G_\Gamma$, as described in the intuition above. More specifically, we show that if there exists a distinguisher $D_2$ that distinguishes the output of $\mathcal{S}_{Bob_2}$ from the output of a corrupted $Bob_2$ in a real execution of $\Pi_2$, then there exists a distinguisher $D_1$ that distinguishes the result of an ideal execution with $\mathcal{SIM}$ from a real execution of $\Pi_1$ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ *with input* $z = 1$, meaning that Alice is also corrupted at the end. (Note that we set $z = 0$ in order to define $\mathcal{S}_{Bob_2}$, but now set $z = 1$ to construct the distinguisher. Since $\mathcal{SIM}$ has to work for all inputs $z$ to $\mathcal{Z}$, this suffices.) Since both Alice and Bob are corrupted in this execution, $D_1$ obtains all of the $(fid, tid)$ pairs generated by queries to $G_\Gamma$ and so it can invert always, enabling it to run $D_2$ and use its $\Gamma$-oracle to answer all of $D_2$'s $\Delta$ queries.

Formally, assume that there exist a PPT machine $D_2$ and a positive polynomial $p(\cdot)$ such that for infinitely many $n$'s, there exist $x_A, x_B \in \{0,1\}^*$ such that:

$$\begin{aligned} &|\Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, VIEW_{Bob_2}^{\Pi_2, O_\Delta}(1^n, x_A, x_B)\right) = 1\right] \\ &- \Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, \mathcal{S}_{Bob_2}^{O_\Delta}(1^n, x_B, y_B)\right) = 1\right] | \geq \frac{1}{p(n)} \end{aligned} \tag{2}$$

We use $D_2$ to construct a PPT machine $D_1$ and a positive polynomial $q(\cdot)$ such that for infinitely many $n$'s, there exist $x_A, x_B \in \{0,1\}^*, z \in \{0,1\}$ such that:

$$\Big|\Pr\Big[D_1^{O_\Gamma}\Big(1^n, x_A, x_B, z, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, z)\Big) = 1\Big] \\ - \Pr\Big[D_1^{O_\Gamma}\Big(1^n, x_A, x_B, z, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, z)\Big) = 1\Big]\Big| \geq \frac{1}{q(n)}$$

The distinguisher $D_1$ receives as input $x_A, x_B, z$ and the output of $\mathcal{Z}$. If $z = 0$, $D_1$ return 0 and halts. Otherwise, the output of $\mathcal{Z}$ consists of both parties' views, including *Alice*'s input $x_A$ and random tape $r_A$ and *Bob*'s input $x_B$ and random tape $r_B$. $D_1$ begins by initializing a table $T_Q$ that will hold all pairs $(fid, tid)$ generated by queries to the oracle. $D_1$ invokes $\langle Alice_1^{O_\Gamma}, Bob_1^{O_\Gamma}\rangle$ on the appropriate input and random tapes (recall that they are a part of $D_1$'s input because $\mathcal{A}$ outputs the views of both parties when $z = 1$) and for every access of one of the parties to $G_\Gamma$, namely a query $G_\Gamma(r) = (fid, tid)$, $D_1$ records the entry $(fid, tid)$ in $T_Q$. When it finishes the execution of $\langle Alice_1^{O_\Gamma}, Bob_1^{O_\Gamma}\rangle$, $D_1$ starts simulating $D_2$ on the view of Bob and proceeds as follows:

- If $D_2$ makes a query $G_\Delta(r)$, $D_1$ makes a query $G_\Gamma(r)$, gets a pair $(fid, tid)$, records the entry $(fid, tid)$ in $T_Q$ and hands $fid$ to $D_2$.
- If $D_2$ tries to compute $Q(fid)$, $D_1$ looks for an entry $(fid, tid)$ in $T_Q$. If such an entry exists, it hands $tid$ to $D_2$ and continues. Else, it hands $\perp$ to $D_2$.
- If $D_2$ tries to compute $Q^{-1}(tid)$, $D_1$ looks for $(fid, tid)$ in $T_Q$. If such an entry exists, it hands $fid$ to $D_2$ and continues. Otherwise, it hands $\perp$ to $D_2$.
- If $D_2$ tries to compute $P(fid, x)$, $D_1$ queries its oracle $F(fid, x)$ and returns its answer.
- If $D_2$ tries to compute $P^{-1}(fid, y)$, $D_1$ checks whether an entry $(fid, tid)$ exists in $T_Q$. If not, it returns $\perp$. If yes, it queries $F^{-1}(tid, y)$ and returns its answer.

Note that if $z = 1$, a run of $D_1$ on input $x_A, x_B, z$ and the views of both parties with access to an oracle $O_\Gamma$ is a simulation of the run of $D_2$ on input $x_B$ and the view of *Bob* with a simulated $\Delta$-oracle obtained from $O_\Gamma$. It's easy to see that only differences between the simulated $\Delta$-oracle obtained from $O_\Gamma$ by $D_1$ and $\phi(O_\Gamma)$ are:

- Queries to $Q$ on an $fid$ in the range of $G_\Delta$ that was not created via a query to $G_\Delta$: For such queries, the simulated $\Delta$-oracle replies with $\perp$ since a pair $(fid, tid)$ doesn't exist in $T_Q$, while $\phi(O_\Gamma)$'s answer is different than $\perp$.
- Queries to $Q^{-1}$ on a $tid$ in the range of $Q$, where the preimage of $tid$ was not created via a query to $G_\Delta$: For such queries, the simulated $\Delta$-oracle replies with $\perp$ since a pair $(fid, tid)$ doesn't exist in $T_Q$, while $\phi(O_\Gamma)$'s answer is different than $\perp$.
- Queries to $P^{-1}$ on an $fid$ in the range of $G_\Delta$ that was not created via a query to $G_\Delta$: For such queries, the simulated $\Delta$-oracle replies with $\perp$ since a pair $(fid, tid)$ doesn't exist in $T_Q$, while $\phi(O_\Gamma)$'s answer is different from $\perp$.

We define the event $\mathsf{FIND}$ to be true if and only if $D_2$ makes a query to its oracle involving an $fid$ that was not created via a query to $G_\Delta$ or a $tid$ such that $Q^{-1}(tid)$ was not created via a query to $G_\Delta$. Note that finding an $fid$ in the range of $G_\Delta$ without making a query to $G_\Delta$ or finding a $tid$ in the range of $Q$ without creating first its preimage via a query to $G_\Delta$ can happen with only a negligible probability (recall that the ranges of $G_\Delta$ and $Q$ are each a negligible fraction of $\{0,1\}^{2n}$) and therefore:

$$\Pr[\mathsf{FIND}] < \frac{1}{p^2(n)}$$

We now examine the behavior of $D_1$ in case $\mathsf{FIND} = false$ and $z = 1$ and show that there exists a polynomial $q(\cdot)$ such that for infinitely many $n$'s, there exist $x_A, x_B \in \{0,1\}^*$ such that:

$$\begin{aligned}\Big|&\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, 1, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right] \\ &- \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, 1, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right]\Big| \geq \frac{1}{q(n)}\end{aligned}$$

Recall that for infinitely many $n$'s, there exist $x_A, x_B \in \{0,1\}^*$ for which Eq. (2) holds. Fix such an $n$ and the corresponding $x_A, x_B \in \{0,1\}^*$.

First, assume that the input of $D_1$ is the random variable $\text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)$:

$$\begin{aligned}&\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right] \\ &= \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}] \\ &\quad + \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \mathsf{FIND}\right] \cdot \Pr[\mathsf{FIND}] \\ &\geq \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}]\end{aligned}$$

In this case, the view of $Bob_1$ that is contained in the input of $D_1$ is also a view of $Bob_2$ in a real-world run of $\Pi_2(1^n, x_A, x_B)$ with oracle access to $O_\Delta = \phi(O_\Gamma)$, since a run of $\Pi_2$ with oracle access to a $\Delta$-oracle $O_\Delta$ is in fact a simulation of a run of $\Pi_1$ with oracle access to a $\Gamma$-oracle $\phi^{-1}(O_\Delta)$. Recall that when $z = 1$, $D_1$ with oracle access to a $\Gamma$-oracle $O_\Delta$ invokes a run of $D_2$ with a simulated $\Delta$-oracle. If $\mathsf{FIND} = false$, $D_1$ returns 1 on $\text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)$ if and only if $D_2$ returns 1 on $VIEW^{\Pi_2,O_\Delta}_{Bob_2}(1^n, x_A, x_B)$ and therefore:

$$\begin{aligned}&\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right] \\ &\geq \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}] \\ &= \Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, VIEW^{\Pi_2,O_\Delta}_{Bob_2}(1^n, x_A, x_B)\right) = 1\right] \cdot \Pr[\neg\mathsf{FIND}]\end{aligned}$$

Now, assume that the input of $D_1$ is the value of the random variable $\text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, z)$:

$$\begin{aligned}&\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right] \\ &= \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}] \\ &\quad + \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \mathsf{FIND}\right] \cdot \Pr[\mathsf{FIND}] \\ &\leq \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}] \\ &\quad + \Pr[\mathsf{FIND}]\end{aligned}$$

In this case, the view of $Bob_1$ that is contained in the input of $D_1$ is also a view of $Bob_2$ created by $\mathcal{S}_{Bob_2}$ with oracle access to $O_\Delta = \phi(O_\Gamma)$. This claim is true, because $\mathcal{S}_{Bob_2}$ emulates a run of $\mathcal{SIM}$

with $\mathcal{Z}$ and $z = 0$ and trusted party $\mathcal{T}$ with a simulated $\Gamma$-oracle that is equal to $\phi^{-1}(O_\Delta) = O_\Gamma$ and outputs the view of Bob created by $\mathcal{SIM}$. Since the view of Bob depends only on the execution phase, the view of Bob that is contained in the output of $\text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, z)$ is equal to the output of $\mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, s_{x_B})$ regardless of the value of $z$. Therefore, if $\mathsf{FIND} = false$, $D_1$ outputs 1 on $\text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)$ if and only if $D_2$ returns 1 on $\mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, s_{x_B})$ and therefore:

$$
\begin{aligned}
&\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1\right] \\
&\quad \leq \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, 1)\right) = 1 | \neg\mathsf{FIND}\right] \cdot \Pr[\neg\mathsf{FIND}] \\
&\quad\quad + \Pr[\mathsf{FIND}] \\
&\quad = \Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, \mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, s_{x_B})\right) = 1\right] \cdot \Pr[\neg\mathsf{FIND}] + \Pr[\mathsf{FIND}]
\end{aligned}
$$

We conclude that:

$$
\begin{aligned}
&\Big|\Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{REAL}^{O_\Gamma}_{\Pi_1,\mathcal{A},\mathcal{Z}}(n, x_A, x_B, z)\right) = 1\right] \\
&\quad\quad - \Pr\left[D_1^{O_\Gamma}\left(1^n, x_A, x_B, \text{IDEAL}^{O_\Gamma}_{OT_1^2,\mathcal{SIM},\mathcal{Z}}(n, x_A, x_B, z)\right) = 1\right]\Big| \\
&\quad \geq \Big|\Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, VIEW^{\Pi_2, O_\Delta}_{Bob_2}(1^n, x_A, x_B)\right) = 1\right] \cdot \Pr[\neg\mathsf{FIND}] \\
&\quad\quad - \Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, \mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, s_{x_B})\right) = 1\right] \cdot \Pr[\neg\mathsf{FIND}] - \Pr[\mathsf{FIND}]\Big| \\
&\quad = \Big|\Pr[\neg\mathsf{FIND}] \cdot \Big(\Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, VIEW^{\Pi_2, O_\Delta}_{Bob_2}(1^n, x_A, x_B)\right) = 1\right] \\
&\quad\quad - \Pr\left[D_2^{O_\Delta}\left(1^n, x_A, x_B, \mathcal{S}^{O_\Delta}_{Bob_2}(1^n, x_B, s_{x_B})\right) = 1\right]\Big) - \Pr[\mathsf{FIND}]\Big| \\
&\quad > (1 - \frac{1}{p^2(n)}) \cdot \frac{1}{p(n)} - \frac{1}{p^2(n)} = \frac{1}{p(n)} - \frac{1}{p^2(n)} - \frac{1}{p^3(n)} \geq \frac{1}{q(n)}
\end{aligned}
$$

for some positive polynomial $q(\cdot)$. We conclude that $\Pi_2 = \langle Alice_2, Bob_2 \rangle$ is statically secure relative to $O_\Delta$ and this completes the proof of Theorem 4.6. ■

**Remark 4.7** *Theorem 4.6 is true only for* random *$\Gamma$-oracles. Specifically, if $O_\Gamma$ is not a random $\Gamma$-oracle, then the claim that finding an $fid$ in the range of $G_\Gamma$ without making a query to it can happen only with negligible probability does not necessarily hold, and therefore the theorem is not necessarily true for an arbitrary $\Gamma$-oracle.*

The following corollary is obtained from Theorem 4.6 by taking oblivious transfer as a special case:

**Corollary 4.8** *If there exists a protocol $\Pi_1$ that securely computes the $OT_1^2$ functionality in the presence of adaptive semi-honest adversaries relative to a random $\Gamma$-oracle $O_\Gamma$, then the protocol $\Pi_2$ defined above securely computes the $OT_1^2$ functionality in the presence of static semi-honest adversaries relative to $\phi(O_\Gamma)$.*

## 4.4 No Static $OT_1^2$ Relative to $\Delta$ Oracles

For the next step of our proof, we show that static $OT_1^2$ does not exist relative to most $\Delta$ oracles. In order to do this, we show that key agreement does not exist relative to most $\Delta$ oracles, and then derive the result from the fact that secure $OT_1^2$ implies key agreement. In order to show that key agreement does not exist relative to most $\Delta$ oracles, we show that a $\Delta$-oracle can be replaced with a "plain random oracle", with at most a negligible difference. Thus, the results of [25] for key agreement relative to a plain random oracle hold also relative to a $\Delta$ oracle. We begin by formally defining a random oracle type, denoted $\rho$, and show its relationship to $\Delta$-oracles.

**$\rho$-oracles.** We define a $\rho$-oracle to be an oracle with the following functions:

- $G_\rho$ is an injective function from $\{0,1\}^n$ to $\{0,1\}^{2n}$.
- $G_{TEST}$ is a function that returns a string in $\{0,1\}^n$ on inputs in the range of $G_\rho(\cdot)$. For any other input, it returns $\perp$. Note that $G_{TEST}$ is in fact a tool for examining whether a string of size $2n$ is in the range of $G_\rho$ or not.[7]
- $F_P$ is a function that on a triple $(I,k,x) \in \{0,\ldots,5\} \times \{0,1\}^{2n} \times \{0,1\}^{\frac{n}{2}}$ returns a string $y \in \{0,1\}^{\frac{n}{2}}$. Note that for a given $I$ and $k \in \{0,1\}^{2n}$, $F_P(I,k,\cdot)$ is a function from $\{0,1\}^{\frac{n}{2}}$ to $\{0,1\}^{\frac{n}{2}}$.
- $F_Q$ is a function that on a pair $(I,x) \in \{0,\ldots,5\} \times \{0,1\}^n$ returns a string $y \in \{0,1\}^n$. Thus, for a given $I$, $F_Q(I,\cdot)$ is a function from $\{0,1\}^n$ to $\{0,1\}^n$.

Note that the output of $G_\rho$ is an $fid$ – or symmetric key $k$ – of length $2n$ which defines 6 random functions $F_P(0,k,\cdot),\ldots,F_P(5,k,\cdot)$ which are then used to simulate the $P$ permutation of a $\Delta$-oracle, using Luby-Rackoff. Likewise, the index $I$ in $F_Q$ is used for deriving 6 different function for Luby-Rackoff (there is no "secret key" $k$ for $F_Q$ because it is used for simulating the $Q$ permutation in a $\Delta$ oracle which is not keyed).

We denote by $\mathcal{U}_\rho$ the uniform distribution on $\rho$-oracles. Namely, we say that a $\rho$-oracle $O_\rho = (G_\rho, G_{TEST}, F_P, F_Q)$ is distributed according to $\mathcal{U}_\rho$ if $G_\rho$ is a uniformly distributed injective function from $\{0,1\}^n$ to $\{0,1\}^{2n}$, $G_{TEST}$ is a uniformly distributed function from the range of $G_\rho$ to $\{0,1\}^n$ (and for inputs not in the range of $G_\rho$, it returns $\perp$), $F_P$ is a uniformly distributed function from $\{0,\ldots,5\} \times \{0,1\}^{2n} \times \{0,1\}^{\frac{n}{2}}$ to $\{0,1\}^{\frac{n}{2}}$ and $F_Q$ is a uniformly distributed function from $(I,x) \in \{0,\ldots,5\} \times \{0,1\}^n$ to $\{0,1\}^n$. We sometimes use the phrase "$O_\rho$ is a random $\rho$-oracle" as an abbreviation for "$O_\rho$ is distributed according to $\mathcal{U}_\rho$".

**$\Delta$-oracles versus $\rho$-oracles.** We now use the Luby-Rackoff construction [30] to replace a random $\Delta$-oracle with a random $\rho$-oracle. We stress that unlike Corollary 4.2, the distributions are only computationally indistinguishable.

**Definition 4.9 (Feistel Permutation)** *Let $f : \{0,1\}^l \rightarrow \{0,1\}^l$ be a function and let $x_1, x_2 \in \{0,1\}^l$. $DES_f$ is the permutation defined by $DES_f(x_1,x_2) \stackrel{\text{def}}{=} (x_2, x_1 \oplus f(x_2))$. $DES_{f_1,\ldots,f_k}$ is the permutation defined by $DES_{f_1,\ldots,f_k}(x_1,x_2) \stackrel{\text{def}}{=} DES_{f_2,\ldots,x_k}(DES_{f_1}(x_1,x_2))$.*

It is not difficult to see that inverting a Feistel permutation is no harder than computing it, as $DES_f^{-1}(y_1,y_2) = (y_2 \oplus f(y_1), y_1)$. Intuitively, a Feistel permutation upon a random $\rho$-oracle can be used in order to obtain an oracle that behaves like a $\Delta$-oracle. Formally, for a given $\rho$-oracle $O_\rho = (G_\rho, G_{TEST}, F_P, F_Q)$, an $fid \in \{0,1\}^{2n}$ and $x_1, x_2 \in \{0,1\}^{\frac{n}{2}}$, we define six functions: $f_0 = F_P(0,fid,\cdot)$, $f_1 = F_P(1,fid,\cdot)$, $f_2 = F_P(2,fid,\cdot)$, $f_3 = F_P(3,fid,\cdot)$ ,$f_4 = F_P(4,fid,\cdot)$ and $f_5 =$

[7] It was shown in [19] that the black-box separation of [25] holds when $G_{TEST}$ is added to the oracle defined in [25].

$F_P(5, fid, \cdot)$. Then, the permutation $PDES$ relative to a given oracle $O_\rho$, that simulates the $P$ permutation in the $\Delta$-oracle, is defined by

$$PDES_{O_\rho,fid}(x_1, x_2) \stackrel{\text{def}}{=} DES_{f_0,\ldots,f_5}(x_1, x_2)$$

Note that $PDES_{O_\rho,fid}$ is a permutation over $\{0,1\}^n$ (similar to $P(fid,\cdot)$ in a $\Delta$-oracle). Let $PDES^{-1}_{O_\rho,fid}$ be the inverse permutation. Similarly, for a given $\rho$-oracle $O_\rho = (G_\rho, G_{TEST}, F_P, F_Q)$ and for $x_1, x_2 \in \{0,1\}^n$ we define $g_0 = F_Q(0,\cdot), \ldots, g_5 = F_Q(5,\cdot)$. (Recall that $Q$ oracle queries in a $\Delta$-oracle are not keyed and thus when simulated using $F_Q$ in a $\rho$-oracle, no key is used.) We define:

$$QDES_{O_\rho}(x_1, x_2) \stackrel{\text{def}}{=} DES_{g_0,\ldots,g_5}(x_1, x_2)$$

As above, $QDES_{O_\rho}$ is a permutation over $\{0,1\}^{2n}$ (similar to $Q$ in a $\Delta$-oracle). Let $QDES^{-1}_{O_\rho}$ be the inverse permutation.

We define a mapping $\psi$ from $\rho$ to $\Delta$ oracles. Let $O_\rho = (G_\rho, G_{TEST}, F_P, F_Q)$ be a $\rho$-oracle. Then $\psi(O_\rho) = (G_\Delta, P, P^{-1}, Q, Q^{-1})$ is the following $\Delta$-oracle:

- For every $r \in \{0,1\}^n$, $G_\Delta(r) = G_\rho(r)$
- For every $fid \in Range(G_\Delta)$ and all $x \in \{0,1\}^n$, $P(fid, x) = PDES_{O_\rho,fid}(x)$
- For every $fid \notin Range(G_\Delta)$ and for every $x \in \{0,1\}^n$, $P(fid, x) = \perp$
- For every $fid \in Range(G_\Delta)$, $Q(fid) = QDES_{O_\rho}(fid)$
- For every $fid \notin Range(G_\Delta)$, $Q(fid) = \perp$
- $P^{-1}$ and $Q^{-1}$ are the inverse functions of $P$ and $Q$

We denote by $\psi(\mathcal{U}_\rho)$ the distribution where a random $\rho$-oracle is chosen and then $\psi$ is applied to it. The following claim states that access to a random $\Delta$-oracle $O_\Delta$ is essentially the same as access to a $\Delta$-oracle $\psi(O_\rho)$, when $O_\rho$ is random.

**Theorem 4.10 ([12])** *There exists a simulator $\mathcal{S}$ and a negligible function $\mu$, such that for every machine $D$ with unbounded running time which makes a polynomial number of queries,*

$$\left|\Pr\left[D^{\mathcal{U}_\rho,\psi(\mathcal{U}_\rho)}(1^n) = 1\right] - \Pr\left[D^{\mathcal{S}^{\mathcal{U}_\Delta},\mathcal{U}_\Delta}(1^n) = 1\right]\right| < \mu(n)$$

We remark that [12] refer to a plain random oracle and a plain random permutation, without the additional $fid$ generating and other functions. However, $G_\rho = G_\Delta$ by definition, and so clearly $G_\rho$ can be simulated given $G_\Delta$. Likewise, $G_{TEST}$ can be simulated using $P$ (because the latter returns $\perp$ if the $fid$ is not in the range). We use Theorem 4.10 in order to prove the following theorem:

**Theorem 4.11** *If $\mathcal{P} = \mathcal{NP}$, then relative to measure 1 of $\Delta$-oracles, there does not exist any statically secure protocol for computing the $OT^2_1$ functionality.*

In order to prove Theorem 4.11, we recall the original black-box separation of key agreement from a random oracle, as proven in [25].

**Theorem 4.12 ([25])** *If $\mathcal{P} = \mathcal{NP}$, then given any key-agreement protocol relative to a random $\rho$-oracle*[8]*, for every polynomial $poly(\cdot)$, there exists a polynomial time Eve such that Eve finds all intersection queries with probability $1 - \frac{1}{poly(n)}$.*

We first show that a similar argument holds relative to $\Delta$-oracles (that is, every key agreement protocol relative to a random $\Delta$-oracle can be broken with probability $1 - \frac{1}{poly(n)}$). Then, using the same methods as in [25], we show that relative to measure 1 of $\Delta$-oracles, any key-agreement can be broken in polynomial time. As described in [19], it is possible to construct a secure key agreement from any static oblivious transfer protocol and it is easy to verify that this construction relativizes. Therefore, we conclude that relative to measure 1 of $\Delta$-oracles, there does not exist any statically secure protocol for computing the $OT_1^2$ functionality. We begin by proving the following claim:

**Proposition 4.13** *If $\mathcal{P} = \mathcal{NP}$, then given any key-agreement protocol relative to a random $\Delta$-oracle, for every polynomial $poly(\cdot)$, there exists a polynomial time Eve such that Eve finds all intersection queries with probability $1 - \frac{1}{2poly(n)}$.*

**Proof Sketch:** Let $\langle \mathcal{A}_1, \mathcal{B}_1 \rangle$ be a key-agreement protocol relative to random $\Delta$-oracles. We use $\langle \mathcal{A}_1, \mathcal{B}_1 \rangle$ to construct a key-agreement protocol $\langle \mathcal{A}_2, \mathcal{B}_2 \rangle$ relative to random $\rho$-oracles. Recall that $\mathcal{A}_2$ and $\mathcal{B}_2$ have oracle access to a $\rho$-oracle while $\mathcal{A}_1$ and $\mathcal{B}_1$ have oracle access to a $\Delta$-oracle. The idea is to use the $\rho$-oracle in order to simulate a $\Delta$-oracle while replacing queries to $P$, $P^{-1}$, $Q$ and $Q^{-1}$ by appropriate Feistel permutations obtained from $F_P$ and $F_Q$.

Let $\langle \mathcal{A}_2, \mathcal{B}_2 \rangle$ be the following protocol:

**Protocol 2** *On input $1^n$, $\mathcal{A}_2$ invokes $\mathcal{A}_1$ on $1^n$ and $\mathcal{B}_2$ invokes $\mathcal{B}_1$ on $1^n$. The execution is described below for a party $\mathcal{P}_2$ emulating $\mathcal{P}_1$, and is the same for both $\mathcal{A}_2$ and $\mathcal{B}_2$. In each round:*

- *When $\mathcal{P}_2$ gets the message sent by the other party in the previous round, it sends it to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $r$ to oracle $G_\Delta$, $\mathcal{P}_2$ queries it oracle $G_\rho(r)$, and hands the output to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $P(fid, x)$, $\mathcal{P}_2$ queries its oracle $G_{TEST}$ on $fid$ (recall that $G_{TEST}(fid)$ returns $\perp$ if and only if $fid$ is not in the range of $G_\rho$). If the oracle returns $\perp$, $\mathcal{P}_2$ returns $\perp$ as well. Otherwise, uses its oracle $F_P$ to compute $y = PDES_{O_\rho, fid}(x)$ and hands $y$ to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $P^{-1}(fid, y)$, $\mathcal{P}_2$ queries $G_{TEST}$ on $fid$. If it returns $\perp$, $\mathcal{P}_2$ returns $\perp$ as well. Otherwise, $\mathcal{P}_2$ uses its oracle $F_P$ to compute $x = PDES^{-1}_{O_\rho, fid}(y)$ and hands $x$ to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $Q(fid)$, $\mathcal{P}_2$ queries $G_{TEST}$ on $fid$. If it returns $\perp$, $\mathcal{P}_2$ returns $\perp$ as well. Otherwise, it uses its oracle $F_Q$ to compute $tid = QDES_{O_\rho}(fid)$ and hands tid to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ makes a query $Q^{-1}(tid)$, $\mathcal{P}_2$ uses its oracle $F_Q$ to compute $fid = QDES^{-1}(tid)$ and queries $G_{TEST}(fid)$. If it returns $\perp$, $\mathcal{P}_2$ returns $\perp$ as well. Otherwise, $\mathcal{P}_2$ hands $fid$ to $\mathcal{P}_1$.*
- *If $\mathcal{P}_1$ writes a string $m$ on its outgoing communication tape, $\mathcal{P}_2$ sends $m$ to the other party.*
- *At the end of the protocol, $\mathcal{P}_2$ outputs the output of $\mathcal{P}_1$.*

[8] [25] refer to a single random permutation oracle; however, the same proof can be extended to $\rho$-oracles.

Now, assume $\mathcal{P} = \mathcal{NP}$. Let $poly(\cdot)$ be some polynomial and let $Eve_2$ be as in Theorem 4.12. We use $Eve_2$ to construct an adversary $Eve_1$ for $\langle \mathcal{A}_1, \mathcal{B}_1 \rangle$. $Eve_1$ simply invokes $Eve_2$ and simulates the $\rho$-oracle using the simulator $\mathcal{S}$ guaranteed to exist by Theorem 4.10. Note that if $Eve_1$ outputs a list of intersection queries with probability less than $1 - \frac{1}{2poly(n)}$, then it is possible to distinguish oracles $\mathcal{U}_\rho, \psi(\mathcal{U}_\rho)$ from $\mathcal{S}^{\mathcal{U}_\Delta}, \mathcal{U}_\Delta$ with non-negligible probability. Specifically, given a pair of oracles $(\mathcal{O}_1, \mathcal{O}_2)$ that are distributed according to $\mathcal{U}_\rho, \psi(\mathcal{U}_\rho)$ or $\mathcal{S}^{\mathcal{U}_\Delta}, \mathcal{U}_\Delta$, distinguisher $D$ first invokes a run of $\langle \mathcal{A}_1^{\mathcal{O}_2}, \mathcal{B}_1^{\mathcal{O}_2} \rangle$ and then invokes $Eve_2^{\mathcal{O}_1}$ on the transcript. $D$ outputs 1 if and only if $Eve_2$ outputs all intersection queries. Now, if $(\mathcal{O}_1, \mathcal{O}_2)$ are distributed according to $\mathcal{U}_\rho, \psi(\mathcal{U}_\rho)$ then $Eve_2$ outputs all intersection queries with probability at least $1 - \frac{1}{poly(n)}$, and if $(\mathcal{O}_1, \mathcal{O}_2)$ are distributed according to $\mathcal{S}^{\mathcal{U}_\Delta}, U_\Delta$ then $Eve_2$ outputs all intersection queries with probability less than $1 - \frac{1}{2poly(n)}$. Thus $D$ distinguishes with non-negligible probability. ■

**Remark 4.14** *Theorem 4.10 holds even when $\mathcal{P} = \mathcal{NP}$ since the running time of D is unbounded.*

The following corollary can be proven using the same methods as in [25] (the only difference between it and what was proven in [25] is the type of oracle used):

**Corollary 4.15** *If $\mathcal{P} = \mathcal{NP}$, then for measure 1 of $\Delta$-oracles, any key-agreement protocol can be broken in polynomial time.*

Recalling that the existence of a secure $OT_1^2$ relative to an oracle $\mathcal{O}$ implies the existence of a secure key agreement relative to $\mathcal{O}$, we obtain:

**Corollary 4.16** *If $\mathcal{P} = \mathcal{NP}$, then for measure 1 of $\Delta$-oracles, there does not exist any statically secure protocol for computing the $OT_1^2$ functionality.*

## 4.5 Concluding the proof

Corollary 4.8 states that if there exists an adaptively secure protocol for $OT_1^2$ relative to a given $\Gamma$ oracle $\mathcal{O}$, then there exists a statically secure protocol for $OT_1^2$ relative to the oracle $\phi(\mathcal{O})$. Now, by Theorem 4.11, for measure 1 of $\Delta$ oracles, there exists no statically secure $OT_1^2$. Using the fact that $\phi$ is a bijection (Claim 4.1), we conclude that for measure 1 of $\Gamma$ oracles, there exists no adaptively secure $OT_1^2$. That is, we have the following:

**Theorem 4.17** *If $\mathcal{P} = \mathcal{NP}$, then for measure 1 of $\Gamma$-oracles, there does not exist any adaptively secure protocol for computing the $OT_1^2$ functionality.*

Similarly to [25], we derive an oracle separation of enhanced trapdoor permutations form adaptively secure $OT_1^2$ (even for semi-honest adversaries):

**Corollary 4.18** *There exists an oracle relative to which enhanced trapdoor permutations exist, but not adaptively secure $OT_1^2$.*

**Proof:** Let $\mathcal{O}$ be a $\mathcal{PSPACE}$-complete oracle combined with a random $\Gamma$-oracle. Enhanced trapdoor permutations exist relative to $\mathcal{O}$ whereas adaptively secure $OT_1^2$ does not, as we have shown. ■

**Acknowledgements.** We thank Omer Reingold for helpful discussions.

# References


[1] D. Beaver. Adaptive Zero Knowledge and Computational Equivocation. In 28*th STOC*, pages 629–638, 1996.

[2] D. Beaver. Plug and play encryption. In *Advances in Cryptology - Crypto '97*, pages 75–89, 1997.

[3] D. Beaver. Adaptively Secure Oblivious Transfer. In *ASIACRYPT'98*, Springer-Verlag (LNCS 1514), pages 300–314, 1998.

[4] M. Bellare, S. Micali, and R. Ostrovsky. Perfect Zero-Knowledge in Constant Rounds. In 22*nd STOC*, pages 482-493, 1990.

[5] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, 133–137, 1982.

[6] M. Blum. How to Prove a Theorem So No One Else Can Claim It. *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, USA.

[7] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[8] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.

[9] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively secure multiparty computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 639–648, 1996.

[10] Yan. Chang, C. Hsiao and C. Lu: On the Impossibilities of Basing One-Way Permutations on Central Cryptographic Primitives. *ASIACRYPT 2002*: 110-124, 2002,

[11] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002. Full version available at `http://eprint.iacr.org/2002/140`.

[12] J.S. Coron, J. Patarin and Y. Seurin. The Random Oracle Model and the Ideal Cipher Model are Equivalent. In *CRYPTO 2008*, Springer-Verlag (LNCS 5157), pages 1–20, 2008.

[13] I. Damgård. On the existence of bit commitment schemes and zero- knowledge proofs. In *Proc.CRYPTO '89*, pages 17–27, 1989.

[14] I. Damgård. Interactive hashing can simplify zero-knowledge protocol design without computational assumptions. In *Proc.CRYPTO '93*, pages 100–109, 1993.

[15] I. Damgård, J. B. Nielsen. Improved Non-committing Encryption Schemes Based on a General Complexity Assumption. In *CRYPTO '00*, pages 432–450, 2000.

[16] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.

[17] U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 526–544, 1989.

[18] R. Gennaro, Y. Gertner and J. Katz, Luca Trevisan. Bounds on the Efficiency of Generic Cryptographic Constructions. *SIAM Journal on Computing*, 35(1):217–246, 2005.

[19] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The Relationship Between Public Key Encryption and Oblivious Transfer. In the 41*st FOCS*, page 325–335, 2000.

[20] Y. Gertner, T. Malkin and S. Myers. Towards a Separation of Semantic and CCA Security for Public Key Encryption. In the 4*th TCC*, Springer-Verlag (LNCS 4392), pages 434–455, 2007.

[21] Y. Gertner, T. Malkin and O. Reingold. On the Impossibility of Basing Trapdoor Functions on Trapdoor Predicates. In the 42*nd FOCS*, pages 126–135, 2001.

[22] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[23] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, 38(1):691–729, 1991.

[24] R. Impagliazzo and M. Luby. One-way Functions are Essential for Complexity Based Cryptography. In the 30*th FOCS*, pages 230–235, 1989.

[25] R. Impagliazzo and S. Rudich. Limits on the Provable Consequences of One-way Permutations. In 21*st STOC*, pages 44–61, 1989.

[26] T. Itoh, Y. Ohta, and H. Shizuya. A Language-Dependent Cryptographic Primitive. *Journal of Cryptology*, 10(1):37–49, 1997.

[27] B. Kapron, L. Malka, and V. Srinivasan. A characterization of non-interactive instance-dependent commitment-schemes (NIC). In *Proc. ICALP 2007*, pages 328–339, 2007.

[28] J. Kahn, M. Saks and C. Smyth. A Dual Version of Reimer's Inequality and a Proof of Rudich's Conjecture. *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, p.98, July 04-07, 2000.

[29] J.H. Kim, D.R. Simon and P. Tetali. Limits on the Efficiency of One-Way Permutation-Based Hash Functions. In the 40*th FOCS*, pages 535–542, 1999.

[30] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[31] M. Fischlin. On the Impossibility of Constructing NonInteractive StatisticallySecret Protocols From any Trapdoor OneWay Function. *Cryptographers' Track — RSA 2002*, LNCS vol. 2271, pp. 79–95, 2002.

[32] D. Micciancio and S. Vadhan. Statistical Zero-Knowledge Proofs with Efficient Provers: Lattice Problems and More. In *CRYPTO 2003*, Springer-Verlag (LNCS 2729), pages 282–298, 2003.

[33] D. Micciancio, S.J. Ong, A. Sahai and S. Vadhan. Concurrent Zero Knowledge without Complexity Assumptions. In *TCC 2006*, Springer-Verlag (LNCS 3876), pages 1–20, 2006.

[34] M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[35] M. H. Nguyen and S. Vadhan. Zero knowledge with ecient provers. In *Proc. 38th STOC*, pages 287–295, 2006.

[36] S.J. Ong and S. Vadhan. An Equivalence between Zero Knowledge and Commitments. In the 5*th TCC*, Springer-Verlag (LNCS 4948), pages 482–500, 2008.

[37] S. Rudich. The Use of Interaction in Public Cryptosystems (Extended Abstract). *CRYPTO 91*, 242–251, 1991.

[38] O. Reingold, L. Trevisan and S.P. Vadhan. Notions of Reducibility between Cryptographic Primitives. In the 1*st TCC*, Springer-Verlag (LNCS 2951), pages 1–20, 2004.

[39] D.R. Simon. Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions? In *EUROCRYPT 1998*, Springer-Verlag (LNCS 1403), pages 334–345, 1998.

[40] S.P. Vadhan. An Unconditional Study of Computational Zero Knowledge. In the 45*th FOCS*, pages 176–185, 2004.