Positive Results for Concurrently Secure Computation in the Plain Model

Vipul Goyal
Microsoft Research, India
Email: vipul@microsoft.com

Abstract

We consider the question of designing concurrently self-composable protocols in the plain model. We first focus on the minimal setting where there is a party P_1 which might interact with several other parties in any unbounded (polynomial) number of concurrent sessions. P_1 holds a single input x which it uses in all the concurrent sessions. An analogy is a server interacting with various clients at the same time. In this "single input" setting, we show that many (or even most) functionalities can be securely realized in the plain model. More precisely, we are able to realize all functionalities except ones which are a (weak form of) cryptographic unpredictable function. We complement our positive result by showing an impossibility result in this setting for a functionality which evaluates a pseudorandom function.

Our security definition follows the standard ideal/real world simulation paradigm (with no super polynomial simulation etc). There is no apriori bound on the number of concurrent executions.

We also show interesting extensions of our positive results to the more general setting where the honest parties may choose different inputs in different session (even adaptively), the roles that the parties assume in the protocol may be interchangeable, etc.

Prior to our work, the only positive results known in the plain model in the fully concurrent setting were for zero-knowledge.

1 Introduction

General positive results for secure computation were obtained more than two decades ago [Yao86, GMW87]. These results were for the setting where each protocol execution is done in isolation. With the proliferation of the network setting (and especially internet), an ambitious effort to generalize these results was started. The study of concurrent zero-knowledge (ZK) was initiated by Dwork, Naor and Sahai [DNS98] with a protocol soon proposed in the plain model by Richardson and Kilian [RK99]. A sequence of works studied the round complexity of concurrent ZK [CKPR01, KP01, PRS02] (see also [Bar01]). In addition, a protocol for the "interchangeable role" setting (where the same party might play prover in one session and verifier in another) was proposed by Barak, Prabhakaran and Sahai [BPS06] (see also [LPTV10]).

However other than the above encouraging results for the zero-knowledge functionality, there are no known positive results in the setting where there could be any unbounded polynomial number of concurrent sessions (referred to as the fully concurrent setting). In fact, far reaching impossibility results were shown in a series of works [CKL06, Lin03b, Lin08, BPS06]. These results refer to the "plain model" where the participating parties are not required to trust any external entity, they have no prior communication among themselves, etc.

To circumvent these results and obtain protocols secure in the setting of concurrent executions, one line of work has studied various "setup assumptions" where, for example, a trusted party publishes a uniformly chosen string or the participating parties may exchange physical tamper proof hardware tokens etc (see for example [CLOS02, BCNP04, Kat]). Another interesting line of works has studied weaker security definitions while still remaining in the plain model [Pas03, PS04, BS05, MPR06, CLP10].

In this paper, we focus on obtaining standard security guarantees in the plain model. Relevant to our paper is the line of works on obtaining concurrent self-composition in the so called bounded concurrent setting [Lin03a, PR03, Pas04]. In this setting, there is an apriori fixed bound on the total number of concurrent sessions in the system (and the protocol in turn might be dependent on this bound). This state of affairs raises the following natural question: *"Can we obtain interesting positive results for functionalities other than zero-knowledge in the fully concurrent setting?"*

Our Results. We first discuss our results for what we call the "single input setting" and then discuss a generalization.

`Concurrently Secure Computation with a Single Input.` We consider the setting where there is a single party P_1 which might interact with several other parties in any unbounded (polynomial) number of concurrent sessions. The party P_1 holds a single input x which it uses in all the concurrent sessions (if honest). However if P_1 is dishonest, there are no restrictions on how it behaves (except that it has be PPT). An example is a server (e.g., holding a password file) interacting with several clients concurrently (to authenticate them). We refer to this as the "single input" setting (a more precise definition is given in section 2.2).

In this setting, we show that many (or even most) functionalities can be securely realized in the plain model. More precisely, we are able to realize all functionalities except which are what we call a worst-case hard unpredictable function (W-UPF). Very roughly, a W-UPF is capable of generating an output with pseudoentropy much larger than allowed information theoretically on at least some inputs (see section 2.1 for a more formal definition).[1]

We complement this positive result by showing an (unconditional) impossibility result in our setting for a functionality which evaluates a pseudorandom function on a committed key. In more detail, let COM be a non-interactive statistically binding commitment scheme and f be a (keyed) pseudorandom function. Our functionality is parameterized by a string σ. It takes as input k, r from P_1 and x from P_2. It first checks if $\sigma = COM(k, r)$, if not, it outputs $\perp$ to P_2. Else, it outputs $f(k, x)$ to P_2. In fact, it suffices to use a notion of worst case hard pseudorandom function (thus matching our positive results more closely). The impossibility holds both w.r.t. black-box as well as non-black-box simulation and represents the first negative result for an arguably natural functionality in the setting of fixed inputs. The only previous negative result known [BPS06] was for a rather contrived functionality (which allowed two modes; one for execution of zero-knowledge and another for oblivious transfer).

To prove our main negative result, the key technical tool is a new garbled circuit construction where the garbled circuit is executed by the receiver by a single k-out-of-2k OT (as opposed to k execution of 1-out-of-2 OT). We

[1]Jumping ahead, the reason why we refer to worst case hard primitives (rather than average case) stems from the fact that the standard definition of secure computation requires the ideal world simulator to work for *every* honest party input (as opposed to just a random one).

believe our construction is of independent interest since, to our knowledge, all previous constructions of protocols based on garbled circuit involved parties executing k (1-out-of-2) OTs while executing a single k-out-of-2k OT may, e.g., be more efficient.[2] To be more precise, we actually provide a construction of *one time programs* [GKR08] based on a *single k-time-memory hardware token* (as opposed to requiring several one-time-memory hardware tokens per program).

Generalization of our Positive Result. We show that the positive results discussed above can be significantly generalized. Our construction only requires that the ideal world satisfy what we call the *key technical property*. The rough intuition behind the key technical property is as follows. In the ideal world execution, we require the existence of a *predictor* which, given information about (adversary's) input/output tuples for sufficiently many (ideal world) sessions, starts to be able to "predict" the output of the ideal functionality in some sessions with a noticeable probability (the exact definition is slightly technical and is discussed in Section 2.3).[3] The positive results for the single input setting were then obtained by simply showing that this ideal world condition is satisfied for all functionalities except for those that behave as a W-UPF.

However, the key technical property (KTP) is quite general and is naturally satisfied in many other settings. We consider the very general setting where the the honest parties may have different (possibly *adaptively chosen*) inputs in different sessions, there may be multiple parties in a protocol session with all of them getting different outputs, the adversary may choose to corrupt parties with different roles in different sessions (i.e., the interchangeable role setting [Lin08]), etc. We prove that the ideal world would still satisfy KTP as long the size of the total "state" of the honest parties in the ideal world is bounded and the ideal world is "hardness-free". In more detail, first we require the existence of an apriori bounded length string S which describes the state of all the honest parties at the beginning of the ideal world execution. This condition is naturally satisfied when, e.g., the total number of honest parties (with each party participating in any unbounded polynomial number of sessions) is apriori bounded. Secondly, a hardness-free ideal world requires that the code "consisting" of the ideal world functionality and the (ideal world) honest parties does not behave as a W-UPF (see appendix 4 for more formal details). Thus, this gives us a *general positive result* for all bounded-size hardness-free ideal worlds.

Interpretations and Applications of Our Positive Results. A simple example of a setting where KTP is satisfied is the well studied setting of concurrent self-composition in the bounded concurrent setting [Lin03a, PR03, Pas04]. In fact our techniques only make use of black-box simulation while all previously reported protocols in the bounded concurrent setting use non-black-box simulation techniques introduced by [Bar01]. On the flip side we note that the protocols in [PR03, Pas04] are constant round while ours might require a large polynomial number of rounds. Another well studied setting where KTP is satisfied is that of concurrent zero-knowledge.

We believe the conceptual insight in this paper improves our basic understanding of when concurrently secure computation is possible. Our results not only subsume the known positive results on fully concurrent zero-knowledge and bounded concurrent secure computation, but rather *present a unified explanation* of why it might be possible to obtain such results. Prior to our work, the intuition behind why these two tasks are possible might have looked very different.

The above is, of course, in addition to our main contribution which is obtaining a host of new positive results in the fully concurrent setting. To start with, we remark that our results imply the first construction of a concurrent password based key exchange (PAKE) protocol in the plain model with standard ideal/real world security guarantees. The only previous construction of (fully) concurrent PAKE in the plain model was given recently by Goyal, Jain and Ostrovsky [GJO10]. However the construction in [GJO10] was according to the original definition of Goldreich and Lindell [GL01] which is a **weaker** definition (in comparison to the standard ideal/real world definition). Prior to the work of [GJO10], obtaining a fully concurrently secure protocol in the plain model according to *any* reasonable definition was an open problem (despite a number of works studying PAKE in the concurrent setting, c.f., [KOY01, CHK$^+$05, BCL$^+$05, GL03]). We note that the setting in ours as well as in [GJO10] is that of a single fixed input.

[2] It has been argued that the key efficiency bottleneck in secure two-party computation based on garbled circuits is the execution of oblivious transfers since the rest just involves symmetric key operations (see, e.g., [PSW09]).

[3] Note that being able to predict the output of a function is very different from requiring that the function is learnable. A simple example is a point function for which it is easy to predict the output on almost every input. Also, note that the existence of such a predictor does not mean that the output has to come from a polynomial-size domain; see section 2.3.

We also get positive results for a number of other functionalities studied previously. One example is that of *private database search* where a party holds a database and another party wants to search and get the matching entries without revealing its search criteria (such as a keyword). Problems such as private information retrieval [CGKS95, KO97], pattern matching [HL08b], oblivious document and database search [HL08a], etc are special instances of the general problem of private database search. Other examples of well studied problems are secure set intersection [FNP04], private matching [FNP04], securely computing the k-th ranked element [AMP04], etc. For all of these functionalities, we get concurrently secure protocols in the single input setting. We refer the reader to section 2.3 for an understanding of why these functionalities might satisfy the KTP.

In general, in the (large) body of published literature studying specific functionalities of interest, we found that almost all of them indeed have hardness-free ideal worlds (i.e., in the ideal world, the trusted party is not required to perform any cryptographic operations, etc). Some functionalities are naturally seen as an interaction between a client and a server where only the server accepts concurrent sessions. For such functionalities, the single input setting is already very realistic. For example, we get positive results for concurrent private database search where there is a server holding the database interacting with multiple clients each of which is holding a search criteria (in particular, this also implies a similar positive result for concurrent private information retrieval, concurrent pattern matching, etc).

Some functionalities however are more symmetric (such as secure set intersection). Hence, there is motivation to also study the setting where there may be multiple honest parties holding different inputs and accepting concurrent sessions. Towards that end, we remark that a positive result may be obtained even in this setting if the bounded-size ideal world requirement is satisfied (it would be, e.g., if the total number of honest parties and the size of their initial states is bounded).

Overview of our Construction. A well established approach to constructing secure computation protocols in the standalone setting is to use the GMW compiler: take a semi-honest secure computation protocol and "compile" it with zero-knowledge arguments. The natural start point of our construction is to follow the same principles in the concurrent setting: somehow compile a semi-honest secure computation protocol with a concurrent zero-knowledge protocol (for security in more demanding settings, compilation with concurrent non-malleable zero-knowledge [BPS06] may be required). Does such an approach (or minor variants) already give us protocols secure according to the standard ideal/real world definition in the plain model?

The fundamental problem with this approach is the following. We note that the known concurrent zero-knowledge simulators (in the fully concurrent setting) work by rewinding the adversarial parties. In the concurrent setting, the adversary is allowed to control the scheduling of the messages of different sessions. Then the following scenario might occur:

- Between two messages of a session s_1, there might exist another entire session s_2
- When the simulator rewinds the session s_1, it may rewind past the beginning of session s_2
- Hence throughout the simulation, the session s_2 may be executed multiple times from the beginning
- Every time the session s_2 is executed, the adversary may choose a different input (e.g., the adversary may choose his input in session s_2 based on the entire transcript of interaction so far).
- In such a case, the simulator would have to query the ideal functionality for session s_2 more than once. However note that for every session, simulator gets to query the ideal functionality only once!

Indeed, some such problem is rather inherent as indicated by various impossibility results [Lin08, BPS06]. This is where the fact that our ideal world satisfies the key technical property comes in. Very roughly, KTP requires the existence of a *predictor* which is successful in predicting the output of the ideal functionality with a noticeable probability (under certain conditions). The basic idea is as follows. The rewinding strategy of the simulator would lead to a "main thread" and several "look ahead threads" (following the terminology of [PRS02]). Whenever the simulator needs to make a call to the ideal functionality in a look-ahead thread, it uses the predictor instead. This would help the simulator achieve the goal of querying the ideal functionality only once per session.

Note that the output of the predictor is not guaranteed to be correct in all cases. Furthermore, the adversary might have complete auxiliary information about the input of the honest parties (and hence may distinguish the incorrect output from correct ones). In such a scenario, adversary might change its behavior in the look-ahead thread (or might simply abort). Hence, several look-ahead threads might now fail. In general, the property of indistinguishability between the main thread and the look-ahead threads (which all previous rewinding strategies rely on) does not hold any more. To solve this problem, we rely on and analyze a specific rewinding strategy by Deng, Goyal and Sahai [DGS09]. We choose the number of rewinding opportunities based on the key parameters of the predictor (guaranteed by KTP).

Our initial protocol is based on compilation with concurrent zero-knowledge. However it only satisfies a weaker notion of security which provides concurrent security for first party while only guaranteeing security for the other party in the standalone setting. Our final protocol is based on compilation with concurrent non-malleable zero-knowledge [BPS06]. There are several problems that arise with such a compilation. First, the security of the BPS construction is analyzed only for the setting where all the statements being proven by honest parties are fixed in advance. Secondly, the extractor of BPS-CNMZK is unsuitable for extracting inputs of the adversary since it works after the entire execution is complete on a *session-by-session* basis. Fortunately, these challenges were tackled in a recent work on password based key exchange by Goyal, Jain and Ostrovsky [GJO10]. Goyal et. al. presented an approach which can be viewed as a technique to correctly compile a semi-honest secure protocol with BPS-CNMZK. In our final protocol, we borrow a significant part of the construction presented in [GJO10].

Open Problem. The number of rounds in our protocol depend on the parameters of the predictor associated with the functionality and may be quite large (although still polynomial). We leave it as an open problem to construct more round efficient protocols. Known lower bounds on the round complexity of protocols proven secure using black-box simulation [Lin08] imply that to get round efficient protocol (e.g., to get a protocol with round complexity dependent only on the security parameter), advancements in our understanding of non-black-box simulation techniques [Bar01] will be required. In particular, it seems that we need a construction for zero-knowledge with a non-rewinding simulator in the fully concurrent setting. Obtaining such a construction is currently an open problem.

Organization of the Paper. What follows is an warmup construction which is meant to highlight the new ideas behind our positive result. This construction satisfies a weaker notion of security where the first party is guaranteed concurrent security (while interacting with multiple adversarial parties) while the other one is guaranteed security only in the standalone setting. This construction illustrates the main ideas in our work and is kept at an informal level. Our final construction is an extension of the warmup construction using known techniques [GJO10, BPS06]. A complete description of the final construction along with a full self-contained proof can be found in the Appendix.

2 Model and Definitions

2.1 Preliminaries

In this section, we first define a very minimal cryptographic primitive called worst-case hard one-way functions (W-OWF). The existence of W-OWF is implied by the assumption **NP** $\not\subseteq$ **BPP**.

Definition 1 (Worst-Case Hard One Way Functions) *A function $f : \{0,1\}^n \to \{0,1\}^*$ is a worst-case hard one-way function if it is polynomial time computable but for all uniform probabilistic polynomial time algorithms A, there exists $x \in \{0,1\}^n$ such that $Pr[f(A(f(x))) = f(x)] = negl(n)$*

We also define worst case hard unpredictable functions (W-UPF) which imply the existence of W-OWF. Very roughly, W-UPF are functions which are capable of generating an output with "much higher pseudo-entropy" than the size of the input.

Definition 2 (Worst-Case Hard Unpredictable Functions) *A function $g : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^*$ is a worst-case hard unpredictable function if for all uniform probabilistic polynomial time algorithms A, there exists x, f and at least $100nf$ inputs $r_1, \ldots, r_{100nf}$ such that for all $i \in [100nf]$,*

$$Pr\big[A(r_1, \ldots, r_i, g(x, r_1), \ldots, g(x, r_{i-1})) \neq g(x, r_i)\big] \geq \frac{1}{f}$$

That is, there exists at least $100nf$ input strings for which the output is "somewhat" unpredictable given all previous outputs in the input set (at least for one secret key x).

2.2 Concurrently Secure Computation with a Single Input

We now discuss the model for concurrently secure computation a single input. For elegance, we focus on a quite minimal model of concurrently secure computation and discuss generalizations later on. There is a party P_1 holding a fixed input $x_1 \in \{0,1\}^r$ which it uses in all sessions. There are only two parties with "fixed-roles" in each protocol execution. That is, the party P_1 interacts with various other (possibly adversarial) parties in several concurrent sessions. There could any unbounded (polynomial) number of parties with each party participating in any unbounded (polynomial) number of protocol session. The adversarial parties are free to behave in any way they want (and in particular, adaptive choose different inputs for different sessions). The functionality being computed is deterministic and non-reactive and is represented by $\mathcal{F} : \{0,1\}^r \times \{0,1\}^s \rightarrow \perp \times \{0,1\}^t$ (i.e., only one of the parties gets an output). We note that our model and results can be significantly generalized to consider different (adaptively chosen) inputs in different sessions, interchangeable roles, multi-party case with multiple parties getting output etc. These generalizations are considered in Section 4.

We consider a static adversary that chooses whom to corrupt before execution of the protocol. We only consider *computational* security and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\stackrel{c}{\equiv}$, and the security parameter by n. Further, we only consider "security with abort". To formalize the above requirements and define security, we follow the standard paradigm for defining secure computation (see also [Lin08]). We define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* by an adversary in the ideal model. More details follow.

IDEAL MODEL. In the ideal model, there is a trusted party that computes the desired functionality $\mathcal{F}$ based on the inputs handed to it by the players. An execution in the ideal model proceeds as follows:

I. Inputs: The party P_1 holds a fixed input x_1. The input of the party P_2 maybe different for each session and is denoted by $x_2[\ell]$ for session index ℓ.

II. Session initiation: If the adversary wishes to initiate a session, it sends a $(\mathsf{start\text{-}session})$ message to the trusted party. On receiving a message of the form $(\mathsf{start\text{-}session})$, the trusted party sends $(\mathsf{new\text{-}session}, \ell)$ to both P_1 and P_2 , where ℓ is the index of the new session.

III. Honest parties send inputs to trusted party: Upon receiving $(\mathsf{new\text{-}session}, \ell)$ from the trusted party, an honest party P_i sends its real input along with the session identifier. More specifically, P_i sets its session ℓ input $x_i[\ell]$ to be the x_1 if $i = 1$ and $x_2[\ell]$ otherwise. P_i then sends $(\ell, x_i[\ell])$ sends to the trusted party.

IV. Corrupted parties send inputs to trusted party: A corrupted party P_i sends a message $(\ell, x'_i[\ell])$ to the trusted party, for any string $x'_i[\ell]$ (of appropriate length) of its choice.

V. Trusted party sends results to adversary: For a session ℓ, when the trusted party has received messages $(\ell, x_1[\ell])$ and $(\ell, x_2[\ell])$, it computes the output $\mathcal{F}(x_1[\ell], x_2[\ell])$. If P_2 is corrupted, it sends $(\ell, \mathcal{F}(x_1[\ell], x_2[\ell]))$ to the adversary. Do nothing if only P_1 is corrupted. If neither of the parties is corrupted, then the trusted party sends the output message $(\ell, \mathcal{F}(x_1[\ell], x_2[\ell]))$ to P_2.

VI. Adversary instructs the trusted party to answer honest players: For a session ℓ where only P_2 is honest, the adversary, depending on its view up to this point, may send the (output, ℓ) message in which case the trusted party sends the output $(\ell, \mathcal{F}(x_1[\ell], x_2[\ell]))$ to P_2.

VIII. Outputs: An honest party always outputs the value that it received from the trusted party. The adversary outputs an arbitrary (PPT computable) function of its entire view (including the view of all corrupted parties) throughout the execution of the protocol.

Let $\mathcal{S}$ be a probabilistic polynomial-time ideal-model adversary that controls a subset M of corrupted parties (M could contain any of the parties playing role P_2 in different session and/or the party playing role P_1). Then the ideal execution of $\mathcal{F}$ (or the ideal distribution) with security parameter n, and auxiliary input z to $\mathcal{S}$ is defined as the output of the honest parties along with the output of the adversary $\mathcal{S}$ resulting from the ideal process described above. It is denoted by $\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(n, z, \{x_1, x_2[\ell]\}_\ell)$.

REAL MODEL. We now consider the real model in which a real two-party secure computation protocol is executed. Let $\mathcal{F}, \{x_1, x_2[\ell]\}_\ell, M$ be as above. Let Σ be the real world secure computation protocol in question. Let $\mathcal{A}$ be probabilistic polynomial-time (PPT) machine such that it controls the parties in set M.

In the real model, a polynomial number (in the security parameter n) of sessions of Σ may be executed concurrently, where the scheduling of all messages throughout the executions is controlled by the adversary. We assume that the communication between the parties takes place over authenticated channels. An honest party follows all instructions of the prescribed protocol, while an adversarial party may behave arbitrarily. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol.

The real concurrent execution of Σ with security parameter n, and auxiliary input z to $\mathcal{A}$ is defined as the output of all the honest parties along with the output of the adversary resulting from the above process. It is denoted as $\text{REAL}^{\Sigma}_{M,\mathcal{A}}(n, z, \{x_1, x_2[\ell]\}_\ell)$.

Having defined these models, we now define what is meant by a concurrently-secure computation protocol in the single input setting.

Definition 3 *Let $\mathcal{F}$ and Σ be as above. Then protocol Σ for computing $\mathcal{F}$ is a concurrently secure computation protocol in the single input setting if for every probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $z \in \{0,1\}^*$, every $\{x_1, x_2[\ell]\}_\ell$ and every M,*

$$\left\{\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(n, z, \{x_1, x_2[\ell]\}_\ell)\right\}_{n \in N} \overset{c}{\equiv} \left\{\text{REAL}^{\Sigma}_{M,\mathcal{A}}(n, z, \{x_1, x_2[\ell]\}_\ell)\right\}_{n \in N}$$

Relaxed Security Notion. Our warm up construction (based on compilation with concurrent zero-knowledge) only satisfies a weaker security notion where P_1 is guaranteed concurrent security while security for a party $P_i, i \neq 1$ is guaranteed only in the standalone setting. We very informally outline the security notion in the following. The ideal and the real world remain as discussed above. However we place conditions on when we require the existence of an ideal world simulator. We wish to provide concurrent security for the party P_1. In particular, consider the case when P_1 is honest and the parties $P_2, P_3, \ldots$ are all corrupt (and interact with P_1 in several concurrent session). We require the existence of a simulator in the ideal world such that the ideal and the real world distribution are indistinguishable. Furthermore, consider the case where there is only a single session between P_1 and P_i such that P_i is honest but P_1 is corrupted. We again require the existence of a simulator in the ideal world such that the ideal and the real world distribution are indistinguishable. In all other cases (in particular when P_1 is corrupt and interacts with several honest parties concurrently), we do not require any guarantees.

2.3 Key Technical Property

In this section, we will define a property of the ideal world experiment (as defined in section 2.2) called the key technical property (KTP). We first introduce some notation. Recall that the adversary may corrupt any subset of parties. Let M denote the subset of corrupted parties. Let there be k sessions. The input of the adversarial party in session ℓ is denoted by $I[\ell]$ while the corresponding output is denoted by $O[\ell]$.

Now, define D disjoint sets $\{S_i\}_{i \in [D]}$ where each S_i is a set of indices of the (ideal world) sessions. The key technical property is defined as follows.

Definition 4 (Key Technical Property) *The key technical property of an ideal world experiment requires the existence of a PPT predictor $\mathcal{P}$ satisfying the following conditions. There exists (D, c) such that for all adversaries and honest parties, all possibilities of $\{S_D\}_D$ as above and all sufficiently large n, there exists i such that,*

$$Pr\left[\forall j \in S_i, \mathcal{P}\left(\{I[\ell]\}_{\ell \leq j}, \{O[\ell]\}_{\ell < j}\right) = O[j]\right] \geq n^{-c}$$

where the probability is taken over the random coins of the predictor. Denote the lower bound on the prediction probability (i.e., n^{-c}) by p.

In other words, KTP roughly states that given sufficient large number of sets containing adversarial inputs and outputs, there is at least one set for which the predictor can correctly predict the adversarial output tuple with a noticeable probability (given the adversarial inputs/outputs in all previous sessions).[4] Note that this also implies that the predictor succeeds with noticeable probability for a *randomly selected* i.

We now give some examples to illustrate this property. These examples provide important intuition about why the key technical property is so general.

Password based key exchange. Consider the following password checking functionality. Say there is a single honest party which might interact with several adversarial parties in an unbounded (polynomial) number of concurrent sessions. The honest party holds a password p and the ideal functionality is such that if both parties input the same password, it outputs 1 to the adversarial party (and $\perp$ otherwise). We note that in this case, the ideal world satisfies the KTP in fact for even $D = 2$. The predictor works as follows. It needs to guess the output in a session given the (actual) inputs/outputs in all previous sessions. If the output in a previous session is 1, then by looking at the *input of the adversary* in that session, the predictor can find out the actual password p. Hence it uses that to correctly compute the output in the next session. However, if the output in all previous sessions is $\perp$, the predictor sets the output in the next session to be $\perp$ as well. It is easy for see that the predictor makes an error in at most a single session. This is because as soon as it makes an error in a session, it can learns the correct password by looking at the input of the adversary in that session. Hence, $D = 2$ suffices in that case. A variant of this functionality is the password based key exchange functionality where, if the passwords match, the ideal functionality outputs a randomly generated key to both parties (and $\perp$ otherwise). This (randomized) functionality similarly satisfies the generalized KTP (see appendix 4) for $D = 2$.

Similar argument can be shown to hold for more general access control functionalities. For example, say that the first party has a list of k valid passwords (or say a list of k valid userID and password tuples). If the password supplied by the second party is in the list, both the parties get a shared secret key and $\perp$ otherwise. In this case, setting $D = k + 1$ would allow the construction of a predictor using similar ideas.

Private database search. In this scenario, the first (honest) party holds a database consisting of k entries. The second (adversarial) party has a predicate $g(\cdot)$ as input and gets as output all database entries on which this predicate evaluates to 1. In this case, the ideal world satisfies the KTP for $D = k + 1$. The predicator is supplied with all previous inputs and outputs and has to produce an answer for a new input g. The predictor looks at the previous outputs, constructs a partial database and then answers g according to this database. Indeed, the output of the predictor maybe incorrect since it might be missing an entry which is in the real database but not in the partial one. However every time the predictor makes a mistake, it learns a new database entry (to be added to the partial database) by looking at the *correct output* for that missing entry. Hence, the predictor can produce an incorrect output at most k times.

Several problems of interest (such as private information retrieval, pattern matching, etc) are instances of the general problem of private database search discussed above. Arguments similar to those used for private database search can be used to construct predictors for other problems such as secure set intersection, computing k-th ranked element, etc (see the introduction for more details).

Bounded concurrent multi-party computation. We now consider the setting where there is an apriori fixed bound k on the total number of concurrent sessions in the ideal world (see, e.g., [Lin03b, PR03, Pas04]). If we set $D > k$, then there is at least one set S_i which will not have any sessions (by the pigeonhole principle). Hence, the predictor trivially succeeds for this set with probability 1.

[4]Jumping ahead, in our proof of security, the simulator will try to rewind a "slot" for successful simulation. A set S_i will correspond to the sessions "encapsulated" in that slot. In order to rewind that slot and execute it till the end, the simulator will need to predict the output of all sessions in set S_i. The partitioning of the sessions in sets $\{S_i\}_{i\in[D]}$ is determined by the adversarial scheduling. More details can be found in the proof of lemma 2.

3 A Warmup Construction

3.1 KTP and Concurrently Secure Computation with a Single Input

Consider an ideal world in the model of concurrently secure computation with a single input (defined in Section 2.2). We now show that the ideal world for all "non-cryptographic" functionalities $\mathcal{F}$ satisfies the key technical property. To be more precise, we prove the following lemma.

Lemma 1 *Consider an ideal world for a functionality $\mathcal{F}$ in the single input setting (section 2.2). If the ideal world does not satisfy the key technical property (definition 4), then the functionality $\mathcal{F}$ must be a W-UPF (definition 2).*

PROOF. The proof of this lemma relies on the fact that if the predictor fails to predict the output in a large number of sessions (even given all previous outputs), the functionality is behaving like a W-UPF (on the given honest party input). Details follow.

If the key technical property is not satisfied, it must be for the case when the party playing the role P_1 is honest with an input $x_1 \in \{0,1\}^r$. This is because otherwise the adversarial parties in the ideal world do not get any output in any session where an honest party is involved (recall that the functionality $\mathcal{F}$ is deterministic and of the form $\mathcal{F} : \{0,1\}^r \times \{0,1\}^s \to \perp \times \{0,1\}^t$). Set $D = rn^2$ and $p = \frac{1}{2}$. The fact that KTP is not satisfied implies that for every predictor $\mathcal{P}$, there exists a choice of D disjoint sets $\{S_D\}_D$ of session indices such that for all $i \in [D]$, with probability at least $\frac{1}{2}$, $\mathcal{P}$ makes an error in at least one session in S_i. We now compute the following called the *error sum* E:

$$\sum_{j \in \{S_1 \cup, \ldots, \cup S_D\}} (\text{Probability that } \mathcal{P} \text{ makes an error in session } j)$$

Clearly, it can be seen that this sum is at least $\frac{1}{2}rn^2$. Now to show that the functionality $\mathcal{F}$ is a W-UPF, we need to show at least $100rf$ sessions such that $\mathcal{P}$ makes an error at least with probability $\frac{1}{f}$. Say $\mathcal{F}$ is not a W-UPF. That implies that for all f', number of sessions where the predictor makes an error with probability at least $\frac{1}{f'}$ is less than $100rf'$. In other words:

there are at most $100r$ session where $\mathcal{P}$ makes an error with probability at least 1
there are at most $100r$ more session where $\mathcal{P}$ makes an error with probability at least $\frac{1}{2}$
there are at most $100r$ more session where $\mathcal{P}$ makes an error with probability at least $\frac{1}{3}$
and so on. Hence the error sum:

$$\begin{aligned} E &\leq 100r(1 + \frac{1}{2} + \frac{1}{3} + \ldots) \\ &\leq 100r \cdot O(\ln(\text{poly}(n))) \qquad \text{(since the number of sessions is polynomial in } n\text{)} \\ &= r \cdot O(n) \end{aligned}$$

However the above is a contradiction since the E is at least $\frac{1}{2}rn^2$. This completes the proof.

3.2 Overview of Our Construction and Simulator

In this section, we describe our protocol Σ for a given two-party functionality $\mathcal{F}$ (extension to the case of multi-party is discussed later). Let P_1 and P_i be two parties with private inputs x_1 and x_i respectively. Let COM denote a non-interactive statistically binding commitment scheme. By WIAOK, we will refer to a witness indistinguishable argument of knowledge. Let Π_{SH} be any *semi-honest* secure two party computation protocol that emulates the functionality $\mathcal{F}$ in question in the stand-alone setting (as per the standard Ideal/Real world definition of secure computation). Let U_η denote the uniform distribution over $\{0,1\}^\eta$, where η is a function of the security parameter.

We shall make use of the DGS preamble [DGS09]. The DGS preamble allows a party to commit to the desired value in a way the simulator can extract that value by rewinding in the concurrent setting (similar PRS [PRS02] and RK [RK99] preambles). The total number of "slots" in the DGS preamble will be $m = \frac{2n^3D^2}{p}$. We now describe our protocol Σ.

I. Input Commitment Phase

1. $P_i \leftrightarrow P_1$: Generate a string $r_i \xleftarrow{\$} U_\eta$ and let $X_i = \{x_i, r_i\}$. Here r_i is the randomness to be used (after coin-flipping with P_1) by P_i in the execution of the protocol Π_{SH}. Using the commitment scheme COM, P_i commits to the string X_i. Denote the commitment by B_i and the decommitment information by β_i (β_i consists of the string X_i and the randomness used to create B_i). Now P_i additionally prepares mn pairs of secret shares $\{\alpha^0_{i,j}, \alpha^1_{i,j}\}_{i\in[n],j\in[m]}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \beta_i$ for all i, j. Using the commitment scheme COM, P_i also commits all its secret shares. Denote these commitments by $\{A^0_{i,j}, A^1_{i,j}\}_{i\in[n],j\in[m]}$. Note that the string β_i will also be called the "preamble secret".

   The party P_i now engages in the execution of a (standalone) computational zero-knowledge argument with P_1 in order to prove that the above commit phase is "consistent" (i.e., all pairs of the secret shares sum up to the same value which is the preamble secret).

   P_i and P_1 now execute a challenge-response phase. For $j \in [m]$:
   (a) $P_1 \to P_i$: Send challenge bits $z_{1,j}, \ldots, z_{n,j} \xleftarrow{\$} \{0,1\}^n$.
   (b) $P_i \to P_1$: Send $\alpha^{z_{1,j}}_{1,j}, \ldots, \alpha^{z_{n,j}}_{n,j}$ along with the relevant decommitment information.

   At the end of this step, party P_i is committed to its input and randomness. Informally speaking, the purpose of this phase is aid the simulator in extracting the adversary's input and randomness in the concurrent setting by rewinding the DGS preamble.

2. $P_1 \leftrightarrow P_i$: Generate a string $r_1 \xleftarrow{\$} U_\eta$ and let $X_1 = \{x_1, r_1\}$. Here r_1 is the randomness to be used (after coin-flipping with P_i) by P_1 in the execution of the protocol Π_{SH}. Using the commitment scheme COM, P_1 commits to X_1; denote this commitments by B_1.

   In addition, P_1 now engages in the execution of a WIAOK with P_i in order to prove that it knows either: (a) a decommitment to the commitment B_1, or, (b) a decommitment to the commitment B_i.

II. Secure Computation Phase. In this phase, the parties run an execution of the semi-honest two party protocol Π_{SH}. Since Π_{SH} is secure only against semi-honest adversaries, parties first run a coin-flipping protocol to force the coins of each party to be unbiased and then "compile" Π_{SH} to enforce honest behavior. More details follow.

Coin Flipping. P_1 and P_i first engage in a coin-flipping protocol. More specifically,

1. $P_1 \to P_i$: P_1 generates $r'_i \xleftarrow{\$} U_\eta$ and sends it to P_i. Define $r''_i = r_i \oplus r'_i$.
2. $P_i \to P_1$: Similarly, P_i generates $r'_1 \xleftarrow{\$} U_\eta$ and sends it to P_1. Define $r''_1 = r_1 \oplus r'_1$.

Now r''_1 and r''_i are the random coins that P_1 and P_i will use in the execution of protocol Π_{SH}.

Protocol Π_{SH}. The parties P_1 and P_i now execute the protocol Π_{SH}. Along with each outgoing message of Π_{SH},
(a) The party P_1 proves using WIAOK that either it has behaved honestly in Π_{SH} so far (based on the input x_1 and randomness r''_1 as defined earlier) or it knows a decommitment to the commitment B_i.
(b) The party P_i proves using a (standalone) computational zero-knowledge argument that it behaved honestly (based on the input x_i and randomness r''_i as defined earlier).

The above protocol satisfies the relaxed security notion for the single input setting as outlined in section 2.2. To prove security, we need to consider the following two arguments. In the standalone setting, we need to exhibit security against a corrupt P_1. Secondly, we need to exhibit concurrent security for an honest P_1 (who may interact with various parties P_i's all of whom may be corrupted). This is the more interesting case and we first give a proof of security for this case is given in the following.

The Simulator. First we provide only a high level overview of the simulator $\mathcal{S}$. The first task of $\mathcal{S}$ is to rewind the DGS preambles (in all concurrent sessions) and extract the preamble secret for each from the adversary. Such rewinding gives rise to a "main thread" and as well as several "look-ahead" threads of execution.

- In the input commitment phase, $\mathcal{S}$ can extract the decommitment of B_i which includes the input/randomness of the adversary (to be used in the protocol Π_{SH}) from the DGS preamble.

- Using this decommitment information, $\mathcal{S}$ can cheat throughout by using it to complete all the WIAOK executions.
- In the secure computation phase, $\mathcal{S}$, by invoking the simulator of the protocol Π_{SH}, can complete the secure computation phase *given the output from the trusted functionality*.
- Note that for any given session, the input of the adversary may be different in different threads (since it might choose its input based on the transcript of the interaction so far). Now to complete the secure computation phase, $\mathcal{S}$ would need to get the corresponding output. To complete the secure computation phases *in the main thread*, $\mathcal{S}$ gets the output simply by querying the trusted party. For all the look-ahead threads, $\mathcal{S}$ gets the output by running the predictor guaranteed by the KTP in the ideal world.

We now give more details of our simulator $\mathcal{S}$.

Detailed Strategy of the simulator. We assume there are a total of k sessions with each session having one DGS preamble. Each of these preambles has $m = \frac{2n^3D^2}{p}$ "slots" with a slot representing a rewinding opportunity. The beginning of a slot is when the simulator gives the challenge, the end of the slot being when it receives the response. In between these two messages, there might be messages of other sessions.

As with the strategy in [RK99] (and [PV08]), the DGS rewinding schedule is "adaptive". At a very high level, whenever a slot s completes, the simulator may rewind s by calling itself recursively on s. That is, the simulator chooses another challenge for s and recursively executes until either it receives the response (and hence "solves" the preamble) or it observes that the adversary has executed "too many" new slots in between or has aborted. By the time the simulator completes the preamble in any thread, our choice of the number of slots guarantees that there would exist at least one recursive level which will have at least $\frac{2n^2D^2}{p}$ slots of that preamble. Whenever the simulator observes $\frac{2n^2D^2}{p}$ slots in one level, it would rewind each of those slots exactly once and will try to solve that preamble. The formal description of our simulator rewinding strategy CEC-Sim is given below. Some of the text is borrowed from [DGS09].

- $d_{max} = \lceil \log_n(2k \cdot m) \rceil$ will denote the maximum depth of recursion. Note that d_{max} is a constant since the number of preambles $2k$ and number of slots per preamble m is polynomial in the security parameter n. It would be helpful to keep in mind that $2k \cdot m$ is the total number of slots at depth 0 (i.e., the main thread).
- slot(i, j) will denote slot j of preamble i. d denotes the current depth of the recursion.

SOLVE$(x, d, h_{initial}, s)$

Let $h \leftarrow h_{initial}$. Repeat forever and update h after each step:

1. If the adversary aborts or the number of slots in h started after $h_{initial}$ (which we will call new slots) exceed $\frac{2k \cdot m}{n^d}$, return h;
2. If the next message is a simulator challenge for the beginning of a slot s', choose the challenge randomly and send it to the adversary.
3. If the next message is the end message (adversary's reply) of a slot $s' = \text{slot}(i', j')$, proceed as follows:
   (a) If $s = s'$, we have succeeded in solving the target slot and hence the preamble. Return h;
   (b) Otherwise if the preamble i' has already been solved or the number of new slots (including s') of preamble i' in h started after $h_{initial}$ is less than $\frac{2n^2D^2}{p}$, the simulator need not rewind this slot. Go to the condition in Step 5;
   (c) Otherwise, we have an unsolved preamble i' such that $\frac{2n^2D^2}{p}$ of its slots (from slot$(i', j' + 1 - \frac{2n^2D^2}{p})$ to slot(i', j')) have appeared at the current level. The $\mathcal{S}$ will rewind each of these slots once and will try to solve preamble i'. Observe that the depth d_{max} of the recursion is a constant and the total number of slots in a preamble is $\frac{2n^3D^2}{p}$. This means just by the pigeonhole principle, for every preamble i', we would have this case at some level before the preamble is concluded. For each slot s" in this list of $\frac{2n^2D^2}{p}$ slots:
      i. Let h" be the prefix of h which contains all messages up to but excluding the simulator challenge for s". Set $h^* \leftarrow$ SOLVE$(x, d + 1, h", s")$.

ii. If h^* contains an accepting execution for slot s", the simulator has succeeded in solving s" and hence preamble i'.

4. If the next message is the last message of a preamble and the preamble has not been solved yet, the current set of look-ahead threads have failed. Abort and output Ext_Fail if we are in the main thread. If we are in a look-ahead thread, return.
5. If the next message is a message which belongs to the secure computation phase, the simulator strategy is given next (for main as well as look ahead threads). If the next message belongs to neither the DGS preamble nor the secure computation phase, the simulator executes in a straightline manner as described above (i.e., using the decommitment to B_i to complete WIAOK, playing honestly during coin flipping and while acting as the zero-knowledge verifier). Details are provided for the final construction in the appendix.

$\underline{CEC - Sim(x, z)}$

Run SOLVE($x, 0, \perp, \perp$) and output the view returned by SOLVE, with the following exception. When the simulator generates random challenge for a slot and it becomes equal to the another challenge generated previously in a different thread for the same slot, the simulator aborts and outputs $\perp$.

Simulator strategy in the secure computation phase. By the time $\mathcal{S}$ begins the secure computation phase in any session, it would have already extracted the input/randomness X_i to be used by the adversary in the semi-honest two-party protocol. It invokes the simulator of the protocol Π_{SH} who makes a call to the trusted functionality. Now we have two possible cases:

Case 1: $\mathcal{S}$ is currently executing the main thread. In this case, $\mathcal{S}$ goes ahead and queries the ideal functionality. It uses the received output to complete the secure computation phase.

Case 2: $\mathcal{S}$ is currently executing some look ahead thread. This is the more interesting case since $\mathcal{S}$ cannot simply query the ideal functionality. $\mathcal{S}$ would now use the predictor $\mathcal{P}$ guaranteed by the KTP of the ideal world. More precisely, $\mathcal{S}$ simply invokes $\mathcal{P}$ on x_i and all previously seen inputs/outputs in the current thread. Note that these previous outputs might have been answered by $\mathcal{S}$ either by making a call to the ideal functionality or using the predictor itself (in which case, correctness of the output is not guaranteed). Indeed assuming the adversary has full auxiliary information about the inputs of the honest parties, it can distinguish a correct output from an incorrect one (generated using the predictor). The core of the analysis of this prediction strategy (along with how it fits with our rewinding strategy) can be found in lemma 2.

3.3 Indistinguishability of the Outputs

We now consider a series of hybrid experiments and show that the views of $\mathcal{A}$ in successive hybrids are indistinguishable from each other.

Experiment H_0: The simulator $\mathcal{S}$ is given all the inputs of the honest parties. By running honest programs for the honest parties, it generates their outputs along with $\mathcal{A}$'s view. This is the execution in the real world in protocol Σ.

Experiment H_1: This experiment is exactly the same as the final simulated experiment except for the following. $\mathcal{S}$ still has all the honest party inputs and uses that to compute the outputs in the look ahead threads (instead of using the predictor).

The indistinguishability of the output distributions in H_0 and H_1 follows from standard techniques. By the property of the DGS preamble [DGS09] (also see appendix A.1), the simulator is successful in extracting the preamble secret β_i (which is a decommitment to B_i). In H_1, the simulator switches to using the witness β_i to complete all WIAOK executions. Here, the indistinguishability argument relies on the witness indistinguishability of the WIAOK system. Furthermore, the secure computation phase is completed by using the simulator of the Π_{SH} protocol. Here, the indistinguishability argument relies on the security of the Π_{SH} protocol. The formal proof is given in full detail in Appendix C for our final construction.

Experiment H_2: This experiment is exactly the same as the previous one except for the following. $\mathcal{S}$ starts using the predictor in the look ahead threads. However it still has all the honest party inputs. *$\mathcal{S}$ aborts the execution of a thread* as soon as the predictor returns an incorrect output value in the execution of that threads. That is, $\mathcal{S}$ returns the view so far in the current recursive call without continuing it any further thus returning the execution to the thread one level below.

We now prove the indistinguishability of the output distributions in H_1 and H_2. This is the core of our rewinding analysis. First observe that actually the main threads in H_1 and H_2 are identical conditioned on the event that the simulator does not fail to solve a preamble (and output Ext_Fail). We shall now prove that the probability of $\mathcal{S}$ outputting Ext_Fail in this experiment is negligible. This is also where we use the special properties of the DGS rewinding strategy and our key technical property.

Lemma 2 *The probability of the simulator outputting Ext_Fail in experiment H_2 is negligible.*

PROOF. The simulator (CEC-Sim routine) outputs Ext_Fail if it reaches the end of a preamble without extracting the corresponding preamble secret. Recall the step 3(c) in the CEC-Sim rewinding strategy. Since the depth d_{max} of the recursion is a constant and the total number of slots in a preamble is $\frac{2n^3D^2}{p}$, for every preamble i, there exists a depth d such that at least $\frac{2n^2D^2}{p}$ slots of the preamble i appear in a thread at depth d. Each of these $\frac{2n^2D^2}{p}$ is rewound exactly once. We would now prove that the simulator solves the preamble except with negligible probability as soon as this case happens.

Assume that the simulator output Ext_Fail with a noticeable probability. This means there should exists a preamble j and a depth d such that the following happens with a noticeable probability: at least $\frac{2n^2D^2}{p}$ slots of preamble j appear in a thread at depth d, each is rewound once but still the preamble secret of j is not extracted. Call these $\frac{2n^2D^2}{p}$ slots $s_1, \ldots, s_{\frac{2n^2D^2}{p}}$. Since each slot is rewound exactly once, this gives rise to $\frac{2n^2D^2}{p}$ look ahead threads. The preamble j is solved if the simulator receives the response to the challenge given in any of these look ahead threads. For this to not happen, each look ahead thread must have been aborted either because: (a) the predicator made an error in the look ahead thread, or, (b) the look ahead thread became "too long" or the adversary aborted in the thread on its own (see step 1 of the rewinding strategy). We will now analyze both events.

First we show that except with negligible probability, there are at least n^2 look ahead threads which were *not* aborted because of an error by $\mathcal{P}$. For each slot s_i, we define *calls belonging to it* as the calls made by $S_{\Pi_{\text{SH}}}$ while $\mathcal{S}$ is trying to compute a message between the beginning and the end of this slot. Note that these calls might have been answered by $\mathcal{S}$ either by making a call to the ideal functionality or by using the predictor (in which case, correctness of the output is guaranteed, else this thread would have been aborted; see description of $\mathcal{S}$ in this experiment). Denote the input and output tuple for calls belonging to s_i as IT_i and OT_i respectively. Also define a set S_i containing the ideal world session indices corresponding to the calls belonging to s_i. Similarly, define IT'_i and OT'_i as the calls made by $S_{\Pi_{\text{SH}}}$ while $\mathcal{S}$ is executing the look ahead thread corresponding to the slot s_i. Also define S'_i in an analogous way.

Now we define $2n^2D/p$ *blocks* of slots with the a-th block containing slots from s_{aD+1} to $s_{(a+1)D}$. By the KTP guarantee, for the collection of sets S_{aD+1} to $S_{(a+1)D}$ (corresponding to block a), the predictor can be used to predict the output tuples corresponding to at least one set with probability p. However, we would be interested in predicting the output tuples for a set S'_i as opposed to S_i (to not abort in the look ahead thread).

For a block a, choose a random index $b \in [D]$ and look at the predictor execution for S'_{aD+b}.

- We claim that the predictor does not make an error for any of the calls corresponding to the set S'_{aD+b} with probability at least p/D. This is because the KTP guarantees that the predictor would successfully work for at least one of the sets S_{aD+1} to $S_{(a+1)D}$ with probability at least p. Hence, it would work for set S_{aD+b} at least with probability p/D (given all the input/output tuples corresponding to the calls in this thread so far). However the distribution of S'_{aD+b} is *identical* to S_{aD+b} given the entire thread up to the point where slot s_{aD+b} is supposed to begin (in other words, one is in fact interchangeable with the other).

- Hence, predictor is successful in set S'_{aD+b} and hence in the look ahead thread for at least one slot in block a with probability p/D. Since there are a total of $2n^2D/p$ blocks, the expected number of look ahead threads where the predictor is successful is $2n^2$.

- This also implies that, except with negligible probability, there are *at least* n^2 look ahead threads which were not aborted because of an error by the predictor $\mathcal{P}$ (by the multiplicative form of Chernoff bounds).

Now this must mean that there exists at least n^2 slots in a thread at depth d such that each was rewound once but the corresponding look ahead thread was aborted because it either had too many new slots or the adversary aborted on its own. Before the we analyze the probability of this event, we first define a $(d+1)$**-good** slot. A slot at depth d is called $(d+1)$-good if:

(a) it has a maximum of $\frac{2k \cdot 2n^3 D^2}{pn^{d+1}}$ new slots between its start and finish messages, and,

(b) it is such that the adversary did not abort the thread before its finish message

Observe that at most n slots out of the $\frac{2n^2 D^2}{p}$ slots at depth d (being rewound) may *not* be $(d+1)$-good. This is because all of them obviously satisfy the condition (b) above (i.e., them all finished before the thread was aborted; this is why they are getting rewound). Plus, if more than n of them have $\frac{2k \cdot 2n^3 D^2}{pn^{d+1}}$ new slots in between their start and finish messages, the total number of new slots at level d will exceed $\frac{2k \cdot 2n^3 D^2}{pn^d}$ (which is impossible; see step 1 of our rewinding strategy).

Now to analyze the probability of event $\mathsf{Ext\_Fail}$, we consider the following experiment. Consider the point (in the thread at depth d) where the first of these $\frac{2n^2 D^2}{p}$ slot begins. The simulator chooses two random tapes and creates two threads (one using each tape) forking off from this point. The simulator uses the predictor to predict the outputs in both the threads. If the predictor fails to correctly predict the outputs in both of them, the simulator proceeds by randomly choosing one of the random tapes to construct the current thread at depth d and the other to construct the corresponding look-ahead thread for this slot. However, in case the the predictor is successful in predicting the output in at least one of threads, the simulator queries an external party B, gets a random bit and then accordingly chooses one of the random tapes to construct the current thread and the other to construct the look-ahead thread. (The strategy followed to construct the current thread and the look-ahead remains the same as in hybrid H_2; we only change the way simulator chooses its random tape for running different parts of the simulation.) Clearly, the output of the simulator in this experiment remains identical to that in hybrid H_2.

As we have shown before, for at least n^2 of these $\frac{2n^2 D^2}{p}$ slots, the predictor correctly predicts the output (in at least one of the two threads). Now we consider two disjoint events:

- At least $\frac{n^2}{2}$ of the n^2 threads (for which the predictor gives correct output) are $(d+1)$-good. Conditioned on this event, clearly the probability of the simulator outputting $\mathsf{Ext\_Fail}$ is $2^{-\frac{n^2}{2}}$. This is because even if one of these $\frac{n^2}{2}$ threads (for which the predictor gives the correct output and $(d+1)$-good property holds) is chosen to be a look-ahead by the external party B, our simulation will be successful.
- At least $\frac{n^2}{2}$ of these n^2 threads are *not* $(d+1)$-good. Conditioned on this event, clearly, the expected number of slots which are not $(d+1)$-good in the current thread at depth d is at least $\frac{n^2}{4}$ (since each is chosen to be a part of the current thread with probability $\frac{1}{2}$). By Chernoff bounds, except with negligible probability conditioned on this event, the current thread at depth d will have at least $\frac{n^2}{4}$ slots which are not $(d+1)$-good. However we have already shown that at most n slots out of the $\frac{2n^2 D^2}{p}$ slots at depth d (being rewound) may not be $(d+1)$-good.

This is in contradiction to the fact that the probability of $\mathcal{S}$ outputting $\mathsf{Ext\_Fail}$ is noticeable.

Experiment H_3: This experiment corresponds to the final simulated experiment. That is, $\mathcal{S}$ no longer has the honest party inputs and hence does not abort the look ahead threads where the predictor makes an error. Observe that in this hybrid, the probability of $\mathcal{S}$ outputting $\mathsf{Ext\_Fail}$ can only go down compared to that in H_2. Hence, indistinguishability of the output distributions in H_2 and H_3 immediately follows thus completing the proof.

Running Time of $\mathcal{S}$. To bound the number of queries $\mathcal{S}$ makes to $\mathcal{A}$, we consider the recursive execution tree (of constant depth) resulting out of $\mathcal{S}$ rewinding $\mathcal{A}$. Each call to the function SOLVE$(\cdot,\cdot,\cdot,\cdot)$ will represent one node in the execution tree. The nodes resulting from all further recursive calls to SOLVE will be treated as children of this node. Thus, the root node (at depth 0) is the call SOLVE$(x, 0, \cdot, \cdot)$ made by $CEC - Sim(x, z)$. This call results in the main thread while recursive calls give rise to the look ahead threads.

Now consider the transcript generated by a function call representing a node at depth d (excluding the transcripts generated by any further recursive calls). The number of new slots in this transcript is bounded by $2k \cdot \frac{2n^3D^2}{p}$ (in fact $\frac{2k \cdot 2n^3D^2}{pn^d}$). Now, each of these slots may have up to one look ahead thread resulting in a total of up to $4k\frac{2n^3D^2}{p}$ children for this node. Hence, the execution tree is a tree of depth up to d_{max} and degree up to $4k\frac{2n^3D^2}{p}$. Hence, the total number of nodes is bounded by $(4k\frac{2n^3D^2}{p})^{d_{max}+1}$. The transcript of each node contains up to $O(4k\frac{2n^3D^2}{p})$ queries. Hence, the total number of queries $\mathcal{S}$ makes to $\mathcal{A}$ is $O((4k\frac{2n^3D^2}{p})^{d_{max}+2})$ which is a polynomial (since d_{max} is a constant). Also, its easy to see that each query to $\mathcal{A}$ takes only PPT assuming $\mathcal{A}$ is a PPT machine. This concludes our analysis.

Final Construction. The above construction only provides concurrent security for one party. This is because in case P_1 is corrupted and interacts with *multiple* honest parties, its input in one session may somehow be dependent upon the input of an honest parties in another session. To overcome this problem, we use techniques from concurrent non-malleable zero-knowledge protocols [BPS06]. Our final construction (relying on BPS concurrent non-malleable ZK) is given in full detail in appendix B.

4 Generalization to Bounded-Size Hardness-Free Ideal World

We first define a stateful version of the notion of W-UPF given in section 2.1.

Definition 5 (General Worst-Case Hard Unpredictable Functionalities) *A general worst-case hard unpredictable functionality $\mathcal{G}$ works as follows. It is initialized by giving an input $x \in \{0,1\}^n$ and can be queried on a string r. The output of the functionality $\mathcal{G}$ can be any polynomial time computable function of (x, r) and all previous queried made (i.e., its state). Further, we require that for all uniform probabilistic polynomial time algorithms A, there exists x, f and at least $100nf$ inputs $r_1, \ldots, r_{100nf}$ such that the following holds. The functionality $\mathcal{G}$ is queried on these strings in the* order *they appear (i.e., r_i is queried before r_{i+1}). There exists a PPT distinguisher $\mathcal{D}$ such that for all $i \in [100nf]$,*

$$Pr\big[A(r_1, \ldots, r_i, \mathcal{G}(x, r_1), \ldots, \mathcal{G}(x, r_{i-1})) \neq \mathcal{G}(x, r_i)\big] \geq \frac{1}{f}$$

It is easy to see that for any general W-UPF functionality $\mathcal{G}$, there exists a (stateless) W-UPF g.[5]. Hence, it can be shown that any general W-UPF functionality implies the existence of W-OWF.

Bounded-size hardness-free ideal world. We now discuss a natural extension of our previous result regarding the single input setting. We consider the very general setting where the honest parties may have different (possibly adaptively chosen) inputs in different sessions, there may be multiple parties participating in a protocol session with potentially all of them getting different outputs, the adversary may choose to corrupt parties with different roles in different sessions (i.e., the interchangeable role setting [Lin08]), etc. Our only requirement is that, informally speaking, the size of the total "state" of the honest parties in the ideal world be bounded and the ideal world be "hardness-free". More formal details follow.

We only discuss the changes from the model in section 2.2. In the ideal world, there are several parties $P_1, P_2, P_3, \ldots$. In addition, we have a deterministic and non-reactive trusted functionality $\mathcal{F}$. Without loss of generality, let t denote the input and the output length for this functionality. There could be any unbounded (polynomial) number of parties in the ideal world with each party participating in any unbounded (polynomial) number of sessions.

- At the beginning of the ideal experiment, each of the parties has some state (which may include some initial input and a random tape).

[5]Hint: a tuple $(r_1, r_2, \ldots, r_{poly(n)})$ for $\mathcal{G}$ would translate to $(r_1, r_1||r_2, \ldots, r1|| \ldots ||r_{poly(n)})$ for $\mathcal{G}'$

- For a given session, any honest party P_i may choose an input $x_i \in \{0,1\}^t$ adaptively depending upon its initial state and the view in the ideal world so far.
- The adversary may corrupt any subset of parties at the beginning of the ideal world execution. If a party is corrupted, it is free to behave in any way it wants (in particular, by starting any number of sessions and adaptively choosing an input in each session depending upon the entire view of the adversary so far).
- The parties send their input to the trusted functionality $\mathcal{F}$.
- The functionality $\mathcal{F}$ computes the output and sends it to the adversarial parties first. Finally, if a signal is received from any adversarial party, $\mathcal{F}$ sends the output to the honest parties as well. (See section 2.2 for more details).
- Each session in the ideal world is assigned a session index in the order of execution (i.e., session ℓ is completed before session $\ell + 1$).

We now define a bounded size ideal world.

Definition 6 (Bounded-size ideal world) *An ideal world is a bounded-size ideal world if for all input lengths t and all possible corrupt subset of parties, there exists a string $S \in \{0,1\}^{poly(t)}$ such that the following holds. The string S fully describes the initial state of each honest party in the ideal world. Given S, it is possible to perfectly emulate the actions of each honest party in the ideal world in (deterministic) polynomial time.*

Assuming that the total number of honest parties in bounded (with each party potentially participating in any unbounded polynomial number of sessions) and a bound on the size of the initial state of each party is known, this condition is automatically satisfied.

We now discuss what we mean by a hardness-free ideal world. We first define a machine $\mathcal{M}$ which encapsulates all the honest party machines and the ideal functionality. The adversary may query this machine in any session (as it would query the ideal functionality) and get the appropriate output tuple.

Definition 7 (Hardness-free ideal world) *An ideal world is a hardness-free ideal world if the following does* not *hold. There exists an input length t, a corrupted subsets of parties and a tuple of starting state for the honest parties such that the corresponding machine $\mathcal{M}$ is a general W-UPF functionality (as per definition 5).*

Note that the fact that an ideal world is bounded-size and hardness-free only imposes restrictions on the honest parties and the ideal functionality. *There are no restrictions on the ideal adversary* (which in particular could perform any cryptographic operations and start with an input state with length potentially dependent upon the real protocol Σ designed to realize this ideal world). Further, both the restriction are necessary as follows from our impossibility result in Section 5 and the known impossibility results on bit transmitting functionalities [Lin08].

We now define a generalization of the key technical property.

Definition 8 (Generalized Key Technical Property) *The key technical property of an ideal world experiment requires the existence of a PPT predictor $\mathcal{P}$ satisfying the following conditions. There exists (D, c) such that for all adversaries (and the corresponding subset of corrupted parties), all honest parties (and their starting states), all $\{S_D\}_D$ as above and all sufficiently large n, there exists i such that for all PPT distinguishers $\mathcal{D}$ the following condition is satisfied.*

$$Pr\big[\forall j \in S_i, \mathcal{P}\big(\{I[\ell]\}_{\ell \leq j}, \{O[\ell]\}_{\ell < j}\big) = O[j]\big] \geq n^{-c}$$

where the probability is taken over the random coins of the predictor. Denote the lower bound on the prediction probability (i.e., n^{-c}) by p.

The following lemma easily follows from lemma 1.

Lemma 3 *If a bounded-size ideal world does not satisfy the generalized key technical property, then the corresponding machine $\mathcal{M}$ described above must be a general W-UPF functionality (definition 5).*

We now have the following result.

Theorem 1 *Assuming that the ideal world for the functionality $\mathcal{F}$ is bounded-size and hardness-free, there exists a real world protocol Σ which securely realizes the ideal world for the functionality $\mathcal{F}$ (under suitable cryptographic assumptions).*

The proof of this theorem is essentially the same as for the case when the ideal world is single input (see section B). The proof of security for the protocol Σ does not use the fact that the ideal world is a single-input ideal world and rather only makes use of the fact that the ideal world satisfies the key technical property (see theorem 3). Even if the above generalized KTP is satisfied in the ideal world, the existence of a predictor required for lemma 2 is still guaranteed. Hence the entire proof of security remains essentially unchanged.

5 Impossibility Result for a Single Input PRF Functionality

Let COM denote a non-interactive statistically binding commitment scheme (see appendix A) and $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ denote a keyed pseudorandom function. We first define a "committed input" PRF functionality F^{σ}_{PRF} as follows. The functionality F^{σ}_{PRF} takes an input k, r from P_1 and x from P_2. It first checks if $\sigma = COM(k, r)$, if not, it outputs $\perp$ to P_2. Else, it outputs $f(k, x)$ to P_2. We will prove the following theorem in this section.

Theorem 2 *There does not exist a protocol Π^{σ}_{PRF} for the functionality F^{σ}_{PRF} as per the security definition 3 for the single input setting.*

Towards the contrapositive, assume any secure protocol Π^{σ}_{PRF} for the functionality F^{σ}_{PRF}. Our impossibility result proceeds by providing an explicit attack on the protocol Π^{σ}_{PRF}. Our proof goes in two stages. We first provide an intuitive high level explanation of the two stages. Full details follow afterwards.

In the first stage, we construct a simple protocol $m\Pi^{\sigma}_{PRF}$ such that Π^{σ}_{PRF} and $m\Pi^{\sigma}_{PRF}$ are "insecure" w.r.t. each other. In more detail, consider two honest parties P_1 and P_2 and an adversary $\mathcal{A}$ acting as a man in the middle between them. P_1 and $\mathcal{A}$ execute the protocol Π^{σ}_{PRF} while $\mathcal{A}$ and P_2 execute the protocol $m\Pi^{\sigma}_{PRF}$. The protocol $m\Pi^{\sigma}_{PRF}$ is such that it simply asks the party P_2 to execute the protocol Π^{σ}_{PRF} with $\mathcal{A}$. Upon successful completion of the protocol Π^{σ}_{PRF}, P_2 in $m\Pi^{\sigma}_{PRF}$ is further instructed to send a secret (random) string secret to $\mathcal{A}$. Thus, $m\Pi^{\sigma}_{PRF}$ is such that the adversary $\mathcal{A}$ can simply relay messages between P_1 and P_2. Upon successful completion of the protocol $m\Pi^{\sigma}_{PRF}$, $\mathcal{A}$ learns secret. However, the protocols are such that if the protocol Π^{σ}_{PRF} is replaced by its corresponding ideal version, it is infeasible for $\mathcal{A}$ to complete $m\Pi^{\sigma}_{PRF}$ in this hybrid world and obtain secret from P_2 .

Indeed, the above does not (yet) contradict the security of the protocol Π^{σ}_{PRF}. This is because the security guaranteed holds only when copies of Π^{σ}_{PRF} are composed with each other as opposed to with arbitrary protocol such as $m\Pi^{\sigma}_{PRF}$ (and that too, only when an honest P_1 uses the same input in all copies of Π^{σ}_{PRF}). In the second stage, we replace the party P_2 in the above scenario by a set of *Yao's Garbled Circuits* [Yao86] given to the adversary as an auxiliary input. More precisely, assuming that the protocol $m\Pi^{\sigma}_{PRF}$ has m rounds (where each round is defined by a message from $\mathcal{A}$ followed by a reply from P_2), we construct m garbled circuits where the i-th garbled circuit $GC[i]$ implements the next message function of P_2 in protocol $m\Pi^{\sigma}_{PRF}$ round i. The garbled circuit $GC[i]$ takes as input the message from $\mathcal{A}$ in round i (along with an authenticated encryption of the previous state). It outputs the message of P_2 in round i (along with an authenticated encryption of the updated state). These set of garbled circuits are given to $\mathcal{A}$ as an auxiliary input and $\mathcal{A}$ would now "execute the protocol $m\Pi^{\sigma}_{PRF}$ with these circuits" (in place of P_2).

The remaining issue is that to execute these garbled circuits, the adversary would need to know the appropriate input wire keys. A similar issue arose in the lower bound of Barak et al [BPS06] where they implemented the required oblivious transfers as follows. All the wire keys are given to the (honest party) P_1 as part of its input. P_1 and $\mathcal{A}$ now run several copies of the basic protocol in question concurrently (as opposed to just one). The functionality in

[BPS06] supports an oblivious transfer (OT) mode where P_1 uses the appropriate pair of wire keys as input and $\mathcal{A}$ can get exactly one of the keys. Thus, $\mathcal{A}$ is able to interact with P_1 concurrently and extract secret from the set of garbled circuits. However in the ideal world, it would be infeasible for the adversary to recover secret similar to the first stage. Thus, Barak et al [BPS06] were able to show insecurity of the protocol they started with.

However, this issue presents a bigger problem in our single input setting. Indeed in our setting, it is easy to transfer an input wire key to $\mathcal{A}$ as an output of the PRF functionality on a specific predetermined input. But its hard to enforce that the adversary gets only one wire key per input wire for each of the garbled circuits. The honest party uses a fixed input in all sessions and the adversary is free to choose its input in any way in these sessions. The only restriction maybe on the total number of sessions that the adversary may participate in. Hence, $\mathcal{A}$ may run the protocol Π^{σ}_{PRF} on two different inputs such that it receives the two different wire keys for a single input wire. Once that happens, all bets about the security of the garbled circuit are off.

To solve this problem, we first propose a new garbled circuit construction where the garbled circuit is executed by the receiver by a single k-out-of-2k OT (as opposed to k execution of 1-out-of-2 OT). Thus, a malicious receiver can potentially choose to get both wire keys for an input wire (as long as the total number of keys he gets is bounded by k). We believe our construction is of independent interest since, to our knowledge, all previous constructions of protocols based on garbled circuit involved parties executing k (1-out-of-2) OTs, while, executing a single k-out-of-2k OT may, e.g., be more efficient[6]. In more detail, we actually provide a construction of *one time programs* [GKR08] based on a *single k-time-memory hardware token*. More details about the notion of one time programs follow later. Note that one time programs refer to the setting where the adversary maybe given access to the garbled circuit *prior* to deciding its input and getting the wire keys. This is closer to our setting (since $\mathcal{A}$ gets the garbled circuits as part of the auxiliary input before the protocol starts) and hence working with one time programs will be more convenient to us than working directly with garbled circuits.

5.1 One Time Programs with a Single k-time-memory Token

5.1.1 The Model

Informally, a *one-time program* (OTP) [GKR08] for a function f lets a party evaluate f on only one input chosen by that party at run time. The intuitive security goal is that no efficient adversary, after evaluating the one-time program on x, can learn anything about $f(y)$ for some $y \neq x$, other than what can be inferred from $f(x)$. Figure 1 defines the ideal functionality for a one-time program for function $f(\cdot)$.

Functionality $\mathcal{F}^{OTP}$

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, f)$ from P_i, where f is the description of a function, do:

1. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ to P_j.
2. Store (P_i, P_j, f).

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, x)$ from P_j, find the stored tuple (P_i, P_j, f) (if no such tuple exists, do nothing). Send $f(x)$ to P_j and delete tuple (P_i, P_j, f).

Figure 1: Ideal functionality for One-time Program for function $f(\cdot)$.

In the real world, a OTP is a "software-hardware" package, i.e., consists of a number of tamper proof hardware tokens along with a sequence of bits. One way of implementing OTPs using hardware tokens is to construct stateful

[6]It has been argued that the key efficiency bottleneck in secure two-party computation based on garbled circuits is the execution of oblivious transfers since the rest just involves symmetric key operations (see, e.g., [PSW09]).

hardware tokens that contain the entire code of the function $f(\cdot)$. However, the emphasis in [GKR08] is to use only the simplest kind of hardware tokens. To this end, they focus on using one-time-memory (OTM) tokens only. An OTM token consists of two string out of which the receiver can read any one string of its choice; the token "self-destructs" afterwards[7]. One can generalize this concept to use tokens which the receiver is allowed to use a fixed k number of times called k-time-memory (kTM). We define the ideal functionality realized by a kTM token in Figure 2.

Functionality kTM

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \dots))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \dots), 0)$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, check if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \dots), i)$ exists. If not, do nothing. Else, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j. Now check if $k = i + 1$. If so, delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \dots), i)$. Else, overwrite the tuple with $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \dots), i + 1)$

Figure 2: Ideal functionality for kTM tokens

Now we define OTPs in the kTM-hybrid model.

Definition 9 *(One-Time Program for $f(\cdot)$) A one-time program for function $f(\cdot)$ is a two-party non-interactive protocol $\Pi = (P_1, P_2)$ in the kTM-hybrid model, such that for every probabilistic polynomial time adversary $\mathcal{A}$ corrupting P_2 , there exists a probabilistic polynomial time ideal-world adversary S called the simulator, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}_{S,\mathcal{Z}}^{\mathcal{F}^{OTP}} \sim \mathsf{REAL}_{\Pi,\mathcal{A},Z}.$$

5.1.2 Our Construction

Let the sender P_1 have a function $f : \{0,1\}^s \to \{0,1\}^t$ and the receiver P_2 has an input $x \in \{0,1\}^s$. The (non-interactive) protocol Π proceeds as follows.

Construction of the One Time Program. The sender does the following:

- Without loss of generality, let t be a multiple of n (where n is the security parameter). Let r be such that $t = rn$. P_1 generates rs random strings $\{a_i^j\}_{i\in[s],j\in[r]}$ s.t. $a_i^j \in \{0,1\}^n$. Similarly generate another set of $2rs$ random strings $\{k_i^{j,b}\}_{i\in[s],j\in[r],b\in\{0,1\}}$ s.t. $k_i^{j,b} \in \{0,1\}^n$.
- Set $a_i = a_i^1||\dots||a_i^r$ and $a = a_1 \oplus \dots \oplus a_s$. Note that $a \in \{0,1\}^{rn}$. Define a modified functionality $g(x) = f(x) \oplus a$.[8] The sender P_1 would construct a garbled circuit for the modified functionality g as opposed to for f. This step is to ensure that unless an adversarial receiver has received at least one string a_i^j for every wire i in the circuit, it gets no information about the output (in a computationally sense).
- Set $k_i^b = k_i^{1,b} \oplus \dots \oplus k_i^{r,b}$. k_i^b will be the wire key corresponding to bit b for the i-th input wire of the garbled circuit (where $i \in [s]$).
- Let C_g represents a circuit to compute the function g as discussed above[9]. The sender P_1 now prepares a modified circuit C_g^{mingle} as follows. Each input wire $i \in [s]$ of the circuit C_g is first subjected to a "dummy" operation with every other input wire in C_g such that the output of the operation is the original value on the wire i itself. This is done by having $s - 1$ dummy binary gates for each input wire i where: (a) the input to the

[7]thus, in some sense, an OTM token implements a form of oblivious transfer

[8]Jumping ahead to the proof of security, such a concatenation is only required to ensure that the input strings in the kTM token are of size only dependent on the security parameter n and not on the size of the output of the one time program. This property is crucial for our impossibility result in this section.

[9]C_g needs to be an "oblivious" circuit for g as in [GKR08], i.e., the topology of C should not leak any information about the function g.

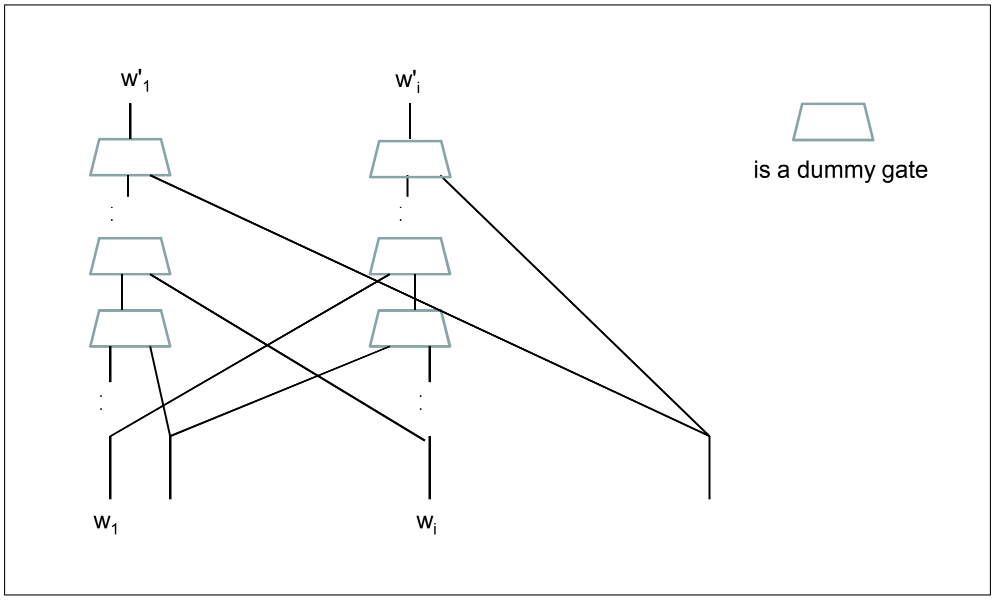

Figure 3: Intermingling of the Input Wires in a Circuit C_g

first dummy gate is the wire i and the first of the remaining $s-1$ input wires, the output is the value on wire i itself, and, (b) the input to the j-th dummy gate (where $j > 1$) takes the output wire of the $(j-1)$-th dummy gate and the j-th of the remaining $s-1$ input wires. After this "input wire intermingling" step, we having s resulting wires which are then taken as input wires for the circuit C_g. This completes the construction of the circuit C_g^{mingle}. A pictorial representation of the input wire intermingle step is given in figure 3. Intuitively, this step ensures the following key security requirement. Say that an adversarial receiver gets both wire keys for some input wires but none for at least one input wire. Then, it would be unable to proceed and get any useful information from the garbled circuit for C_g^{mingle} (that we construct next).

- The sender P_1 now constructs a garbled circuit for the circuit C_g^{mingle} in the standard way. That is, for any (possibly dummy) gate in the circuit, given a wire keys for each of the two input wires of this gate, the receiver should be able to compute a wire key for the output wire (where which output wire key the receiver gets is determined by the functionality of the gate). However if the receiver does not have any wire key for at least one of input wires of this gate, the output wire keys remain semantically secure. We review a technique for garbled circuit construction from [LP09] in the Appendix E. Let the resulting garbled circuit for C_g^{mingle} be denoted by GC.

- Define $2rs$ strings $\{K_i^{j,b}\}_{i\in[s],j\in[r],b\in\{0,1\}}$ such that $K_i^{j,b} = k_i^{j,b}||a_i^j$. The sender P_1 now prepares a rs-time-memory hardware token $\mathcal{T}$ by putting in these $2rs$ strings inside $\mathcal{T}$ as its input.

- The one time program OTP consists of the garbled circuit GC and rs-time-memory token $\mathcal{T}$. The sender P_1 sends the one time program OTP to the receiver P_2 .

Evaluation of the One Time Program. The receiver P_2 runs the received OTP consisting of GC and $\mathcal{T}$ on the input $x \in zos$ as follows. P_2 queries the token $\mathcal{T}$ rs times to get the strings $\{K_i^{j,x_i}\}_{i\in[s],j\in[r]}$ where x_i represents the i-th bit of x. In other words, it gets strings $\{k_i^{j,x_i}\}_{i\in[s],j\in[r]}$ and $\{a_i^{j,x_i}\}_{i\in[s],j\in[r]}$. It then evaluates the garbled circuit GC in the natural way. First compute the appropriate wire keys for the input wires $k_i^{x_i} = k_i^{1,x_i} \oplus \cdots \oplus k_i^{r,x_i}$ and evaluate the garbled circuit to recover the output $g(x) = f(x) \oplus a$. Set $a_i = a_i^1||\ldots||a_i^r$ and $a = a_1 \oplus \cdots \oplus a_s$. Finally, recover $f(x) = g(x) \oplus a$. Output $f(x)$.

Proof of Security. We now prove that the above construction indeed is secure as per the definition 9. Note that the definition only requires security against a malicious receiver.[10] We only give a sketch of the proof since the details following using standard ideas. Our simulator $\mathcal{S}$ works as follows. It simply constructs a one time program honestly for any arbitrary function f' of similar size as f and sends it to the adversary $\mathcal{A}$. We consider three separate cases regarding how the adversary queries the token $\mathcal{T}$

- There exists an input wire i corresponding to which the adversary did not query for exactly r strings. That is, there exists an i such that the strings of form $K_i^{\cdot,\cdot}$ received by the adversary are not exactly r. That means there must exists an i such that the strings of form $K_i^{\cdot,\cdot}$ received by the adversary is *less* than r (this follows by the total bound rs on the number of queries). Thus, both the strings $\{k_i^b\}_{b\in\{0,1\}}$ are semantically secure to the adversary. Then by a simple examination of our input intermingling stage, it follows that all the wire keys for the input wires to the circuit C_g (which is a part of the circuit C_g^{mingle}) are actually semantically secure. Then by the security of the garbled circuit construction in Appendix E, it follows that the adversary fails to learn any useful information from the garbled circuit. Thus, the simulator can play honestly for rest of the game (even with the garbled circuit for the incorrect function f') and still the view of the adversary will be indistinguishable from that in the real world.

- The adversary queries for exactly r strings corresponding to each input wire i. However there exists an input wire for which the adversary did not query consistently for a single input bit x_i. That is, adversary received two keys $K_i^{\cdot,p}$ and $K_i^{\cdot,q}$ s.t. $p \neq q$. Then it follows that both the strings $\{k_i^b\}_{b\in\{0,1\}}$ are semantically secure to the adversary. The rest of the argument is identical to the first case.

- The adversary queried the token $\mathcal{T}$ honestly for a fixed input x of its choice. In this case, before the adversary learns any information about the string a, the simulator can recover the entire input string x by looking at the queries the adversary makes to $\mathcal{T}$. Thus, very similar to the proofs in [GKR08, GIS$^+$10], the simulator can query the ideal functionality to get the output $f(x)$ and then is able to set the remainder of the strings a_i^j such that the output of the garbled circuit $g(x)$ XOR'ed with a gives the correct output $f(x)$.

5.2 The Impossibility Result

We now move on to the first stage of our proof of impossibility of a protocol Π_{PRF}^{σ} realizing the functionality F_{PRF}^{σ} as described in the beginning of this section. In the first stage, we first construct a simple protocol $m\Pi_{PRF}^{\sigma}$ such that Π_{PRF}^{σ} and $m\Pi_{PRF}^{\sigma}$ are "insecure" w.r.t. each other.

Consider three parties P_1, $\mathcal{A}$ and P_2. The party P_1 holds the secret input k, r (s.t. $\sigma = COM(k, r)$). P_1 and $\mathcal{A}$ execute the protocol Π_{PRF}^{σ} where $\mathcal{A}$ has the opportunity to choose an input and get the corresponding PRF output. $\mathcal{A}$ and P_2 execute a protocol $m\Pi_{PRF}^{\sigma}$ which works as follows. The party P_2 has an secret (random) input x with which it executes the protocol Π_{PRF}^{σ} with $\mathcal{A}$. Upon successful completion of the protocol Π_{PRF}^{σ}, P_2 in $m\Pi_{PRF}^{\sigma}$ is further instructed to send a secret (random) string secret to $\mathcal{A}$.[11] Now consider the case when the party $\mathcal{A}$ is adversarial. It is easy to see that it can recover the secret secret by simply relaying messages between P_1 and P_2 where in the final round, P_2 would send the string secret to $\mathcal{A}$.

We now argue that if the protocol Π_{PRF}^{σ} is replaced by its corresponding ideal version, it is infeasible for any adversary $\mathcal{S}$ to complete $m\Pi_{PRF}^{\sigma}$ in this hybrid world and obtain secret from P_2 . This holds even if $\mathcal{S}$ is allowed any polynomial number of queries to the functionality F_{PRF}^{σ} (as opposed to just one). We look at the call(s) made by $\mathcal{S}$ to the ideal functionality F_{PRF}^{σ}. We consider three different cases:

- The adversary $\mathcal{S}$ queries the ideal functionality F_{PRF}^{σ} with the (correct) input x as chosen by P_2 . It is easy to show that this violates the (standalone) security of the protocol Π_{PRF}^{σ}. This is because the protocol $m\Pi_{PRF}^{\sigma}$ simply consists of running Π_{PRF}^{σ} and then sending secret after successful completion.

[10]Goyal et al [GIS$^+$10] recently proposed extensions to consider security against a malicious *sender* as well. However in the current work, the original security notion of Goldwasser et al [GKR08] is sufficient.

[11]If the protocol Π_{PRF}^{σ} involves the use of identities of the participants, $m\Pi_{PRF}^{\sigma}$ instructs $\mathcal{A}$ and P_2 to use identities P_1 and $\mathcal{A}$ respectively.

- The adversary $\mathcal{S}$ does not query the ideal functionality F^{σ}_{PRF} with x but nonetheless, at the end of Π^{σ}_{PRF} (run as part of $m\Pi^{\sigma}_{PRF}$), P_2 gets the correct output $f(k, x)$. This violates the security of the pseudorandom function f.

- Finally, the adversary $\mathcal{S}$ does not query the ideal functionality F^{σ}_{PRF} with x and at the end of Π^{σ}_{PRF} (run as part of $m\Pi^{\sigma}_{PRF}$), P_2 receives an incorrect output. Again, it is easy to show that this violates the (standalone) security of the protocol Π^{σ}_{PRF}.

Now we move on to the second stage of our proof where our goal would to be replace the party P_2 (and the protocol $m\Pi^{\sigma}_{PRF}$ between $\mathcal{A}$ and P_2) by m one time programs given to $\mathcal{A}$ as an auxiliary input. We would implement the rs-time-memory tokens required as part of these OTPs by letting the required strings (that would been put inside these tokens) be the output of the functionality F^{σ}_{PRF} on certain inputs. We now allow $\mathcal{A}$ to interact with P_1 in protocol Π^{σ}_{PRF} concurrently in several additional sessions to enable it to obtain the strings required to execute the one time programs.

Trying to implement the above idea in the naive way leads the following problem. The adversary now is not restricted to making only rs queries for a OTP (but rather has a total bound on the number of queries). Hence, for some of the one time programs, the adversary might make more than rs queries in which case, all bets are off regarding the security of those one time programs and the adversary might learn secret even in the ideal world.

Fixing this problem is easy. To recover secret, we force the adversary to query each of the m one time programs as follows. The ℓ-th one time program gives out a string $\mathsf{secret}^{\ell} \in \{0,1\}^n$ such that $\mathsf{secret} = \mathsf{secret}^1 || \dots || \mathsf{secret}^m$. Now if adversary uses up more than rs queries for a single OTP, it is guaranteed that there would be at least one OTP such that the adversary fails to learn any information about its output. Hence, it fails to recover secret. More details follow.

We modify the protocol $m\Pi^{\sigma}_{PRF}$ described in the first stage slightly such that in round ℓ, in addition to the next message for protocol Π^{σ}_{PRF}, it gives out a string secret^{ℓ}. Without loss of generality, we assume that the first message in protocol $m\Pi^{\sigma}_{PRF}$ is sent by $\mathcal{A}$ and consequently in Π^{σ}_{PRF} by P_1 . Assume that the protocol $m\Pi^{\sigma}_{PRF}$ has m rounds (where each round is defined by a message from $\mathcal{A}$ followed by a reply from P_2). Let the next message function of P_2 in protocol $m\Pi^{\sigma}_{PRF}$be denoted by F_{NMF}. We now construct m one time programs as follows. In the following, when we talk about a one time program, we follow the same notations as in the previous subsection but add a postfix $[\ell]$ to the variables to denote that we are referring to $OTP[\ell]$.

- We first specify the functionality of these m OTPs. $OTP[1]$ takes as input the first message from $\mathcal{A}$ and outputs the resulting message by applying F_{NMF}. In addition, it output secret^1 and S_1 where S_1 is state (in the protocol $m\Pi^{\sigma}_{PRF}$ after the first round) encrypted using an authenticated encryption scheme (whose secret key is shared by all one time programs). The OTP $OTP[\ell]$ for $\ell > 1$ takes as input the message from $\mathcal{A}$ in round ℓ and $S_{\ell-1}$. It outputs the resulting message by applying F_{NMF} in addition to secret^{ℓ} and S_{ℓ}.[12]

- We would implement the rs-time-memory tokens required as part of these OTPs by means of the F^{σ}_{PRF} functionality as follows. We set the string $K^{j,b}_i[\ell]$ to $f(k, j||b||i||\ell)$ (where $||$ denotes concatenation); we require the output of the functionality F^{σ}_{PRF} to be a $2n$-bit string.

- The m garbled circuits corresponding to such m one time programs are constructed and given to $\mathcal{A}$ as an auxiliary input.

- $\mathcal{A}$ is allowed to interact with P_1 concurrently in the protocol Π^{σ}_{PRF} in $mrs + 1$ sessions (as opposed to running just a single sessions).

- $\mathcal{A}$ now starts the first session of $\Pi^{\sigma}_{PRF}$with P_1 . It simply passes the message received from P_1 in round ℓ in this session to the ℓ-th one time program as input and passes the output back to P_1 (except secret^{ℓ} and S_{ℓ}). $\mathcal{A}$ uses the remaining mrs sessions of Π^{σ}_{PRF} to get the appropriate input wire keys for the m one time programs.

[12]This is similar to the construction of one time programs *against a malicious sender* in [GIS$^+$10].

- Thus, at the end of the concurrent interaction, $\mathcal{A}$ has recovered the secret $\mathsf{secret} = \mathsf{secret}^1 || \ldots || \mathsf{secret}^m$ with probability 1.

The only remaining step is to show that any ideal world adversary $\mathcal{S}$ fails to recover secret with a noticeable probability given the same auxiliary input *and $mrs + 1$ queries to the ideal functionality F^{σ}_{PRF}*. This would show the distinguishability of the ideal and the real world distributions and hence the insecurity of the protocol Π^{σ}_{PRF}. We consider two different cases.

- The ideal adversary $\mathcal{S}$ makes more than rs queries for a single one time program with a noticeable probability. Then, it is guaranteed that there would be at least one OTP such that the ideal adversary $\mathcal{A}$ fails to learn any information about its output. Hence, it fails to recover secret with a noticeable probability.
- The adversary makes up to rs queries for every one time program except with negligible probability. However in this case, it is easy to show that by ideal/real world security of our one time program construction, this is similar to interacting with an external party P_2 in the protocol $m\Pi^{\sigma}_{PRF}$. In this case, the $\mathcal{S}$ can recover secret only with a negligible probability as already shown in stage 1 of our proof.

This contradicts the security of the protocol Π^{σ}_{PRF} and completes our proof. The above proof also holds for the following worst case notion of a pseudorandom function: for every algorithm A, there exists at least one key k for which the output of the PRF looks indistinguishable from uniform on all input queries. This would show that there exists at least one input for which the ideal adversary fails hence showing insecurity of the protocol Π^{σ}_{PRF}.

Acknowledgement. We thank Rafael Pass for useful suggestions and in particular for suggesting to present a simpler protocol first based on compilation only with concurrent zero-knowledge. We thank Amit Sahai for many useful suggestions. Thanks also to Abhishek Jain and Rafail Ostrovsky for useful discussions. Finally, we thank Yuval Ishai to urging us to explore connections with other published works more deeply. This led to a substantial strengthening of our results.

References

[AMP04] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In Cachin and Camenisch [CC04], pages 40–55.

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

[BCL$^+$05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, pages 361–377, 2005.

[BCNP04] B. Barak, R. Canetti, J.B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS*, pages 345–354, 2006.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552. IEEE Computer Society, 2005.

[Can08] Ran Canetti, editor. *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*. Springer, 2008.

[CC04] Christian Cachin and Jan Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.

[CHK$^+$05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT*, pages 404–421, 2005.

[CKL06] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\widetilde{\Omega}\ (\log n)$ rounds. In *STOC*, pages 570–579, 2001.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security from standard assumptions. In *FOCS*, 2010.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260. IEEE Computer Society, 2009.

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

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Cachin and Camenisch [CC04], pages 1–19.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Rabin [Rab10], pages 277–294.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008.

[GL01] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO*, pages 408–432, 2001.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT*, pages 524–543, 2003.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the 19th annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *STOC*, 2011.

[HL08a] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standardsmartcards. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 491–500. ACM, 2008.

[HL08b] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Canetti [Can08], pages 155–175.

[Kat] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Eurocrypt 2007*.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, pages 475–494, 2001.

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

[Lin03a] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC*, pages 683–692. ACM, 2003.

[Lin03b] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403. IEEE Computer Society, 2003.

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *J. Cryptology*, 21(2):200–249, 2008.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LPTV10] Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable zero knowledge proofs. In Rabin [Rab10], pages 429–446.

[MPR06] Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *FOCS*, pages 367–378. IEEE Computer Society, 2006.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2003.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. pages 232–241, 2004.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. 2003.

[PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, pages 563–572, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In László Babai, editor, *STOC*, pages 242–251. ACM, 2004.

[PSW09] Benny Pinkas, Thomas Schneider 0003, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In Canetti [Can08], pages 553–570.

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE, 1986.

A Building Blocks

In this section, we explain some of the building blocks that we use in our construction.

Statistically Binding Commitments. In our protocol, we shall use a non-interactive statistically binding commitment scheme. An example of such a scheme is the following. Let P be a one-way permutation, and H be the hard-core predicate associated with P. Then the commitment to a bit b is computed as $P(x)||H(x) \oplus b$, where x is a random string in the domain of P. The decommitment simply consists of the string x. We denote such a commitment scheme by COM.

Semi-Honest Two Party Computation. We will also use a semi-honest two party computation protocol Π_{SH} that emulates the functionality $\mathcal{F}$ in questions in the stand-alone setting. The existence of such a protocol Π_{SH} follows from [Yao86, GMW87].

A.1 Preamble from DGS [DGS09]

In this subsection, we describe the preamble from [DGS09] and some of its main properties useful for our context. Let n be the security parameter and m be a parameter that determines the round-complexity of the protocol. Let COM be a statistically binding commitment scheme. The preamble (as with preambles in [RK99, PV08]) consists of two main phases described below.

Commitment Phase Let β be the bit string the committer wishes to commit. In the commit phase, the committer prepares mn pairs of secret shares $\{\alpha^0_{i,\ell}, \alpha^1_{i,\ell}\}_{i\in[n],\ell\in[m]}$ such that $\alpha^0_{i,\ell} \oplus \alpha^1_{i,\ell} = \beta$ for all i, ℓ. The committer will commit to all these bit strings using COM, with fresh randomness each time[13]. The committer then sends these mn commitments to the receiver.

Challenge-Response Phase This phase consists of m iterations where in the ℓ^{th} iteration, the receiver sends a random n-bit string $b_\ell = b_{1,\ell}, \ldots, b_{n,\ell}$, and the committer decommits to $\mathrm{COM}(\alpha^{b_{1,\ell}}_{1,\ell}), \ldots, \mathrm{COM}(\alpha^{b_{n,\ell}}_{n,\ell})$. On reaching this point, the receiver considers the preamble to have "concluded".

There is an optional *preamble opening phase* where the committer opens all the commitments made in the commitment phase, and the receiver verifies the consistency of the revealed values. On reaching this point, the receiver is supposed to have "accepted" the preamble.

Simulator CEC-Sim. We call the simulator for the DGS preamble CEC-Sim, where CEC stands for concurrently-extractable commitment (intuitively, the DGS preamble can be viewed as a concurrently-extractable commitment). Consider polynomially many concurrent sessions of the DGS preamble that we wish to simulate. The simulator CEC-Sim produces an ordered list of "threads of execution", where a thread of execution (consisting of the views of all the parties) is a perfect simulation of a prefix of an actual execution. In particular, the *main thread*, is a perfect simulation of a complete execution, and this is the execution thread that is output by the simulator. Any other thread is referred to as a *look-ahead thread*. Here, each thread shares a possibly empty prefix with the previous thread.

The goal of CEC-Sim is, for each preamble commitment that it comes across in any session in any thread, to extract the preamble secret before that preamble is concluded in that thread. CEC-Sim is said to "get stuck" if it fails in extracting the preamble secret in a session on a thread such that the preamble commit phase of that session in that thread is concluded. The probability of CEC-Sim getting "stuck" is negligible, as stated below.

Lemma 4 *([DGS09]). Consider a concurrent adversarial committer and a receiver running polynomially many (in the security parameter) sessions of a protocol with the DGS preamble. Then except with negligible probability, in every thread of execution output by CEC-Sim; if the receiver accepts a DGS preamble commit phase as valid, then at the point when that preamble is concluded, CEC-Sim would have already recorded the secret of that preamble.*

Modified DGS preamble. In our construction, we shall additionally make use of a *modified* version of the DGS preamble (referred to as *m*DGS) where, for a given receiver challenge, the committer does not "open" the commitments, but instead simply reveals the committed value (without revealing the randomness used to create the commitment) and proves its correctness by using a sWI. Additionally, the committer gives a proof of consistency of the committed values using a sWI.

We note that lemma 4 is applicable to the *m*DGS preamble as well as long as the sWI are sound. In our construction, the statements for sWI will have a "trapdoor condition" that will allow our simulator to cheat; however, in our security proof, we will ensure that that the trapdoor condition is false for each instance of sWI where the adversary plays the role of the prover. Therefore, we will still be able to use lemma 4.

[13]Note that statistically binding commitments are used in this preamble. Therefore, if the receiver accepts the preamble, then except with negligible probability, there is a well-defined value α in the commitments, and it is this value that the receiver accepted as the committer's secret in the preamble.

A.2 Concurrent Non-Malleable Zero Knowledge Argument

Concurrent non-malleable zero knowledge (CNMZK) considers the setting where a man-in-the-middle adversary is interacting with several honest provers and honest verifiers in a concurrent fashion: in the "left" interactions, the adversary acts as verifier while interacting with honest provers; in the "right" interactions, the adversary tries to prove some statements to honest verifiers. The goal is to ensure that such an adversary cannot take "help" from the left interactions in order to succeed in the right interactions. Recently, using only one-way functions, Barak, Prabhakaran and Sahai [BPS06] gave the first construction of a concurrent non-malleable zero knowledge (CNMZK) argument for every language in **NP** with perfect completeness and negligible soundness error. They gave a *simulation-extractability* [PR05] based definition for CNMZK, that, informally speaking, requires the construction of a machine called the simulator-extractor that generates the view of the man-in-the-middle adversary and additionally also outputs a witness from the adversary for each "valid" proof given to the verifiers in the right sessions. We will use the BPS-CNMZK protocol to guarantee non-malleability in our construction. However, we would require some changes to the original construction, as described below. We stress that the original security guarantees of BPS-CNMZK still follow despite our modifications, as should be evident from our description. We now describe the (modified) BPS-CNMZK construction (henceforth referred to as *m*BPS-CNMZK).

At a high level, the *m*BPS-CNMZK protocol consists of two main phases - (a) a *preamble phase*, where the verifier commits to a random secret (say) σ using a DGS [DGS09] preamble, and (b) a *post-preamble phase*, where the prover proves an NP statement. The construction allows straight-line simulation of the post-preamble phase if the preamble secret σ is provided to the simulator. We now give more details.

Let P and V denote the prover and the verifier respectively. Let L be an NP language with a witness relation R. The common input to P and V is a statement y. P additionally has a private input w (witness to y). The *m*BPS-CNMZK protocol proceeds as follows.

Phase I. P and V engage in an execution of the DGS preamble[14] with the number of rounds m (where m is a parameter determined by our ideal world as discussed later), where V commits to a random secret.

Phase II. P commits to 0 using a statistically-hiding commitment scheme. Let c be the commitment string. Additionally, P proves the knowledge of a valid decommitment to c using a statistical zero-knowledge argument of knowledge (sZKAOK).

Phase III. P and V now engage in the execution of the opening phase of the phase I preamble. Let σ be the preamble secret (revealed by V).

Phase IV. P commits to the witness w using a public-coin extractable non-malleable commitment scheme[15]

Phase V. P now proves the following statement to V using sZKAOK:

- the value committed to in phase IV is a valid witness to y. That is, $R(y, w) = 1$, where w is committed value.
- the value committed to in phase II is the preamble secret σ.

P uses the witness corresponding to the first part of the statement.

[14]In contrast, the original BPS-CNMZK construction used the PRS preamble [PRS02].

[15]The original BPS-CNMZK construction only required a public-coin extraction phase inside the non-malleable commitment scheme. We, however, require that the entire commitment protocol be public-coin. We note that the recent protocol of Goyal [Goy11] can be instantiated to result in a (super-constant round) public-coin non-malleable commitment protocol. Similarly, one can instantiate the DDN construction [DDN00] to be public coin as well.

B Our Final Protocol

In this section, we describe our protocol Σ for a given two-party functionality $\mathcal{F}$ (extension to the case of multi-party is discussed later). Let P_1 and P_2 be two parties with private inputs x_1 and x_2 respectively. Let COM denote a non-interactive statistically binding commitment scheme. By *m*BPS-CNMZK, we will refer to the modified version of the concurrent non-malleable zero knowledge argument of [BPS06] described in Appendix A. Let $\Pi_{m\text{BPS},P_i \to P_j}$ denote an instance of the *m*BPS-CNMZK protocol where P_i plays the role of the prover. By sWI, we will refer to a statistically witness indistinguishable argument. Let Π_{SH} be any *semi-honest* two party computation protocol that emulates the functionality $\mathcal{F}$ in the stand-alone setting (as per the standard Ideal/Real world definition of secure computation). Let U_η denote the uniform distribution over $\{0,1\}^\eta$, where η is a function of the security parameter. The protocol Σ proceeds as follows.

I. Trapdoor Creation Phase.

1. $P_1 \to P_2$: P_1 creates a commitment com_1 to bit 0 using the commitment scheme COM, and sends it to P_2. P_1 and P_2 now engage in the execution of a *m*BPS-CNMZK argument $\Pi_{m\text{BPS},P_1 \to P_2}$ where P_1 proves that com_1 is a commitment to 0.

2. $P_2 \to P_1$: P_2 now acts symmetrically. Specifically, it creates a commitment com_2 to bit 0 using the commitment scheme COM, and sends it to P_1. P_2 and P_1 now engage in the execution of a *m*BPS-CNMZK argument $\Pi_{m\text{BPS},P_2 \to P_1}$ where P_2 proves that com_2 is a commitment to 0

Informally speaking, the purpose of this phase is to aid the simulator in obtaining a "trapdoor" to be used during the simulation of the protocol in the concurrent setting. This is done by having the simulator commit to 1 as opposed to 0 (and simulate the associated *m*BPS-CNMZK argument).

II. *m*DGS Preamble Phase

In this phase, each party P_i engages in the execution of a *modified* DGS preamble (henceforth referred to as *m*DGS) with P_j where it commits to its input and randomness. In our modified version of the DGS preamble, for a given receiver challenge, the committer does not "open" the commitments, but instead simply reveals the committed value (without the randomness) and proves its correctness by using a sWI. This is discussed in more detail in appendix A. Let $\Pi_{m\text{DGS},P_i \to P_j}$ denote an instance of the *m*DGS protocol where P_i plays the role of the committer.

We now describe the steps in this phase.

1. $P_1 \leftrightarrow P_2$: Generate a string $r_1 \xleftarrow{\$} U_\eta$ and let $\beta_1 = \{x_1, r_1\}$. Here r_1 is the randomness to be used (after coin-flipping with P_2) by P_1 in the execution of the protocol Π_{SH} in Phase III. We assume that $|r_1| = \eta$ is sufficiently long for that purpose. Now P_1 and P_2 engage in the execution of a *m*DGS preamble $\Pi_{m\text{DGS},P_1 \to P_2}$ in the following manner.

   The party P_1 first prepares mn pairs of secret shares $\{\alpha^0_{i,j}\}_{i\in[n],j\in[m]}$, $\{\alpha^1_{i,j}\}_{i\in[n],j\in[m]}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \beta_1$ (= $\{x_1, r_1\}$) for all i, j. Using the commitment scheme COM, P_1 commits to β_1 and all its secret shares. Denote these commitments by B_1, $\{A^0_{i,j}\}_{i\in[n],j\in[m]}$, $\{A^1_{i,j}\}_{i\in[n],j\in[m]}$. P_1 now engages in the execution of a sWI with $\mathcal{A}$ in order to prove the following statement: either

   (a) the above commit phase is "valid", i.e., there exist values $\hat\beta_1$, $\{\hat\alpha^0_{i,j}, \hat\alpha^1_{i,j}\}_{i\in[n],j\in[m]}$ such that (a) $\hat\alpha^0_{i,j} \oplus \hat\alpha^1_{i,j} = \hat\beta_1$ for all i, j, and, (b) commitments B_1, $\{A^0_{i,j}\}_{i\in[n],j\in[m]}$, $\{A^1_{i,j}\}_{i\in[n],j\in[m]}$ can be decommitted to $\hat\beta_1$, $\{\hat\alpha^0_{i,j}, \hat\alpha^1_{i,j}\}_{i\in[n],j\in[m]}$, or,

   (b) com_1 in phase I is a commitment to bit 1.

   It uses the witness corresponding to the first part of the statement.

   P_1 and P_2 now execute a challenge-response phase. For $j \in [m]$:

(a) $P_2 \rightarrow P_1$: Send challenge bits $z_{1,j}, \ldots, z_{n,j} \overset{\$}{\leftarrow} \{0,1\}^n$.

(b) $P_1 \rightarrow P_2$: Send $\alpha_{1,j}^{z_{1,j}}, \ldots, \alpha_{n,j}^{z_{n,j}}$. Now, P_1 and P_2 engage in the execution of a sWI, where P_1 proves the following statement: either (a) commitments $A_{1,j}^{z_{1,j}}, \ldots, A_{n,j}^{z_{n,j}}$ can be decommitted to $\alpha_{1,j}^{z_{1,j}}, \ldots, \alpha_{n,j}^{z_F n,j}$ respectively, or (b) com_1 in Phase I is a commitment to bit 1. It uses the witness corresponding to the first part of the statement.

2. $P_2 \leftrightarrow P_1$: P_2 now acts symmetrically.

At the end of this phase, party P_i is committed to its input and randomness. Informally speaking, the purpose of this phase is aid the simulator in extracting the adversary's input and randomness in the concurrent setting.

III. Secure Computation Phase.

In this phase, we will run an execution of the semi-honest two party protocol Π_{SH}. Since Π_{SH} is secure only against semi-honest adversaries, we will first run a coin-flipping protocol to force the coins of each party to be unbiased and then "compile" Π_{SH} with sWI to enforce honest behavior on the parties. We now give more details.

Coin Flipping. P_1 and P_2 first engage in a coin-flipping protocol. More specifically,

1. $P_1 \rightarrow P_2$: P_1 generates $r_2' \overset{\$}{\leftarrow} U_\eta$ and sends it to P_2. Define $r_2'' = r_2 \oplus r_2'$.

2. $P_2 \rightarrow P_1$: Similarly, P_2 generates $r_1' \overset{\$}{\leftarrow} U_\eta$ and sends it to P_1. Define $r_1'' = r_1 \oplus r_1'$.

Now r_1'' and r_2'' are the random coins that P_1 and P_2 will use in the execution of protocol Π_{SH}.

Protocol Π_{SH}. Let the protocol Π_{SH} have t rounds where one round is defined to have a message from P_1 to P_2 followed by a reply from P_2 to P_1. Let transcript $T_{1,j}$ (resp., $T_{2,j}$) be defined to contain all the messages exchanged between P_1 and P_2 before the point party P_1 (resp., P_2) is supposed to send a message in round j. Now, each message sent by either party in protocol Π_{SH} is compiled into a message block in Σ. For $j = 1, \ldots, t$:

1. $P_1 \rightarrow P_2$: P_1 sends the next message $\Delta_{1,j} (= \Pi_{\text{SH}}(T_{1,j}, x_1, r_1''))$ as per protocol Π_{SH}. Now, P_1 and P_2 engage in the execution of a sWI where P_1 proves the following statement: either

   (a) there exists a value $\hat{\beta}_1 = \{\hat{x}_1, \hat{r}_1\}$ such that (a) the commitment B_1 in phase II.1 can be decommitted to $\hat{\beta}_1 = \{\hat{x}_1, \hat{r}_1\}$, and (b) the sent message $\Delta_{1,j}$ is consistent with input $\hat{x}_1$ and randomness $\hat{r}_1 \oplus r_1'$ (i.e., $\Delta_{1,j} (= \Pi_{\text{SH}}(T_{1,j}, \hat{x}_1, \hat{r}_1 \oplus r_1'))$, or

   (b) com_1 in Phase I is a commitment to bit 1.

   It uses the witness corresponding to the first part of the statement. In the sequel, we will refer to the second part of the above statement as the *trapdoor* condition. Further, we will call the witness corresponding to the first part of the statement as *real* witness and that corresponding to the second part of the statement as the *trapdoor* witness.

2. $P_2 \rightarrow P_1$: P_2 now acts symmetrically.

This completes the description of the protocol Σ. We analyze the security of Σ in the next section.

C Proof of Security based on the Key Technical Property

Theorem 3 *Assuming that the ideal world for the functionality $\mathcal{F}$ satisfies the key technical property, the proposed protocol Σ securely realizes the functionality $\mathcal{F}$ as per definitions in section 2.2 (under suitable cryptographic assumptions).*

Let there be N parties in the system where different pairs of parties are involved in one or more sessions of Σ, such that the total number of sessions k is polynomial in the security parameter n. Let $\mathcal{A}$ be an adversary who controls a subset of parties. In order to prove theorem 3, we will first construct an ideal world simulator $\mathcal{S}$ that will simulate the view of $\mathcal{A}$. Then, we will argue that the output distributions of the real and ideal world executions are computationally indistinguishable. For simplicity of exposition, we will assume that exactly one party is corrupted in each session. We note that if the real and ideal distributions are indistinguishable for this case, then by using standard techniques we can easily relax this assumption.

We describe the construction of our simulator in section C.1 and argue the indistinguishability of the views in section C.2.

C.1 Description of Simulator $\mathcal{S}$

In the sequel, for any session $\ell \in [k]$, we will use the notation H to denote the honest party and $\mathcal{A}$ to denote the corrupted party. Let $\Pi_{m\text{BPS},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{BPS},\mathcal{A}\to H}$) denote an instance of *m*BPS-CNMZK where H (resp., $\mathcal{A}$) plays the role of the prover and $\mathcal{A}$ (resp., H) plays the verifier. Similarly, let $\Pi_{m\text{DGS},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{DGS},\mathcal{A}\to H}$) denote an instance of *m*DGS where H (resp., $\mathcal{A}$) plays the role of the committer and $\mathcal{A}$ (resp., H) plays the receiver. Wherever necessary, we shall augment our notations with a super-script that denotes the session number.

We now describe the simulator $\mathcal{S}$ for protocol Σ. For simplicity of exposition, we shall first describe the simulator strategy for rewinding and executing the preamble messages and then the strategy for rest of the phases (for main as well as look ahead threads).

C.1.1 The rewinding Strategy

Before going into the details of $\mathcal{S}$, we first fix some terminology. We assume there are a total of k sessions with each session having two DGS preambles given from the simulator to the adversary (one in the trapdoor creation phase as part of the *m*BPS-CNMZK argument and another in the *m*DGS phase). Hence there are a total of $2k$ such preambles. Each of these preambles has $m = \frac{2n^3D^2}{p}$ "slots" with a slot representing a rewinding opportunity. The beginning of a slot is when the simulator gives the challenge, the end of the slot being when it receives the response. In between these two messages, there might be messages of other sessions.

As with the strategy in [RK99] (and [PV08]), the DGS rewinding schedule is "adaptive". At a very high level, whenever a slot s completes, the simulator may rewind s by calling itself recursively on s. That is, the simulator chooses another challenge for s and recursively executes until either it receives the response (and hence "solves" the preamble) or it observes that the adversary has executed "too many" new slots in between or has aborted. By the time the simulator completes the preamble in any thread, our choice of the number of slots guarantees that there would exist at least one recursive level which will have at least $\frac{2n^2D^2}{p}$ slots of that preamble. Whenever the simulator observes $\frac{2n^2D^2}{p}$ slots in one level, it would rewind each of those slots exactly once and will try to solve that preamble.

The formal description of our simulator rewinding strategy CEC-Sim is given below. Some of the text is borrowed from [DGS09].

- $d_{max} = \lceil \log_n(2k \cdot m) \rceil$ will denote the maximum depth of recursion. Note that d_{max} is a constant since the number of preambles $2k$ and number of slots per preamble m is polynomial in the security parameter n. It would be helpful to keep in mind that $2k \cdot m$ is the total number of slots at depth 0 (i.e., the main thread).
- slot(i, j) will denote slot j of preamble i.
- d denotes the current depth of the recursion.

SOLVE$(x, d, h_{initial}, s)$

Let $h \leftarrow h_{initial}$
Repeat forever and update h after each step:

1. If the adversary aborts or the number of slots in h started after $h_{initial}$ (which we will call new slots) exceed $\frac{2k \cdot m}{n^d}$, return h;

2. If the next message is an adversary message of the commit phase of some preamble (i.e., commitment to the shares of the preamble secret and possibly sWI), continue and reply honestly if needed;

3. If the next message is a simulator challenge for the beginning of a slot s', choose the challenge randomly and send it to the adversary.

4. If the next message is the end message (adversary's reply) of a slot $s' = \text{slot}(i', j')$, proceed as follows:

   (a) If $s = s'$, we have succeeded in solving the target slot and hence the preamble. Return h;

   (b) Otherwise if the preamble i' has already been solved or the number of new slots (including s') of preamble i' in h started after $h_{initial}$ is less than $\frac{2n^2D^2}{p}$, the simulator need not rewind this slot. Go to the condition in Step 5;

   (c) Otherwise, we have an unsolved preamble i' such that $\frac{2n^2D^2}{p}$ of its slots (from slot$(i', j' + 1 - \frac{2n^2D^2}{p})$ to slot(i', j')) have appeared at the current level. The $\mathcal{S}$ will rewind each of these slots once and will try to solve preamble i'. Observe that the depth d_{max} of the recursion is a constant and the total number of slots in a preamble is $\frac{2n^3D^2}{p}$. This means just by the pigeonhole principle, for every preamble i', we would have this case at some level before the preamble is concluded. For each slot s" in this list of $\frac{2n^2D^2}{p}$ slots:

      i. Let h" be the prefix of h which contains all messages up to but excluding the simulator challenge for s". Set $h^* \leftarrow \text{SOLVE}(x, d+1, h", s")$.

      ii. If h^* contains an accepting execution for slot s", the simulator has succeeded in solving s" and hence preamble i'.

5. If the next message is the last message of a preamble and the preamble has not been solved yet, the current set of look-ahead threads have failed. Abort and output Ext_Fail if we are in the main thread. If we are in a look-ahead thread, return.

6. If the next message is a message which doesn't belong to a preamble, the simulator strategy is given in the next subsection (for main as well as look ahead threads).

$\underline{CEC - Sim(x, z)}$

Run SOLVE$(x, 0, \perp, \perp)$ and output the view returned by SOLVE, with the following exception. When the simulator generates random challenge for a slot and it becomes equal to the another challenge generated previously in a different thread for the same slot, the simulator aborts and outputs $\perp$.

Note that the probability with simulator outputs $\perp$ is exponentially small (assuming the running time of the simulator and hence the number of challenges generated by it is polynomial) and hence we do not further analyze this event in our proof of security. Looking ahead, the core of the analysis of our rewinding strategy can be found in Lemma 2 where we prove that the probability with which the simulator outputs Ext_Fail is negligible in n.

C.1.2 Simulator Strategy in Each Phase

Trapdoor Creation Phase. Let us assume that the honest party sends the first message in this session (the simulation strategy for the opposite case can be easily derived from this case). The simulator starts phase I by sending a commitment to bit 1 (instead of committing to bit 0). Now since $\mathcal{S}$ committed to bit 1, it does not possess the real witness for the *m*BPS-CNMZK argument $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ that accompanies the bit commitment. However, recall that "straight-line" simulation of *m*BPS-CNMZK is possible if the *preamble secret* inside it is known to the simulator. This is guaranteed by the routine CEC-Sim (else CEC-Sim would have aborted by the time preamble concluded).

Hence, $\mathcal{S}$ goes ahead and gives a simulated proof (of the false statement) using the preamble secret regardless of the whether the current thread is the main thread or a look ahead thread.

$\mathcal{S}$ executes the rest of the phase (i.e., where the adversary commits and gives a *m*BPS-CNMZK that the commitment is to the value 0) honestly by running the code of an honest verifier.

***m*DGS Preamble Phase.** $\mathcal{S}$ uses the following strategy in Phase II for both the main thread as well as the look-ahead threads. Consider a session $\ell \in [k]$. Let $\Pi^{\ell}_{m\text{DGS},H\to\mathcal{A}}$ denote an instance of the *m*DGS preamble where $\mathcal{S}$ plays the role of the committer. Then in session ℓ, $\mathcal{S}$ first engages in an execution of $\Pi^{\ell}_{m\text{DGS},H\to\mathcal{A}}$ with $\mathcal{A}$ where it commits to a random value and answers $\mathcal{A}$'s challenges with random values. $\mathcal{S}$ uses the trapdoor witness in order to successfully simulate each instance of *s*WI inside $\Pi^{\ell}_{m\text{DGS},H\to\mathcal{A}}$. Note that the trapdoor witnesses are available to $\mathcal{S}$ since it committed to bit 1 (instead of 0) during Phase I.

Next, $\mathcal{S}$ engages in an execution of $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$ with $\mathcal{A}$. By using CEC-Sim, $\mathcal{S}$ has already extracted from $\mathcal{A}$ a value $\sigma^{\ell}_{m\text{DGS},\mathcal{A}\to H}$ consistent with the transcript of $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$. Note that since $\mathcal{A}$ proves consistency of commitments across the preamble (the proof later establishes that except with negligible probability, the trapdoor condition is false in the sWI where $\mathcal{A}$ acts as the prover), we have that $\sigma^{\ell}_{m\text{DGS},\mathcal{A}\to H}$ is the preamble secret of $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$. That is, $\mathcal{S}$ has extracted the input and the randomness to be used later by $\mathcal{A}$ during the execution of Π_{SH} in session ℓ.

Secure Computation Phase. By the time $\mathcal{S}$ begins the secure computation phase in any session, it would have already extracted the input and the randomness to be used by the adversary in the semi-honest two-party protocol Π_{SH} as described in the *m*DGS preamble phase. Let $S_{\Pi_{\text{SH}}}$ denote the simulator for the semi-honest two-party protocol Π_{SH} used in our construction. Now consider a session $\ell \in [k]$. $\mathcal{S}$ internally runs the simulator $S_{\Pi_{\text{SH}}}$ on adversary's input (in session ℓ) that it extracted in the last phase. $S_{\Pi_{\text{SH}}}$ starts executing, and, at some point, it makes a call to the trusted party in the ideal world with some input (say) x. Now we have two possible cases:

Case 1: $\mathcal{S}$ is currently executing the main thread. In other words, the next message to be sent by the simulator $\mathcal{S}$ would count as a message belonging to the main thread. In this case, $\mathcal{S}$ goes ahead and queries the ideal functionality with the input x for the current session ℓ. It receives the output from the functionality and forwards it the simulator $S_{\Pi_{\text{SH}}}$.

Case 2: $\mathcal{S}$ is currently executing some look ahead thread at level $d > 0$. This is the more interesting case since $\mathcal{S}$ cannot simply query the ideal functionality.[16] $\mathcal{S}$ would now use the predictor $\mathcal{P}$ guaranteed by the KTP of the ideal world. More precisely, $\mathcal{S}$ simply invokes $\mathcal{P}$ on input x and the inputs/outputs in the calls made by the simulator $S_{\Pi_{\text{SH}}}$ while $\mathcal{S}$ is trying to compute a message of the current thread. Note that these previous calls might have been answered by $\mathcal{S}$ either by making a call to the ideal functionality or using the predictor itself (in which case, correctness of the output is not guaranteed). Indeed assuming the adversary has full auxiliary information about the inputs of the honest parties, it can distinguish a correct output from an incorrect one (generated using the predictor). The core of the analysis of this prediction strategy (along with how it fits with our rewinding strategy) can be found in lemma 2.

Finally, $S_{\Pi_{\text{SH}}}$ halts and outputs a transcript $\Delta^{\ell}_{1,1}, \Delta^{\ell}_{2,1}, \ldots, \Delta^{\ell}_{1,t}, \Delta^{\ell}_{2,t}$ of the execution of Π_{SH}, and an associated randomness $r^{\ell}_{\mathcal{A}}$. Let $\hat{r}^{\ell}_{\mathcal{A}}$ be the randomness that $\mathcal{S}$ extracted from $\mathcal{A}$ in phase II. Now, $\mathcal{S}$ computes a random string $\tilde{r}^{\ell}_{\mathcal{A}}$ such that $r^{\ell}_{\mathcal{A}} = \tilde{r}^{\ell}_{\mathcal{A}} \oplus \hat{r}^{\ell}_{\mathcal{A}}$. Now, in order to force $\mathcal{A}$ to use randomness $r^{\ell}_{\mathcal{A}}$ during the execution of Π_{SH}, $\mathcal{S}$ sends $\tilde{r}^{\ell}_{\mathcal{A}}$ to $\mathcal{A}$ during the coin-flipping phase prior to the execution of Π_{SH}. Finally, $\mathcal{S}$ forces the transcript $\Delta^{\ell}_{1,1}, \Delta^{\ell}_{2,1}, \ldots, \Delta^{\ell}_{1,t}, \Delta^{\ell}_{2,t}$ onto $\mathcal{A}$ during the execution of Π_{SH}. This is done as follows. Without loss of generality, let us assume that the honest party sends the first message in this instance of Π_{SH}. Then, in round j, $1 \le j \le t$, $\mathcal{S}$ sends $\Delta^{\ell}_{1,j}$ to $\mathcal{A}$ (instead of sending a message as per the input and randomness committed to in the preamble in Phase II). $\mathcal{S}$ uses the trapdoor witness to complete the associated sWI. The proof later establishes that the trapdoor condition is false (except with negligible probability) in each sWI where $\mathcal{A}$ acts as the prover. Therefore, the reply of $\mathcal{A}$ must be the message $\Delta^{\ell}_{2,j}$ except with negligible probability.

This completes the description of $\mathcal{S}$.

[16]This is because it might end up querying the functionality multiple times for the same session (with different inputs in different threads). However it gets to query the functionality only once for a session.

Running Time of $\mathcal{S}$. To bound the number of queries $\mathcal{S}$ makes to $\mathcal{A}$, we consider the recursive execution tree (of constant depth) resulting out of $\mathcal{S}$ rewinding $\mathcal{A}$. Each call to the function SOLVE$(\cdot,\cdot,\cdot,\cdot)$ will represent one node in the execution tree. The nodes resulting from all further recursive calls to SOLVE will be treated as children of this node. Thus, the root node (at depth 0) is the call SOLVE$(x, 0, \cdot, \cdot)$ made by $CEC - Sim(x, z)$. This call results in the main thread while recursive calls give rise to the look ahead threads.

Now consider the transcript generated by a function call representing a node at depth d (excluding the transcripts generated by any further recursive calls). The number of new slots in this transcript is bounded by $2k \cdot \frac{2n^3D^2}{p}$ (in fact $\frac{2k \cdot 2n^3D^2}{pn^d}$). Now, each of these slots may have up to one look ahead thread resulting in a total of up to $4k\frac{2n^3D^2}{p}$ children for this node. Hence, the execution tree is a tree of depth up to d_{max} and degree up to $4k\frac{2n^3D^2}{p}$. Hence, the total number of nodes is bounded by $(4k\frac{2n^3D^2}{p})^{d_{max}+1}$. The transcript of each node contains up to $O(4k\frac{2n^3D^2}{p})$ queries. Hence, the total number of queries $\mathcal{S}$ makes to $\mathcal{A}$ is $O((4k\frac{2n^3D^2}{p})^{d_{max}+2})$ which is a polynomial (since d_{max} is a constant). Also, its easy to see that each query to $\mathcal{A}$ takes only PPT assuming $\mathcal{A}$ is a PPT machine. This concludes our analysis.

C.2 Indistinguishability of the Outputs

We now consider a series of hybrid experiments and show that the views of $\mathcal{A}$ in successive hybrids are indistinguishable from each other. Our initial experiment will be the actual protocol as executed by honest parties and $\mathcal{A}$ in the real world. Our final experiment will be the simulated protocol as described above.

Experiment H_0: The simulator $\mathcal{S}$ is given all the inputs of the honest parties. By running honest programs for the honest parties, it generates their outputs along with $\mathcal{A}$'s view. This is the execution in the real world in protocol Σ.

Experiment H_1: This experiment is exactly the same as the final simulated experiment except for the following. $\mathcal{S}$ still has all the honest party inputs and uses that to answer the queries of $S_{\Pi_{\text{SH}}}$ in the look ahead threads (instead of using the predictor). $\mathcal{S}$ outputs the view of $\mathcal{A}$. Each honest party outputs the response it receives from the trusted party.

The indistinguishability of the output distributions in H_0 and H_1 follows directly from the proof in [GJO10]. A self-contained proof of indistinguishability is given in appendix D.

Experiment H_2: This experiment is exactly the same as the previous one except for the following. $\mathcal{S}$ starts using the predictor in the look ahead threads. However it still has all the honest party inputs. *$\mathcal{S}$ aborts the execution of a thread* as soon as the predictor returns an incorrect output value in the execution of that threads. That is, $\mathcal{S}$ returns the view so far in the current recursive call without continuing it any further thus returning the execution to the thread one level below.

We now prove the indistinguishability of the output distributions in H_1 and H_2. This is the core of our security analysis. First observe that actually the main threads in H_1 and H_2 are identical conditioned on the event that the simulator does not fail to solve a preamble (and output Ext_Fail). We shall now prove that the extraction of the preamble secrets is not noticeably affected by this change (i.e., aborting in some look ahead threads). This is also where we use the special properties of the DGS rewinding strategy and our key technical property.

Lemma 2 *The probability of the simulator outputting* Ext_Fail *in experiment H_2 is negligible.*

PROOF. The simulator (CEC-Sim routine) outputs Ext_Fail if it reaches the end of a preamble without extracting the corresponding preamble secret. Recall the step 3(c) in the CEC-Sim rewinding strategy. Since the depth d_{max} of the recursion is a constant and the total number of slots in a preamble is $\frac{2n^3D^2}{p}$, for every preamble i, there exists a depth d such that at least $\frac{2n^2D^2}{p}$ slots of the preamble i appear in a thread at depth d. Each of these $\frac{2n^2D^2}{p}$ is rewound exactly once. We would now prove that the simulator solves the preamble except with negligible probability as soon as this case happens.

Assume that the simulator output Ext_Fail with a noticeable probability. This means there should exists a preamble j and a depth d such that the following happens with a noticeable probability: at least $\frac{2n^2D^2}{p}$ slots of preamble j appear in a thread at depth d, each is rewound once but still the preamble secret of j is not extracted. Call these $\frac{2n^2D^2}{p}$ slots $s_1, \ldots, s_{\frac{2n^2D^2}{p}}$. Since each slot is rewound exactly once, this gives rise to $\frac{2n^2D^2}{p}$ look ahead threads. The preamble j is solved if the simulator receives the response to the challenge given in any of these look ahead threads. For this to not happen, each look ahead thread must have been aborted either because: (a) the predicator made an error in the look ahead thread, or, (b) the look ahead thread became "too long" or the adversary aborted in the thread on its own (see step 1 of the rewinding strategy). We will now analyze both events.

First we show that except with negligible probability, there are at least n^2 look ahead threads which were *not* aborted because of an error by $\mathcal{P}$. For each slot s_i, we define *calls belonging to it* as the calls made by $S_{\Pi_{\text{SH}}}$ while $\mathcal{S}$ is trying to compute a message between the beginning and the end of this slot. Note that these calls might have been answered by $\mathcal{S}$ either by making a call to the ideal functionality or by using the predictor (in which case, correctness of the output is guaranteed, else this thread would have been aborted; see description of $\mathcal{S}$ in this experiment). Denote the input and output tuple for calls belonging to s_i as IT_i and OT_i respectively. Also define a set S_i containing the ideal world session indices corresponding to the calls belonging to s_i. Similarly, define IT'_i and OT'_i as the calls made by $S_{\Pi_{\text{SH}}}$ while $\mathcal{S}$ is executing the look ahead thread corresponding to the slot s_i. Also define S'_i in an analogous way.

Now we define $2n^2D/p$ *blocks* of slots with the a-th block containing slots from s_{aD+1} to $s_{(a+1)D}$. By the KTP guarantee, for the collection of sets S_{aD+1} to $S_{(a+1)D}$ (corresponding to block a), the predictor can be used to predict the output tuples corresponding to at least one set with probability p. However, we would be interested in predicting the output tuples for a set S'_i as opposed to S_i (to not abort in the look ahead thread).

For a block a, choose a random index $b \in [D]$ and look at the predictor execution for S'_{aD+b}.

- We claim that the predictor does not make an error for any of the calls corresponding to the set S'_{aD+b} with probability at least p/D. This is because the KTP guarantees that the predictor would successfully work for at least one of the sets S_{aD+1} to $S_{(a+1)D}$ with probability at least p. Hence, it would work for set S_{aD+b} at least with probability p/D (given all the input/output tuples corresponding to the calls in this thread so far). However the distribution of S'_{aD+b} is *identical* to S_{aD+b} given the entire thread up to the point where slot s_{aD+b} is supposed to begin (in other words, one is in fact interchangeable with the other).

- Hence, predictor is successful in set S'_{aD+b} and hence in the look ahead thread for at least one slot in block a with probability p/D. Since there are a total of $2n^2D/p$ blocks, the expected number of look ahead threads where the predictor is successful is $2n^2$.

- This also implies that, except with negligible probability, there are *at least* n^2 look ahead threads which were not aborted because of an error by the predictor $\mathcal{P}$ (by the multiplicative form of Chernoff bounds).

Now this must mean that there exists at least n^2 slots in a thread at depth d such that each was rewound once but the corresponding look ahead thread was aborted because it either had too many new slots or the adversary aborted on its own. Before the we analyze the probability of this event, we first define a $(d+1)$**-good** slot. A slot at depth d is called $(d+1)$-good if:

(a) it has a maximum of $\frac{2k \cdot 2n^3D^2}{pn^{d+1}}$ new slots between its start and finish messages, and,

(b) it is such that the adversary did not abort the thread before its finish message

Observe that at most n slots out of the $\frac{2n^2D^2}{p}$ slots at depth d (being rewound) may *not* be $(d+1)$-good. This is because all of them obviously satisfy the condition (b) above (i.e., them all finished before the thread was aborted; this is why they are getting rewound). Plus, if more than n of them have $\frac{2k \cdot 2n^3D^2}{pn^{d+1}}$ new slots in between their start and finish messages, the total number of new slots at level d will exceed $\frac{2k \cdot 2n^3D^2}{pn^{d}}$ (which is impossible; see step 1 of our rewinding strategy).

Now to analyze the probability of event Ext_Fail, we consider the following experiment. Consider the point (in the thread at depth d) where the first of these $\frac{2n^2D^2}{p}$ slot begins. The simulator chooses two random tapes and creates two threads (one using each tape) forking off from this point. The simulator uses the predictor to predict the outputs in both the threads. If the predictor fails to correctly predict the outputs in both of them, the simulator proceeds by randomly choosing one of the random tapes to construct the current thread at depth d and the other to construct the corresponding look-ahead thread for this slot. However, in case the the predictor is successful in predicting the output in at least one of threads, the simulator queries an external party B, gets a random bit and then accordingly chooses one of the random tapes to construct the current thread and the other to construct the look-ahead thread. (The strategy followed to construct the current thread and the look-ahead remains the same as in hybrid H_2; we only change the way simulator chooses its random tape for running different parts of the simulation.) Clearly, the output of the simulator in this experiment remains identical to that in hybrid H_2.

As we have shown before, for at least n^2 of these $\frac{2n^2D^2}{p}$ slots, the predictor correctly predicts the output (in at least one of the two threads). Now we consider two disjoint events:

- At least $\frac{n^2}{2}$ of the n^2 threads (for which the predictor gives correct output) are $(d+1)$-good. Conditioned on this event, clearly the probability of the simulator outputting $\mathsf{Ext\_Fail}$ is $2^{-\frac{n^2}{2}}$. This is because even if one of these $\frac{n^2}{2}$ threads (for which the predictor gives the correct output and $(d+1)$-good property holds) is chosen to be a look-ahead by the external party B, our simulation will be successful.
- At least $\frac{n^2}{2}$ of these n^2 threads are *not* $(d+1)$-good. Conditioned on this event, clearly, the expected number of slots which are not $(d+1)$-good in the current thread at depth d is at least $\frac{n^2}{4}$ (since each is chosen to be a part of the current thread with probability $\frac{1}{2}$). By Chernoff bounds, except with negligible probability conditioned on this event, the current thread at depth d will have at least $\frac{n^2}{4}$ slots which are not $(d+1)$-good. However we have already shown that at most n slots out of the $\frac{2n^2D^2}{p}$ slots at depth d (being rewound) may not be $(d+1)$-good.

This is in contradiction to the fact that the probability of $\mathcal{S}$ outputting $\mathsf{Ext\_Fail}$ is noticeable.

Experiment H_3: This experiment corresponds to the final simulated experiment. That is, $\mathcal{S}$ no longer has the honest party inputs and hence does not abort the look ahead threads where the predictor makes an error. Observe that in this hybrid, the probability of $\mathcal{S}$ outputting $\mathsf{Ext\_Fail}$ can only go down compared to that in H_2. Hence, indistinguishability of the output distributions in H_2 and H_3 immediately follows thus completing the proof.

Extension to the Multi-Party Case. The above protocol can be generalized to the multi-party case using standard ideas. We provide a sketch here and defer the details to the full versions. Our protocol Σ is completely symmetric in all steps. In particular, in each stage, first the party P_1 acts and then the party P_2 acts symmetrically. If there are more parties $(P_3, \dots)$, they would follow P_2 and act symmetrically one by one. Note that our proof strategy is oblivious to the number of parties (and refer only to the adversary and the ideal functionality). In particular, our rewinding strategy only talks about the total number of preambles that adversarial parties have across all sessions (and is not affected by having more than one adversarial parties in each sessions). Thus, our simulator would extract the preamble secret for all preambles and use that to simulate the rest of the protocol. The definitions of KTP and the predictor are already oblivious to the number of parties in each session and refer only to the adversary's output tuple in each session.

D Analyzing the Distributions in H_0 and H_1

We will prove the indistinguishability of the output distributions in experiments H_0 and H_1 using a carefully designed series of intermediate hybrid experiments. Much of the text in this section is borrowed from [GJO10]. We first describe some notation.

We will use the notation H and $\mathcal{A}$ to denote the honest party and the corrupted party respectively in each session. Now consider any session between H and $\mathcal{A}$. Let $\Pi_{m\text{DGS},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{DGS},\mathcal{A}\to H}$) denote the instance of *m*DGS where H (resp., $\mathcal{A}$) plays the role of the committer and $\mathcal{A}$ (resp., H) plays the receiver. Similarly, let $\Pi_{m\text{BPS},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{BPS},\mathcal{A}\to H}$) denote the instance of *m*BPS-CNMZK where H (resp., $\mathcal{A}$) plays the role of the prover and $\mathcal{A}$ (resp., H) plays the verifier. In the sequel, whenever necessary, we will augment our notation with a super-script that denotes the session number.

Now, for any session, consider the first message that H sends to $\mathcal{A}$ during the *post-preamble phase* inside $\Pi_{m\text{BPS},H\to\mathcal{A}}$. We will refer to this message as an FM of **type I**. Further, in that session, consider the first message that H sends to $\mathcal{A}$ during the execution of Π_{SH} in phase III. We will refer to this message as an FM of **type II**. Consider an ordered numbering of all the occurrences of FM (irrespective of its type) across the m sessions. Note that there may be up to $2m$ FM's in total on any execution thread. In particular, there will be exactly $2m$ FM's on the *main* thread. For any execution thread, let FM_i denote the ith FM. Let $s(i)$ be the index of the protocol session that contains FM_i. In the sequel, our discussion will mainly pertain to the FM's on the main thread. Therefore, we omit the reference to the main thread and unless otherwise stated, it will be implicit that the FM's in our discussion correspond to the main thread.

We will now describe a series of hybrid experiments $\mathcal{H}_{i:j}$, where $i \in [1, 2m]$, and $j \in [1, 6]$. We additionally define a dummy hybrid $\mathcal{H}_{0:6}$ that represents the real execution (i.e., H_0, as defined in section 3.3). Hybrid $\mathcal{H}_{2m:6}$ will be the ideal execution (i.e., H_1, as defined in section 3.3). For each intermediate hybrid $\mathcal{H}_{i:j}$, we define a random variable $v^{i:j}$ that represents the output (including the view of the adversary and the outputs of the honest parties) of $\mathcal{H}_{i:j}$.

Looking ahead, while proving the indistinguishability of the outputs of our hybrid experiments, we will need to argue that in each session $\ell \in [m]$, the trapdoor condition is false for each instance of sWI where $\mathcal{A}$ plays the role of the prover. In the sequel, we will refer to this as the *soundness condition*. Note that the soundness condition trivially holds if we can argue that (except with negligible probability) $\mathcal{A}$ commits to bit 0 in phase I of each session. For technical reasons, however, we will in fact maintain a stronger invariant throughout the hybrids. Specifically, consider the *m*BPS-CNMZK instance $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ in session ℓ. Let y^{ℓ} denote the proof statement for this *m*BPS-CNMZK instance[17]. We will prove that in each session $\ell \in [m]$, $\mathcal{A}$ commits to a valid witness to the statement y^{ℓ} in the non-malleable commitment (NMCOM) inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$. To this end, we define m random variables, $\{\alpha^{i:j,\ell}_{\mathcal{A}}\}^{m}_{\ell=1}$, where $\alpha^{i:j,\ell}_{\mathcal{A}}$ is the value contained in NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ in Phase I of session ℓ as per $v^{i:j}$.

Soundness lemma. Before we proceed to the description of our hybrids, we first claim a "soundness" lemma pertinent to the real execution. Informally speaking, we argue that in each session $\ell \in [m]$ in the *real execution*, $\mathcal{A}$ commits to a valid witness (to the proof statement y^{ℓ}) in the non-malleable commitment inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$.

Lemma 5 *Let y^{ℓ} be the proof statement for the m*BPS-CNMZK *instance $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ in session ℓ. Then, for each session $\ell \in [m]$, if the honest party does not abort the session in the view $v^{0:6}$, then $\alpha^{0:6,\ell}_{\mathcal{A}}$ is a valid witness to the statement y^{ℓ}, except with negligible probability.*

Intuitively, the above lemma follows due the knowledge soundness of the statistical zero knowledge argument of knowledge used in *m*BPS-CNMZK. We refer the reader to [Claim 2.5, [BPS06]] for a detailed proof.

Public-coin property of NMCOM. We now describe a strategy that we will repeatedly use in our proofs in order to argue that for every session $\ell \in [m]$, the value contained in NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ remains indistinguishable as we change our simulation strategy from one hybrid experiment to another. Intuitively, we will reduce our indistinguishability argument to a specific cryptographic property (that will be clear from context) that holds in a stand-alone setting. Specifically, we will consider a stand-alone machine M_{ℓ} that runs $\mathcal{S}$ and $\mathcal{A}$ internally. Here we explain how for any session ℓ, M_{ℓ} can "expose" the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ to an external party R (i.e., M_{ℓ} will send the commitment messages from $\mathcal{A}$ to R and vice-versa, instead of handling them internally). Note that $\mathcal{S}$ may be rewinding $\mathcal{A}$ during the simulation. However, since R is a stand-alone receiver; M_{ℓ} can use its responses only on a single thread of execution.

In order to deal with this problem, we will use the following strategy. When $\mathcal{A}$ creates the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$, any message in this NMCOM from $\mathcal{A}$ on the main-thread is forwarded externally to R; the responses from R are forwarded internally to $\mathcal{A}$ on the main-thread. On the other hand, any message in this NMCOM from $\mathcal{A}$ on a look-ahead thread is handled internally; M_{ℓ} creates a response on its own and sends it internally to $\mathcal{A}$ on that look-ahead thread. We stress that this possible because NMCOM is a public-coin protocol.

[17]Recall that, informally speaking, y^{ℓ} states that $\mathcal{A}$ committed to bit 0 in phase I

In the sequel, whenever we use the above strategy, we will omit the details of the interaction between M_ℓ and R.

D.1 Description of the Hybrids

For $i \in [1, 2m]$, the hybrid experiments are described as follows.

Experiment $\mathcal{H}_{i:1}$: Same as $\mathcal{H}_{i-1:6}$, except that $\mathcal{S}$ performs rewindings *upto* FM_i using the DGS simulator CEC-Sim (see appendix A). Specifically, the rewindings are performed with the following restrictions:

- No new-look ahead threads are created beyond FM_i on the main thread (i.e., the execution is straight-line beyond FM_i).
- Consider any look-ahead thread that is created before the execution reaches FM_i on the main-thread. Then, any such look-ahead thread is terminated as soon as the execution reaches the i^{th} FM *on that thread*[18].

Additionally, $\mathcal{S}$ extracts and records the preamble secret for each preamble (where $\mathcal{A}$ play the role of the committer) that concludes before FM_i. $\mathcal{S}$ outputs an abort message $\perp$ if CEC-Sim gets stuck. Otherwise, it outputs the view of the adversary in the main thread of this simulation as $v^{i:1}$.

We now claim that,

$$v^{i-1:6} \overset{c}{\equiv} v^{i:1} \tag{1}$$

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i-1:6,\ell} \overset{c}{\equiv} \alpha_{\mathcal{A}}^{i:1,\ell} \tag{2}$$

Hybrid $\mathcal{H}_{i-1:6:1}$. In order to prove our claim, we will first consider an intermediate hybrid experiment $\mathcal{H}_{i-1:6:1}$ where $\mathcal{S}$ employs the same strategy as described above, except that whenever it fails to extract the preamble secrets, it does not abort, but instead continues the simulation and outputs the main thread. Now, since the main thread in this experiment remains unchanged from $\mathcal{H}_{i-1:6}$, it follows that:

$$v^{i-1:6} \overset{s}{\equiv} v^{i-1:6:1} \tag{3}$$

where $\overset{s}{\equiv}$ denotes statistical indistinguishability. We further claim that:

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i-1:6,\ell} \overset{c}{\equiv} \alpha_{\mathcal{A}}^{i-1:6:1,\ell} \tag{4}$$

Let us assume that equation 4 is false. That is, $\exists \ell \in [m]$ such that $\alpha_{\mathcal{A}}^{i-1:6,\ell}$ and $\alpha_{\mathcal{A}}^{i-1:6:1,\ell}$ are distinguishable by a probabilistic polynomial time (PPT) distinguisher. In this case, we can create an unbounded adversary that extracts the value contained in the non-malleable commitment inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ and is then able to distinguish between the main threads in $\mathcal{H}_{i-1:6}$ and $\mathcal{H}_{i-1:6:1}$, which is a contradiction.

We now argue that in hybrid $\mathcal{H}_{i-1:6:1}$, $\mathcal{S}$ is able to extract (except with negligible probability) the preamble secret for each preamble that concludes before FM_i. Recall that we are interested in the following two extraction processes:

1. For each session $\ell \in [m]$, consider the DGS preamble inside the *m*BPS-CNMZK argument $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$. We wish to argue that if the execution of this preamble concludes *before* FM_i, then $\mathcal{S}$ extracts (except with negligible probability) the corresponding preamble secret $\sigma^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ from $\mathcal{A}$.
2. For each session $\ell \in [m]$, consider the *m*DGS preamble $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$. We wish to argue that if the execution of this preamble concludes *before* FM_i, then $\mathcal{S}$ extracts (except with negligible probability) the corresponding preamble secret $\sigma^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$ from $\mathcal{A}$. Note that this preamble secret is in fact the input and randomness of $\mathcal{A}$.

We first note that by construction, simulator's strategy in this experiment is identical for each thread, irrespective of whether it is the main-thread or a look-ahead thread. Now consider an imaginary adversary who aborts once the execution reaches FM_i on any thread. Note that lemma 4 holds for such an adversary (i.e. the probability that the

[18]Note that the FM_i's on different executions threads may not be identical, and in particular, may correspond to different sessions

simulator fails to extract the preamble secret of a "concluded" preamble is negligible). Then, if the adversary does not abort (as is the case with $\mathcal{A}$), the probability that the simulation successfully extracts the preamble secrets must be only higher. Hence our claim follows for case 1. For case 2, we note that lemma 4 is applicable if we can argue that the *soundness condition* holds (specifically, we require that the trapdoor condition is false for each instance of sWI in $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$ if $\Pi^{\ell}_{m\text{DGS},\mathcal{A}\to H}$ concludes before FM_i). Note that this is already implied by equation 4. Hence, our claim follows for case 2 as well.

Proving Equations 1 *and* 2. Note that the only difference between $\mathcal{H}_{i-1:6:1}$ and $\mathcal{H}_{i:1}$ is that $\mathcal{S}$ outputs the abort symbol $\perp$ if CEC-Sim "gets stuck". We have shown that this event happens only with negligible probability. Hence our claim follows.

Experiment $\mathcal{H}_{i:2}$**:** Same as $\mathcal{H}_{i:1}$, except that if FM_i is of type I, then $\mathcal{S}$ simulates the post-preamble phase of $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$ in a *straight-line* fashion, explained as follows. Recall that no look-ahead threads are started once the execution reaches FM_i on the main thread. All the changes in the main thread, as explained below, are performed *after* FM_i.

Let $\sigma^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$ be the preamble secret in $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$ that $\mathcal{S}$ has already extracted. Let $y^{s(i)}$ be the proof statement in $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$. Then, $\mathcal{S}$ performs the following steps:

1. In phase II of $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, $\mathcal{S}$ creates a statistically hiding commitment (sCOM) to $\sigma^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$ (instead of a string of all zeros) and follows it up with an honest execution of sZKAOK to prove knowledge of the decommitment.

2. In phase IV of $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, $\mathcal{S}$ creates a non-malleable commitment (NMCOM) to an all zeros string (instead of a valid witness to $y^{s(i)}$).

3. In phase V of $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, $\mathcal{S}$ proves the following statement using sZKAOK: (a) the value committed to in phase IV is a valid witness to $y^{s(i)}$, or (b) the value committed to in phase II is $\sigma^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$. Here it uses the witness corresponding to the *second* part of the statement. Note that this witness is available to $\mathcal{S}$ since it already performed step 1 earlier. Below, we will refer to this witness as the *trapdoor* witness, while the witness corresponding to the first part of the statement will be referred to as the *real* witness.

Now we prove that,

$$v^{i:1} \overset{c}{\equiv} v^{i:2} \tag{5}$$
$$\forall \ell \quad \alpha^{i:1,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:2,\ell}_{\mathcal{A}} \tag{6}$$

In order to prove the above equations, we will create three intermediate hybrids $\mathcal{H}_{i:1:1}$, $\mathcal{H}_{i:1:2}$, and $\mathcal{H}_{i:1:3}$. Hybrid $\mathcal{H}_{i:1:1}$ is identical to $\mathcal{H}_{i:1}$, except that it changes its strategy to perform step 1 (as described above). Hybrid $\mathcal{H}_{i:1:2}$ is identical to $\mathcal{H}_{i:1:1}$, except that it changes its strategy to perform step 3. Finally, hybrid $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:1:2}$, except that it changes its strategy to perform step 2. Note that $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:2}$.

We now claim the following:

$$v^{i:1} \overset{c}{\equiv} v^{i:1:1} \tag{7}$$
$$\forall \ell \quad \alpha^{i:1,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:1:1,\ell}_{\mathcal{A}} \tag{8}$$
$$v^{i:1:1} \overset{c}{\equiv} v^{i:1:2} \tag{9}$$
$$\forall \ell \quad \alpha^{i:1:1,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:1:2,\ell}_{\mathcal{A}} \tag{10}$$
$$v^{i:1:2} \overset{c}{\equiv} v^{i:1:3} \tag{11}$$
$$\forall \ell \quad \alpha^{i:1:2,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:1:3,\ell}_{\mathcal{A}} \tag{12}$$

Note that equation 5 follows by combining the results of equations 7, 9, and 11. Similarly, equation 6 follows by combining the results of equations 8, 10, and 12. We now prove the above set of equations.

Proving Equations 7 and 8. We first note that sCOM and sZKAOK can together be viewed as a statistically hiding commitment scheme. Let $\overline{\text{sCOM}}$ denote this new commitment scheme. Then, equation 7 simply follows from the hiding property of $\overline{\text{sCOM}}$.

In order to prove equation 8, we will use the fact that $\overline{\text{sCOM}}$ is *statistically* hiding. Let us first assume that the claim is false, i.e., $\exists \ell \in [m]$ such that $\alpha_{\mathcal{A}}^{i:1,\ell}$ and $\alpha_{\mathcal{A}}^{i:1:1,\ell}$ are distinguishable by a PPT distinguisher D. We will create a standalone machine M_ℓ that is identical to $\mathcal{H}_{i:1}$, except that instead of simply committing to a string of all zeros using $\overline{\text{sCOM}}$ in $\Pi^{s(i)}_{m\text{BPS},H\rightarrow\mathcal{A}}$, M_ℓ takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$. Additionally, M_ℓ "exposes" the NMCOM in $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\rightarrow H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Let us describe the interaction between M_ℓ and C in more detail. M_ℓ first sends the preamble secret $\sigma^{s(i)}_{m\text{BPS},H\rightarrow\mathcal{A}}$ to C. Now, when C starts the execution of $\overline{\text{sCOM}}$ in $\Pi^{s(i)}_{m\text{BPS},H\rightarrow\mathcal{A}}$, M_ℓ forwards the messages from C to $\mathcal{A}$; the responses from $\mathcal{A}$ are forwarded externally to C. Note that if C commits to a string of all zeros in the $\overline{\text{sCOM}}$ execution, then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:1:1}$. On the other hand, if C commits to the preamble secret $\sigma^{s(i)}_{m\text{BPS},H\rightarrow\mathcal{A}}$, then the (C, M_ℓ, R) system is equivalent to $\mathcal{H}_{i:1:2}$. We will now construct a computationally unbounded distinguisher D' that distinguishes between these two executions, thus contradicting the statistically hiding property of $\overline{\text{sCOM}}$. D' simply extracts the value inside the NMCOM received by R and runs D on this input. D' outputs whatever D outputs. By our assumption, D's output must be different in these two experiments; this implies that D' output is different as well, which is a contradiction.

Proving Equations 9 and 10. Equation 9 simply follows due to the witness indistinguishability property of sZKAOK. Equation 10 follows from the fact that sZKAOK is *statistically* witness indistinguishable. The proof details are almost identical to the proof of equation 8 and therefore omitted.

Proving Equations 11 and 12. Equation 11 simply follows from the hiding property of NMCOM. To see this, we can construct a standalone machine M that internally runs $\mathcal{S}$ and $\mathcal{A}$ and outputs the view generated by $\mathcal{S}$. M is identical to $\mathcal{H}_{i:1:2}$ except that in phase IV of $\Pi^{s(i)}_{m\text{BPS},H\rightarrow\mathcal{A}}$, instead of simply committing (using NMCOM) to a valid witness (to the proof statement $y^{s(i)}$), it takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$.

In order to prove equation 12, we will use the non-malleability property of NMCOM. Let us assume that equation 12 is false, i.e., $\exists \ell \in [m]$ such that $\alpha_{\mathcal{A}}^{i:1:2,\ell}$ and $\alpha_{\mathcal{A}}^{i:1:3,\ell}$ are distinguishable by a PPT machine. We will construct a standalone machine M_ℓ that is identical to the machine M described above, except that it will "expose" the non-malleable commitment inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\rightarrow H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Now, if E commits to the witness to y^ℓ, then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:1:2}$, whereas if E commits to a random string, then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:1:3}$. From the non-malleability property of NMCOM, we establish that the value committed by M_ℓ to R must be computationally indistinguishable in both cases.

Experiment $\mathcal{H}_{i:3}$: Same as $\mathcal{H}_{i:2}$, except that if FM_i is of type I, then the simulator commits to bit 1 instead of 0 in phase I of session $s(i)$. Let $\Pi^{s(i)}_{\text{COM},H\rightarrow\mathcal{A}}$ denote this commitment.

We now claim that,

$$v^{i:2} \overset{c}{\equiv} v^{i:3} \tag{13}$$

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i:2,\ell} \overset{c}{\equiv} \alpha_{\mathcal{A}}^{i:3,\ell} \tag{14}$$

Proving Equations 13 and 14. Equation 13 simply follows from the (computationally) hiding property of the commitment scheme COM. In order to prove equation 14, we will leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in *m*BPS-CNMZK. Let us first assume that equation 14 is false, i.e., $\exists \ell \in [m]$ such that $\alpha_{\mathcal{A}}^{i:2,\ell}$ and $\alpha_{\mathcal{A}}^{i:3,\ell}$ are distinguishable by a PPT distinguisher. Note that it cannot be the case that the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\rightarrow H}$ concludes *before* $\mathcal{S}$ sends the non-interactive commitment $\Pi^{s(i)}_{\text{COM},H\rightarrow\mathcal{A}}$ in session

$s(i)$, since in this case, the execution of NMCOM is independent of $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$. Now consider the case when the NMCOM inside $\Pi^{\ell}_{m\mathrm{BPS},\mathcal{A}\to H}$ concludes *after* $\mathcal{S}$ sends $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$.

We will create a standalone machine M_ℓ that is identical to $\mathcal{H}_{i:2}$, except that instead of committing to bit 0 in $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$, it takes this commitment from an external sender C and forwards it internally to $\mathcal{A}$. Additionally, it "exposes" the NMCOM inside $\Pi^{\ell}_{m\mathrm{BPS},\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Note that if C commits to bit 0 then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:2}$, otherwise it is identical to $\mathcal{H}_{i:3}$. Now, recall that NMCOM is an extractable commitment scheme. Therefore, we now run the extractor (say) E of NMCOM on (C, M_ℓ) system. Note that E will rewind M_ℓ, which in turn may rewind the interaction between C and M_ℓ. However, since COM is a non-interactive commitment scheme, M_ℓ simply re-sends the commitment string received from C to $\mathcal{A}$ internally. Now, if the extracted values are different when C commits to bit 0 as compared to when it commits to bit 1, then we can break the (computationally) hiding property of COM, which is a contradiction.

Experiment $\mathcal{H}_{i:4}$: Same as $\mathcal{H}_{i:3}$, except that if FM_i is of type I, then $\mathcal{S}$ uses the following modified strategy. In session $s(i)$, $\mathcal{S}$ uses the trapdoor witness (instead of the real witness) in each instance of sWI where the honest party plays the role of the prover. Note that the false witness for each of these sWI must be available to the simulator at this point since it earlier committed to bit 1 in phase I of session $s(i)$.

We now claim that,

$$v^{i:3} \overset{c}{\equiv} v^{i:4} \tag{15}$$

$$\forall \ell \quad \alpha^{i:3,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:4,\ell}_{\mathcal{A}} \tag{16}$$

Proving Equations 15 and 16. Equation 15 simply follows from the witness indistinguishability of sWI by a standard hybrid argument.

In order to prove equation 16, let us first consider the simpler case where $\mathcal{S}$ uses the trapdoor witness only in the *first* instance (in the order of execution) of sWI in session $s(i)$ where the honest party plays the role of the prover. In this case, we can leverage the "statistical" nature of the witness indistinguishability property of sWI in a similar manner as in the proof of equation 10. Then, by a standard hybrid argument, we can extend this proof for multiple sWI.

Experiment $\mathcal{H}_{i:5}$: Same as $\mathcal{H}_{i:4}$, except that if FM_i is of type I, then $\mathcal{S}$ uses the following strategy in the execution of $\Pi^{s(i)}_{m\mathrm{DGS},H\to\mathcal{A}}$ in session $s(i)$:

1. During the commit phase, instead of committing to the input (and its secret shares) of the honest party, $\mathcal{S}$ commits to random strings.

2. During the challenge-response phase, instead of honestly revealing the values committed to in the commit phase (as selected by $\mathcal{A}$), $\mathcal{S}$ sends random strings to $\mathcal{A}$.

We now claim that,

$$v^{i:4} \overset{c}{\equiv} v^{i:5} \tag{17}$$

$$\forall \ell \quad \alpha^{i:4,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:5,\ell}_{\mathcal{A}} \tag{18}$$

In order to prove these equations, we will define two intermediate hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$. Experiment $\mathcal{H}_{i:4:1}$ is the same as $\mathcal{H}_{i:4}$, except that $\mathcal{S}$ also performs steps 1 as described above. Experiment $\mathcal{H}_{i:4:2}$ is the same as $\mathcal{H}_{i:4:1}$, except that $\mathcal{S}$ also performs step 2 as described above. Therefore, by definition, $\mathcal{H}_{i:4:2}$ is identical to $\mathcal{H}_{i:5}$.

We now claim the following:

$$v^{i:4} \overset{c}{\equiv} v^{i:4:1} \tag{19}$$

$$\forall \ell \quad \alpha^{i:4,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:4:1,\ell}_{\mathcal{A}} \tag{20}$$

$$v^{i:4:1} \overset{c}{\equiv} v^{i:4:2} \tag{21}$$

$$\forall \ell \quad \alpha^{i:4:1,\ell}_{\mathcal{A}} \overset{c}{\equiv} \alpha^{i:4:2,\ell}_{\mathcal{A}} \tag{22}$$

Note that equation 17 follows by combining the results of equations 19 and 21. Similarly, equation eq:b45 follows by combining the results of equations 20 and 22. We now prove the above set of equations.

Proving Equations 19 and 20. Equation 19 simply follows from the (computational) hiding property of the commitment scheme COM.

In order to prove equation 20, let us first consider the simpler case where $\mathcal{S}$ only modifies the first commitment in the commit phase in $\Pi^{s(i)}_{m\text{DGS},H\to\mathcal{A}}$. In this case, we can leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in *m*BPS-CNMZK in a similar manner as in the proof of equation 14. Then, by a standard hybrid argument, we can extend this proof to the case where $\mathcal{S}$ modifies all the commitments in the commit phase in $\Pi^{s(i)}_{m\text{DGS},H\to\mathcal{A}}$.

Proving Equations 21 and 22. Note that the main-thread is identical in hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$ since we are only changing some random strings to other random strings; furthermore, the strings being changed are not used elsewhere in the protocol. Equations 21 and 22 follow as a consequence.

Experiment $\mathcal{H}_{i:6}$: Same as $\mathcal{H}_{i:5}$, except that if FM_i is of type II, $\mathcal{S}$ "simulates" the execution of Π_{SH} in session $s(i)$, in the following manner. Let $S_{\Pi_{\text{SH}}}$ be the simulator for the semi-honest two party protocol Π_{SH} used in our construction. $\mathcal{S}$ internally runs the simulator $S_{\Pi_{\text{SH}}}$ for the semi-honest two party protocol Π_{SH} on $\mathcal{A}$'s input in session $s(i)$ that was extracted earlier. When $S_{\Pi_{\text{SH}}}$ makes a query to the trusted party with some input, $\mathcal{S}$ selects a session index s' and forwards the query to the trusted party in the same manner as explained earlier in section C.1. The response from the trusted party is passed on to $S_{\Pi_{\text{SH}}}$. Further, $\mathcal{S}$ decides whether the output must be sent to the honest party in the same manner as explained earlier. $S_{\Pi_{\text{SH}}}$ finally halts and outputs a transcript of the execution of Π_{SH}, and an associated random string for the adversary.

Now, $\mathcal{S}$ forces this transcript and randomness on $\mathcal{A}$ in the same manner as described in section C.1. We claim that during the execution of Π_{SH}, each reply of $\mathcal{A}$ must be consistent with this transcript, except with negligible probability. Note that we have already established from the previous hybrids that the *soundness condition* holds (except with negligible probability) at this point. This means that the trapdoor condition is false for each instance of sWI in session $s(i)$ where $\mathcal{A}$ plays the role of the prover. Then our claim follows from the soundness property of sWI used in our construction.

We now claim that:

$$v^{i:5} \overset{c}{\equiv} v^{i:6} \tag{23}$$

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i:5,\ell} \overset{c}{\equiv} \alpha_{\mathcal{A}}^{i:6,\ell} \tag{24}$$

Proving Equation 23. Informally speaking, equation 23 follows from the semi-honest security of the two-party computation protocol Π_{SH} used in our construction. We now give more details.

We will construct a standalone machine M that is identical to $\mathcal{H}_{i:5}$, except that instead of engaging in an honest execution of Π_{SH} with $\mathcal{A}$ in session $s(i)$, it obtains a protocol transcript from an external sender C and forces it on $\mathcal{A}$ in the following manner. M first queries the ideal world trusted party on the extracted input of $\mathcal{A}$ for session $s(i)$ in the same manner as explained above for $\mathcal{S}$. Let $x_{\mathcal{A}}^{s(i)}$ denote the extracted input of $\mathcal{A}$. Let $x_H^{s(i)}$ denote the input of the honest party in session $s(i)$. Let K be the output that M receives from the trusted party. Now M sends $x_H^{s(i)}$ along with $x_{\mathcal{A}}^{s(i)}$ and K to C and receives from C a transcript for Π_{SH} and an associated random string. M forces this transcript and randomness on $\mathcal{A}$ in the same manner as $\mathcal{S}$ does. Now, the following two cases are possible:

1. C computed the transcript and randomness by using *both* the inputs - $x_H^{s(i)}$ and $x_{\mathcal{A}}^{s(i)}$ - along with the output K. In this case, the transcript output by C is a real transcript of an honest execution of Π_{SH}.

2. C computed the transcript and randomness by using only adversary's input $x_{\mathcal{A}}^{s(i)}$, and the output K. In this case C simply ran the simulator $S_{\Pi_{\text{SH}}}$ on input $x_{\mathcal{A}}^{s(i)}$ and answered its query with K. The transcript output by C in this case is a simulated transcript for Π_{SH}.

In the first case, the (C, M) system is identical to $\mathcal{H}_{i:5}$, while in the second case, the (C, M) system is identical to $\mathcal{H}_{i:6}$. By the (semi-honest) security of Π_{SH}, we establish that the output of M must be indistinguishable in both the cases, except with negligible probability. This proves equation 23.

Proving Equation 24. We will leverage the semi-honest security of the two-party computation protocol Π_{SH} and the extractability property of the non-malleable commitment scheme in *m*BPS-CNMZK to prove equation 24.

Specifically, we will construct a standalone machine M_ℓ that is identical to M as described above, except that it "exposes" the NMCOM in $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Note that if C produces a transcript Π_{SH} according to case 1 (as described above), then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:5}$. On the other hand, if C produces a transcript for Π_{SH} according to case 2, then the (C, M_ℓ, R) system is identical to $\mathcal{H}_{i:6}$. We can now run the extractor E of NMCOM on (C, M_ℓ) system. Note that E will rewind M_ℓ, which in turn may rewind the interaction between C and M_ℓ. However, since this interaction consists of a single message from C, M_ℓ simply re-uses (if necessary) the transcript received from C in order to interact with $\mathcal{A}$ internally. Now, if the extracted values are different in case 1 and case 2, then we can break the semi-honest security of Π_{SH}, which is a contradiction.

E The Garbled Circuit Generation Algorithm

Here we review generation of a garbled circuit for the given circuit from [LP09]. Consider a circuit C with fan-in two gates. For every wire in the circuit, there will be two random strings k_j^0 and k_j^1 corresponding to bit values 0 and 1 respectively. Then, the garbled circuit is computed by "garbling" every gate of the circuit individually. Consider an arbitrary gate $g : \{0,1\} \times \{0,1\} \to \{0,1\}$ in the circuit. Let the two input wires going to g be labeled w_i and w_j, and let the output wire coming out of g be labeled w_o. Furthermore, let $k_i^0, k_i^1, k_j^0, k_j^1, k_o^0, k_o^1$ be the six random keys corresponding to w_i, w_j and w_o. We wish to be able to compute $k_o^{g(a,b)}$ from k_i^a and k_j^b. The gate g is defined by the following four values:

$$\begin{aligned}
c_{0,0} &= E_{k_i^0}(E_{k_j^0}(k_o^{g(0,0)})) \\
c_{0,1} &= E_{k_i^0}(E_{k_j^1}(k_o^{g(0,1)})) \\
c_{1,0} &= E_{k_i^1}(E_{k_j^0}(k_o^{g(1,0)})) \\
c_{1,1} &= E_{k_i^1}(E_{k_j^1}(k_o^{g(1,1)}))
\end{aligned}$$

The actual garbled gate is a random permutation of the above four ciphertexts. The encryption scheme E is a private-key encryption scheme with *indistinguishable encryptions for multiple messages*, and has an *elusive efficiently verifiable range*; see [LP09] for more detail. One instantiation of the encryption scheme is the following: Let $F = \{f_k\}$ be a family of pseudorandom functions, where $f_k : \{0,1\}^n \to \{0,1\}^{2n}$ for $k \in \{0,1\}^n$. Then define $E_k(x) =< r, f_k(r) \oplus x0^s >$ where $x \in \{0,1\}^n, r \in_R \{0,1\}^n$ and $x0^n$ denotes the concatenation of x and 0^n. The garbled circuit is simply the collection of these garbled gates in addition to a translation table that translates the random keys corresponding to the output wires to their actual bit value.