Password-Authenticated Session-Key Generation on the Internet in the Plain Model

Vipul Goyal* Abhishek Jain† Rafail Ostrovsky‡

Abstract

The problem of password-authenticated key exchange (PAKE) has been extensively studied for the last two decades. Despite extensive studies, no construction was known for a PAKE protocol that is secure in the plain model in the setting of *concurrent self-composition*, where polynomially many protocol sessions with the same password may be executed on the distributed network (such as the Internet) in an arbitrarily interleaved manner, and where the adversary may corrupt any number of participating parties.

In this paper, we resolve this long-standing open problem. In particular, we give the first construction of a PAKE protocol that is secure (with respect to the standard definition of Goldreich and Lindell) in the fully concurrent setting and without requiring any trusted setup assumptions. We stress that we allow polynomially-many concurrent sessions, where polynomial is not fixed in advance and can be determined by an adversary an an adaptive manner. Interestingly, our proof, among other things, requires important ideas from *Precise* Zero Knowledge theory recently developed by Micali and Pass in their STOC'06 paper.

*Microsoft Research, India, vipul@microsoft.com.

†UCLA Department of Computer Science, abhishek@cs.ucla.edu. Supported in Part by NSF grants 0830803, 0916574.

‡UCLA Department of Computer Science and Department of Mathematics, rafail@cs.ucla.edu Supported in part by IBM Faculty Award, Xerox Innovation Group Award, the Okawa Foundation Award, Intel, Lockheed Martin, Teradata, NSF grants 0716835, 0716389, 0830803, 0916574 and U.C. MICRO grant.

1 Introduction

The problem of password authenticated key exchange (PAKE) has been studied since early 1990's. PAKE involves a pair of parties who wish to establish a high entropy session key in an authenticated manner when their *a priori* shared secret information only consists of a (possibly low entropy) password. More formally, the problem of PAKE can be modeled as a two-party functionality $\mathcal{F}$ involving a pair of parties P_1 and P_2; if the inputs (passwords) of the parties match, then $\mathcal{F}$ outputs a uniformly distributed session key, else it outputs $\perp$. Hence the goal of PAKE is to design a protocol that securely realizes the functionality $\mathcal{F}$. Unfortunately, positive results for secure multi-party computation (MPC) [Yao86, GMW87] do not immediately translate to this setting; the reason being that known solutions for secure MPC require the existence of authenticated channels – which is in fact the end goal of PAKE. Therefore, very informally speaking, secure multi-party computation and PAKE can be viewed as complementary problems.

The problem of password authenticated key exchange was first studied by Bellovin and Meritt [BM92]. This was followed by several additional works proposing protocols with only heuristic security arguments (see [KOY09] for a survey). Subsequently, PAKE was formally studied in various models, including the random oracle/ideal cipher model, common reference string (CRS) model, and the plain model (which is the focus of this work). We briefly survey the state of the art on this problem. The works of Bellare et al [BPR00] and Boyko et al [BMP00] deal with defining and constructing PAKE protocols in the ideal cipher model and random oracle model respectively. In the CRS model, Katz, Ostrovsky and Yung [KOY01] gave the first construction for PAKE without random oracles based on the DDH assumption. Their result were subsequently improved by Gennaro and Lindell [GL03], and Genarro [Gen08]. Again in the CRS model, Canetti, Halevi, Katz, Lindell and MacKenzie [CHK$^+$05] proposed new definitions and constructions for a PAKE protocol in the framework of Universal Composability [Can01]. They further proved the *impossibility* of a Universally Composable PAKE construction in the plain model.

Goldreich and Lindell [GL01] formulated a new simulation-based definition for PAKE and gave the first construction for a PAKE protocol in the *plain model*. Their construction was further simplified (albeit at the cost of a weaker security guarantee) by Nguyen and Vadhan [NV04]. Recently, Barak et al [BCL$^+$05] gave a very general construction for a PAKE protocol that is secure in the bounded-concurrent setting (see below) in the plain model.

To date, [GL01, NV04] and [BCL$^+$05] remain the only known solutions for PAKE in the plain model. However, an important limitation of Goldreich and Lindell [GL01] (as well as Nguyen and Vadhan [NV04]) is that their solution is only relevant to the stand-alone setting where security holds only if a single protocol session is executed on the network. A more natural and demanding setting is where several protocol sessions may be executed concurrently (a typical example being protocols executed over the Internet). In such a setting, an adversary who controls parties across different sessions may be able to mount a coordinated attack; as such, stand-alone security does not immediately translate to concurrent security [FS90]. In the context of PAKE, this problem was was fully resolved assuming CRS trusted setup (see below) and only partially addressed in the plain model by Barak, Canetti, Lindell, Pass and Rabin [BCL$^+$05] who gave a construction that maintains security in the setting of *bounded-concurrency*. In this setting, an *a priori* bound is known over the number of sessions that may be executed concurrently at any time; this bound is crucially used in the design of the protocol. It is natural to consider the more general setting of full concurrent self-composition, where any polynomially many protocol sessions (with no a priori bound) with the same password may be executed in an arbitrary interleaved manner by an adversary who may corrupt any number of parties. We stress that although the works of [KOY01, KOY02, GL03, CHK$^+$05, KOY09] solve this problem (where [KOY01, GL03] are secure under self-composition, and [KOY02] also enjoy forward secrecy, while [CHK$^+$05] is secure under general-composition), they all require a trusted setup in the form of a common reference string. Indeed, to date, no constructions are known for a PAKE protocol that is secure in the plain model in the setting of concurrent self-composition.

Our Contribution. In this paper, we resolve this open problem. In particular, we give the first construction of a PAKE protocol in the plain model that allows for concurrent executions of the protocol between

parties with the same password. Our techniques rely on several previous works, most notably the works of Barak, Prabhakaran and Sahai [BPS06] and Micali and Pass [MP06].

Our construction is proven secure as per the definition of Goldreich and Lindell [GL01] in the concurrent setting. We stress that Lindell's impossibility result [Lin04] for concurrent self-composition is *not* applicable here since (a) Goldreich and Lindell used a *specific* definition that is different from the standard paradigm for defining secure computation[1], and (b) further, they only consider the scenario where the honest parties hold fixed inputs (while Lindell's impossibility result crucially requires adaptive inputs).

In fact, our security definition is somewhat stronger than the one by Goldreich and Lindell [GL01]. The definition in [GL01], for example, does not consider the case where the adversary may have some a priori information on the password of the honest parties in a protocol execution. We consider an improved simulation-based security model similar to that proposed by [BMP00]. More specifically, in our model, the simulator in the ideal world is empowered to make a *constant* number of queries per (real world) session to the ideal functionality (as opposed to just one)[2]. Our security definition then requires computational indistinguishability of the output distributions of real and ideal world executions in keeping with the standard paradigm for secure computation. As noted in [GL06], this improved definition implies the original definition of Goldreich and Lindell (see appendix B for a proof sketch).

In our main construction, we consider the setting where the honest parties across the (polynomially-many) concurrent executions hold the same password or independently chosen passwords[3]. An example of the same password case is when a server expects a specific password for authentication and several parties are trying to authenticate simultaneously.

We note that our techniques and constructions are quite general. Our construction can be instantiated with a basic semi-honest secure computation protocol for any PPT computable functionality. This would lead to a concurrently secure protocol for that functionality as per the security definition where we allow the simulator to make an expected constant number of calls to the ideal function per (real world) session. The meaningfulness of such a definition is shown in the case of password based key exchange which is the focus of this work (more precisely, by comparing it with the definition of [GL06]). However we anticipate that the above general construction with such security guarantees might be acceptable in many other settings as well.

A related model is that of *resettably secure computation* proposed by Goyal and Sahai [GS09]. In resettably secure computation, the ideal simulator is given the power to reset and query the trusted party any (polynomial) number of times. However there are important differences. Goyal and Sahai [GS09] consider only the "fixed role" setting and only one of the parties can be thought of as accepting concurrent sessions. This means that the key technical problems we face in the current work (arising out of the possibility of mauling attacks in the concurrent setting) do not arise in [GS09]. Secondly, [GS09] do not try to optimize (or even bound) the number of queries the ideal simulator makes to the trusted party per session.

Overview of Main Ideas. Note that in the setting of concurrent self-composition, an adversary may corrupt different parties across the various sessions. Consider for instance two different sessions where one of the parties is corrupted in each session. We can view one of these sessions as a "left" session and the other

[1]Note that in the standard simulation paradigm, the output distributions of the "real" and "ideal" worlds must be computationally indistinguishable; in contrast, the definition of Goldreich and Lindell [GL01] allows these distributions to be $\mathcal{O}(1/|D|)$ apart (where D is the password dictionary).

[2]Note that as opposed to a universal constant, we have a constant for every (PPT) adversary

[3]A more general question is to consider the setting where the passwords of honest parties in different sessions might be *correlated in any arbitrary way*. Towards this end, we note that our construction can be easily extended to this setting. However, in this case we require the ideal simulator to be able to query the ideal functionality an *expected constant* number of times per session. Jumping ahead, in case the honest parties were using the same password or fully independent passwords, the simulator is able to "trade" ideal functionality calls in one session for another. Hence, the simulator is able to even out the number of calls to a fixed constant in each session. This in turn means that for the setting of correlated passwords, our construction will satisfy a security definition which is slightly weaker (in that the number of ideal functionality calls are constant only in expectation). Obtaining a construction for correlated (in an arbitrary way) passwords where the number of calls are not just constant in expectation but always bounded by a constant is left as an interesting open question.

as a "right session", while the corrupted parties can be jointly viewed as an adversarial man-in-the-middle. An immediate side-effect of this setting is that it allows an adversary to possibly "maul" a "left" session in order to successfully establish a session key with an honest party (say) P in a "right" session *without* the knowledge of P's secret password. Clearly, in order to provide any security guarantee in such a setting, it is imperative to achieve independence between various protocol sessions executing on the network. Note that this is akin to guaranteeing non-malleability across various sessions in the concurrent setting. Then, as a first step towards solving this problem, we borrow techniques from the construction of concurrent non-malleable zero knowledge argument due to Barak, Prabhakaran and Sahai [BPS06] (BPS-CNMZK). In fact, at a first glance, it might seem that compiling a semi-honest two-party computation protocol (that emulates the PAKE functionality in the stand-alone setting) with the BPS-CNMZK argument or some similar approach might fully resolve this problem. However, such an approach fails on account of several reasons. We highlight some important problems in such an approach.

We first note that the simulation of BPS-CNMZK is based on a rewinding strategy. In a concurrent setting, the adversary is allowed to control the scheduling of the messages of different sessions. Then for a given adversarial scheduling, it is possible that the simulator of BPS-CNMZK may rewind past the beginning of a session (say) s when "simulating" another session. Now, every time session s is re-executed, an adversary may be able to change his input (i.e., make a new password guess possibly based on the auxiliary information it has). In such a case, the simulator would have to query the ideal functionality for that session more than once; therefore, we need to allow the simulator to make extra (i.e., more than one) queries per session to ideal functionality. In order to satisfy our definition, we would need to limit the number of queries to a *constant* per session. However, the simulator for BPS-CNMZK, if used naively, may require large polynomially many queries per session to the ideal functionality, and therefore, fail to satisfy our definition.

In order to overcome this problem, we build on the techniques of precise simulation, introduced by Micali and Pass [MP06] in the context of (stand-alone) zero knowledge and later extended to the setting of concurrent zero knowledge by Pandey, Pass, Sahai, Tseng, and Venkitasubramaniam [PPS$^+$08]. Specifically, Pandey et. al. [PPS$^+$08] use a time-oblivious rewinding schedule that (with a careful choice of system parameters) ensures that the the time spent by the simulator in the "look-ahead" threads[4] is only within a *constant* factor of the time spent by the simulator in the "main" thread. We remark that we do not require this precision in simulation time; instead we require that the number of queries made by the simulator in the look-ahead threads is only within a constant factor of the number of queries made in the main thread. For this purpose, we employ the precise Zero-Knowedlge paradigm of Micali and Pass and consider an imaginary experiment in which our adversary takes a disproportionately large amount of time in generating the message after which the simulator has to query the trusted party. Our rewinding strategy is determined by running the PPSTV [PPS$^+$08] simulator using the next message generation timings of such an (imaginary) adversary (even though our simulator is fully black-box and does not even measure the timings for the real adversary) in order to bound the number of queries.

We further note that in the security proof of the above approach, the simulator must be able to extract the inputs of the adversary in *all* the sessions in order to simulate its view. However, the extractor of [BPS06] is unsuitable for this task since it can extract adversary's inputs (in the setting of BPS-CNMZK) only on a *session-by-session* basis. To further elaborate, let us first recall the setting of BPS-CNMZK, where an adversary is interacting with some honest provers as well as some honest verifiers. Now, in order to extract the input of an adversarial prover in a particular session s, the extractor in [BPS06] honestly runs all the uncorrupted verifiers except the verifier in session s. We stress that the extractor is able to run the honest verifiers by itself since they do not possess any secret inputs; clearly, such an extraction technique would fail in our setting since the simulator does not know the inputs of the honest parties.

To address this problem, we require each party in our protocol to commit to its input and randomness inside a separate preamble [PPS$^+$08, PRS02] that allows extraction of the committed values in a concurrent setting. However, we note that such a preamble requires a complicated rewinding strategy for extraction

[4]Very roughly speaking, a "thread of execution" between the simulator and the adversary is a simulation of a prefix of an actual execution. The simulator may run multiple threads of execution, and finally output a single thread, called the *main thread*. Any other thread is referred to as a *look-ahead thread*. See appendix A.1 for more details.

of committed value, and so is the case for simulating the BPS-CNMZK argument. Indeed, it seems that we might need to *compose* the (possibly conflicting) individual rewinding strategies of BPS-CNMZK and the additional preamble into a new uniform rewinding strategy. Fortunately, by ensuring that we use the same kind of preamble (for committing to the input of a party) as the one used inside BPS-CNMZK, we are able to avoid such a scenario, and crucially, we are able to use the BPS-CNMZK strategy as a single coherent rewinding strategy. The above idea also gives us a *new construction of a concurrent non-malleable zero-knowledge* protocol where the extraction can be *automatically done in-line* along with the simulation. We believe this implication to be of independent interest.

Finally, the construction in [BPS06] is only analyzed for the setting where the theorems to be proven by the honest parties are fixed in advance before any session starts (in keeping with the impossibility results of Lindell [Lin04]). Towards that end, our protocol only makes use of BPS-CNMZK in the very beginning of the protocol to prove a statement which could be generated by the honest parties before the start of any session.

2 Definitions and Preliminaries

2.1 Background

Goldreich and Lindell [GL01] proposed a definition for password authenticated key exchange based on the "simulation paradigm" [Can00]. In particular, they model the problem of PAKE as a three-party functionality $\mathcal{F}$ involving honest parties P_1 and P_2 and an adversary $\mathcal{A}$. They define appropriate "ideal" and "real" models of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model. We give more details below.

Ideal Model. In the ideal model, the parties send their input passwords to a trusted party that evaluates $\mathcal{F}$; if the passwords match, then the trusted party sends a uniformly distributed session key to the parties, else it sends $\perp$. On the other hand, the adversary $\mathcal{A}$ receives no output, and in particular, no information on the password or the session key. However, $\mathcal{A}$ is allowed to control whether or not both the honest parties receive the output (since $\mathcal{A}$ possesses the ability to abort the real execution, see below). The ideal distribution is defined as the output of the honest parties along with the output of the adversary $\mathcal{A}$ resulting from the ideal process.

Real Model. In the real model, parties engage in an execution of a real password-authenticated key exchange protocol. In this model, the adversary $\mathcal{A}$ controls the communication link between the honest parties; as such it is allowed to modify the protocol messages of the honest parties. The real distribution is defined as the output of the honest parties along with the output of the adversary $\mathcal{A}$.

Note that in the real model, $\mathcal{A}$ can attempt to impersonate an honest party by guessing its secret password p and participating in the protocol. Assuming that the passwords are chosen uniformly from a dictionary D, each guess of $\mathcal{A}$ will be correct with probability $1/|D|$. Each guess allows $\mathcal{A}$ to learn some information (whether or not a guess is correct) and since $|D|$ may be small, it is not possible to obtain a protocol that emulates an ideal world execution of F up to computational indistinguishability.

Goldreich and Lindell [GL01] formalize the above limitation in the following manner. They propose a definition where, very informally, the ideal and the real distributions must be distinguishable for any PPT machine at most with probability $\mathcal{O}(1/|D|)$. We refer the reader to [GL01] for more details.

We note that the above definition does not consider the case where the adversary may have some a priori information on the password of the honest parties participating in a session. To this end, we instead consider an improved simulation-based definition that implies the above definition, yet seems more natural and closer to the standard paradigm for defining secure computation. Looking ahead, we note that our security model is similar to the one used by Boyko et al. [BMP00]. Further, as noted in [GL06], the improved definition implies the original definition of [GL01] (see appendix B for a proof sketch). More details are given in the next subsection.

2.2 Our Model

We first summarize the main differences in our model with respect to [GL01]. We first note that even in the stand-alone setting, if an adversary $\mathcal{A}$ controls the communication link between two honest parties, then $\mathcal{A}$ can execute separate "left" and "right" executions with the honest parties. Therefore, these executions can be viewed as two concurrent executions where $\mathcal{A}$ is the common party. In keeping with this observation, in our model, the adversary $\mathcal{A}$ is cast as a party participating in the protocol instead of being a separate entity who controls the communication link (as in [GL01]). We stress that this modeling allows us to assume that the communication between protocol participants takes place over authenticated channels. More details follow.

Description of $\mathcal{F}$. We model the problem of password-authenticated key exchange as a two-party functionality $\mathcal{F}$ involving parties P_1 and P_2 (where either party may be adversarial). If the inputs (password from a dictionary D) of P_1 and P_2 match, then $\mathcal{F}$ sends them a uniformly distributed session key (whose length is determined by the security parameter), else it sends $\perp$.

Further, in contrast to the stand-alone setting of [GL01] (where security holds only if a single protocol session is executed on the network), we consider the more general setting of *concurrent self-composition*, where polynomially many (in the security parameter) protocols with the same password may be executed on the network in an arbitrarily interleaved manner. In this setting, an adversary $\mathcal{A}$ may corrupt several parties across all the different sessions.

To formalize the above requirements and define security, we extend the standard paradigm for defining secure computation. 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* (in the specific sense described below) by an adversary in the ideal model. In particular, we allow the adversary in the ideal world to make a constant number of (output) queries to the trusted party for each protocol session. In the definition below, we focus only on the case where the honest parties hold the same password p. However it can be extended to the case of arbitrarily correlated passwords (or, in fact, general secure computation) in a natural way where the simulator in the ideal world might make an expected constant number of calls to the ideal functionality for every session in the real world.

We consider a static adversary that chooses whom to corrupt before execution of the protocol. Finally, we consider *computational* security only 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 κ. Let D be the dictionary of passwords.

IDEAL MODEL. In the ideal model, there is a trusted party that computes the password functionality $\mathcal{F}$ (described above) based on the inputs handed to it by the players. Let there be n parties $P_1, \ldots, P_n$ where different pairs of parties are involved in one or more sessions, such that the total number of sessions is polynomial in the security parameter κ. Let $M \subset [n]$ denote the subset of corrupted parties controlled by an adversary. An execution in the ideal model with an adversary who controls the parties M proceeds as follows:

I. Inputs: The honest parties hold a fixed input which is a password p chosen from a dictionary D. The input of a corrupted party is not fixed in advance.

II. Session initiation: If a party P_i wishes to initiate a session with another party P_j, it sends a $(\mathsf{start\text{-}session}, i, j)$ message to the trusted party. On receiving a message of the form $(\mathsf{start\text{-}session}, i, j)$, the trusted party sends $(\mathsf{new\text{-}session}, i, j, k)$ to both P_i and P_j, where k is the index of the new session.

III. Honest parties send inputs to trusted party: Upon receiving $(\mathsf{new\text{-}session}, i, j, k)$ 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 k input $x_{i,k}$ to be the password p and sends $(k, x_{i,k})$ to the trusted party.

IV. Corrupted parties send inputs to trusted party: A corrupted party P_i sends a message $(k, x_{i,k})$ to the trusted party, for any $x_{i,k} \in D$ of its choice.

V. Trusted party sends results to adversary: For a session k involving parties P_i and P_j, when the trusted party has received messages $(k, x_{i,k})$ and $(k, x_{j,k})$, it computes the output $\mathcal{F}(x_{i,k}, x_{j,k})$. If at least one of the parties is corrupted, then the trusted party sends $(k, \mathcal{F}(x_{i,k}, x_{j,k}))$ to the adversary[5]. On the other hand, if both P_i and P_j are honest, then the trusted party sends the output message $(k, \mathcal{F}(x_{i,k}, x_{j,k}))$ to them.

VI. Adversary instructs the trusted party to answer honest players: For a session k involving parties P_i and P_j where exactly one party is honest, the adversary, depending on its view up to this point, may send the (output, k) message in which case the trusted party sends the most recently computed session k output $(k, \mathcal{F}(x_{i,k}, x_{j,k}))$ to the honest party. (Intuitively, for each session k where exactly one party is honest, we allow the adversary to choose which one of the λ output values would be received by the honest party.)

VII. Adversary makes more queries for a session: The corrupted party P_i, depending upon its view up to this point, can send the message $(\mathsf{new\text{-}query}, k)$ to the trusted party. In this case, execution of session k in the ideal world comes back to stage IV. P_i can then choose its next input *adaptively* (i.e., based on previous outputs).

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 the subset of corrupted parties $M \subset [n]$. Then the ideal execution of $\mathcal{F}$ (or the ideal distribution) with security parameter κ, password $p \in D$ 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 $\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(\kappa, p, z)$.

REAL MODEL. We now consider the real model in which a real two-party password-based key exchange protocol is executed.

Let $\mathcal{F}, P_1, \ldots, P_n, M$ be as above. Let Σ be the password-based key exchange protocol in question. Let $\mathcal{A}$ be probabilistic polynomial-time (PPT) machine such that for every $i \in M$, the adversary $\mathcal{A}$ controls the party P_i.

In the real model, a polynomial number (in the security parameter κ) of sessions of Σ may be executed concurrently, where the scheduling of all messages throughout the executions is controlled by the adversary. We *do not* assume that all the sessions have a unique session index. We assume that the communication between the parties takes place over authenticated channels[6]. 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 (or the real distribution) with security parameter κ, password $p \in D$ 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 $\text{REAL}^{\Sigma}_{M,\mathcal{A}}(\kappa, p, z)$.

Having defined these models, we now define what is meant by a concurrently-secure password-authenticated key exchange protocol.

Definition 1 *Let $\mathcal{F}$ and Σ be as above. Let D be the dictionary of passwords. Then protocol Σ for computing $\mathcal{F}$ is a concurrently secure password authenticated key exchange protocol if for every probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a probabilistic expected polynomial-time adversary $\mathcal{S}$ such that $\mathcal{S}$ makes a constant number of queries to the ideal functionality per session, and, for every $z \in \{0,1\}^*$, $p \in D$, $M \subset [n]$,*

[5]Note that here, the ideal functionality does not restrict the adversary to a *fixed* constant number of queries per session. However, in our security definition, we will require that the ideal adversary only makes a constant number of queries per session.

[6]As mentioned earlier, this is a reasonable assumption since in our model, the adversary is a protocol participant instead of being a separate entity that controls the communication links (as in [GL01]).

$$\left\{\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(\kappa,p,z)\right\}_{\kappa\in N} \stackrel{c}{\equiv} \left\{\text{REAL}^{\Sigma}_{M,\mathcal{A}}(\kappa,p,z)\right\}_{\kappa\in N}$$

Remark 1. We remark that even if the total number of sessions is such that there are sufficient number of corrupted participants to do brute-force attack and guess the password, the two aforementioned distributions should remain indistinguishable to satisfy our definition above.

Remark 2. Note that in the setting of concurrent self composition, an adversary may be able to maul the conversation (with an honest party) of a particular session in order to successfully establish a session key with an honest party in another session *without* the knowledge of the secret password. Clearly, in order to provide any security guarantee in such a setting, it is imperative to achieve independence between various protocol sessions executing on the network. We note that this property is implicit in our security definition.

We note that our security definition implies the original definition of Goldreich and Lindell [GL01] (adapted to the concurrent setting), as stated below.

Lemma 1 (Informally stated) *Security of a protocol Σ as per Definition 1 implies its security as per the definition of Goldreich and Lindell [GL01].*

We refer the reader to appendix B for a proof of lemma 1. We now state our main result.

Theorem 1 (Main Result) *Assume the existence of non-interactive statistically binding commitments and 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries. Let $\mathcal{F}$ be the two-party PAKE functionality as described above. Then, there exists a protocol Σ that securely realizes $\mathcal{F}$ as per Definition 1.*

We prove the above theorem by constructing such a protocol Σ in section 3. If the underlying primitives are uniform (resp., non-uniform), then the protocol Σ is uniform (resp., non-uniform) as well. A polynomial time adversary against Σ translates to a polynomial time adversary against one of the underlying primitives. Note that non-interactive statistically binding commitments can be constructed out of common type of one way permutations (see the next subsection). Even though we use non-interactive commitments for clarity, our constructions work even if we use a two round statistically binding commitment scheme based on any one way function (and hence the PAKE functionality can be realized assuming only 1-out-of-2 oblivious transfer protocols). Finally we remark that, 1-out-of-2 oblivious transfer (OT) secure against honest but curious adversaries implies 1-out-of-2 OT secure against malicious adversaries [Hai08].

2.3 Building Blocks

We now briefly mention some of the main cryptographic primitives 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 f be a one-way permutation, and H be the hard-core predicate associated with f. Then the commitment to a bit b is computed as $f(x)||H(x)\oplus b$, where x is a random string in the domain of f. The decommitment simply consists of the string x. We denote such a commitment scheme by COM.

Preamble from PPSTV [PPS$^+$08]. A PPSTV preamble is a protocol between a committer and a receiver that consists of two main phases, namely, (a) the commitment phase, and (b) the challenge-response phase. Let k be a parameter that determines the round-complexity of the protocol. Then, in the commit phase, very roughly speaking, the committer commits to a secret string σ and k^2 pairs of its 2-out-of-2 secret shares. The challenge-response phase consists of k iterations, where in each iteration, very roughly speaking, the committer "opens" k shares, one each from k different pairs of secret shares as chosen by the receiver.

The goal of this protocol is to enable the simulator to be able to rewind and extract the "preamble secret" σ with high probability. In the concurrent setting, rewinding can be difficult since one may rewind

past the start of some other protocol [DNS98]. However, as it has been demonstrated in [PPS+08] (see also [PRS02, KP01]) there is a fixed "time-oblivious" rewinding strategy that the simulator can use to extract the preamble secrets from every concurrent cheating committer, with high probability. For our purpose, we will use PPSTV preambles with linear (in the security parameter κ) number of rounds. Then, the simulation strategy in [PPS+08] guarantees a *linear precision* in the running time of the simulator. Specifically, the running time of the simulator is only a constant multiple of the running time of the adversarial committer in the real execution. We refer the reader to appendix A.1 for more details.

Concurrent Non-Malleable Zero Knowledge Argument. We shall use a concurrent non-malleable zero knowledge (CNMZK) argument for every language in **NP** with perfect completeness and negligible soundness error. In particular, we will use a slightly modified version of the CNMZK protocol of Barak, Prabhakaran and Sahai [BPS06], henceforth referred to as mBPS-CNMZK. In the modified version, we replace the PRS [PRS02] preamble used in the original construction with a PPSTV preamble with linear (in the security parameter) number of rounds. We will also require that the non-malleable commitment scheme used in the protocol is public-coin [DDN00]. See appendix A.2 for more details.

Statistically Witness Indistinguishable Arguments. In our construction, we shall use a statistically witness indistinguishable argument (sWI) for proving membership in any **NP** language with perfect completeness and negligible soundness error. Such a scheme can be constructed by using $\omega(\log n)$ copies of Blum's Hamiltonicity protocol [Blu87] in parallel, with the modification that the prover's commitments in the Hamiltonicity protocol are made using a statistically hiding commitment scheme. Statistically hiding commitments were constructed by Naor, Ostrovsky, Venkatesan and Yung [NOVY92] in $O(k/\log(k))$ rounds using a one way permutation ([NOVY92] in turn builds on the *interactive hashing* technique introduced in [OVY91]). Constructions based on one way functions were given in [HNO+09, HRVW09].

Semi-Honest Two Party Computation. We will also use a semi-honest two party computation protocol $\Pi_{\text{SH-PAKE}}$ that emulates the PAKE functionality $\mathcal{F}$ (as described in section 2.2) in the stand-alone setting. The existence of such a protocol $\Pi_{\text{SH-PAKE}}$ follows from [Yao86, GMW87, Kil88].

3 Our Construction

In this section, we describe our two-party protocol Σ that securely realizes the password functionality $\mathcal{F}$ in the setting of concurrent self composition as per Definition 1. Let P_1 and P_2 be two parties with private inputs (password from dictionary D) x_1 and x_2 respectively. Let COM denote a non-interactive statistically binding commitment scheme. By mBPS-CNMZK, we will refer to the modified version of the concurrent non-malleable zero knowledge argument of [BPS06] described in section 2.3. Let $\Pi_{m\text{BPS},P_i \to P_j}$ denote an instance of the mBPS-CNMZK protocol where P_i plays the role of the prover. By sWI, we will refer to a statistically witness indistinguishable argument. Let $\Pi_{\text{SH-PAKE}}$ be any *semi-honest* two party computation protocol that emulates the functionality $\mathcal{F}$ in the stand-alone setting (as per the standard 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 mBPS-CNMZK argument $\Pi_{m\text{BPS},P_1 \to P_2}$ where P_1 proves that com_1 is a commitment to 0.

2. $P_2 \rightarrow P_1$: 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 mBPS-CNMZK argument $\Pi_{m\text{BPS},P_2 \rightarrow P_1}$ 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.

II. mPPSTV Preamble Phase

In this phase, each party P_i engages in the execution of a *modified* PPSTV preamble (henceforth referred to as mPPSTV) with P_j where it commits to its input and randomness. In our modified version of the PPSTV 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. Let $\Pi_{m\text{PPSTV},P_i \rightarrow P_j}$ denote an instance of the mPPSTV 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 $r_1 \stackrel{\$}{\leftarrow} U_\eta$ 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 $\Pi_{\text{SH-PAKE}}$ 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 mPPSTV preamble $\Pi_{m\text{PPSTV},P_1 \rightarrow P_2}$ in the following manner.

   Let k be a polynomial in the security parameter κ. P_1 first prepares $2k^2$ secret shares $\{\alpha^0_{i,j}\}^k_{i,j=1}$, $\{\alpha^1_{i,j}\}^k_{i,j=1}$ 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, $\{A^0_{i,j}\}^k_{i,j=1}$, $\{A^1_{i,j}\}^k_{i,j=1}$. 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 $\hat{\beta}_1$, $\{\hat{\alpha}^0_{i,j}, \hat{\alpha}^1_{i,j}\}^k_{i,j=1}$ 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, $\{A^0_{i,j}\}^k_{i,j=1}$, $\{A^1_{i,j}\}^k_{i,j=1}$ can be decommitted to $\hat{\beta}_1$, $\{\hat{\alpha}^0_{i,j}, \hat{\alpha}^1_{i,j}\}^k_{i,j=1}$, 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. P_1 and P_2 now execute a challenge-response phase. For $j = 1, \ldots, k$:

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

   (b) $P_1 \rightarrow P_2$: Send $\alpha^{z_{1,j}}_{1,j}, \ldots, \alpha^{z_{k,j}}_{k,j}$. Now, P_1 and P_2 engage in the execution of a sWI, where P_1 proves the following statement: either (a) commitments $A^{z_{1,j}}_{1,j}, \ldots, A^{z_{k,j}}_{k,j}$ can be decommitted to $\alpha^{z_{1,j}}_{1,j}, \ldots, \alpha^{z_{k,j}}_{k,j}$ 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 $\Pi_{\text{SH-PAKE}}$. Since $\Pi_{\text{SH-PAKE}}$ 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" $\Pi_{\text{SH-PAKE}}$ 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 $r_2' \xleftarrow{\$} U_\eta$ and sends it to P_2. Define $r_2'' = r_2 \oplus r_2'$.
2. $P_2 \rightarrow P_1$: Similarly, P_2 generates $r_1' \xleftarrow{\$} U_\eta$ 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 $\Pi_{\text{SH-PAKE}}$.

Protocol $\Pi_{\text{SH-PAKE}}$. Let the protocol $\Pi_{\text{SH-PAKE}}$ 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 $\Pi_{\text{SH-PAKE}}$ 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 $\Delta_{1,j} (= \Pi_{\text{SH-PAKE}}(T_{1,j}, x_1, r_1''))$ as per protocol $\Pi_{\text{SH-PAKE}}$. 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., $\Delta_{1,j} (= \Pi_{\text{SH-PAKE}}(T_{1,j}, \hat{x}_1, \hat{r}_1 \oplus r_1'))$, 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.
2. $P_2 \rightarrow P_1$: P_2 now acts symmetrically.

This completes the description of the protocol Σ. Note that Σ consists of several instances of sWI, such that the proof statement for each sWI instance consists of two parts. Specifically, the second part of the statement states that prover committed to bit 1 in the trapdoor creation phase. In the sequel, we will refer to the second part of the proof 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.

4 Proof of Security

Theorem 2 *The proposed protocol Σ is a concurrently secure PAKE protocol as per Definition 1.*

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 m is polynomial in the security parameter κ. Let $\mathcal{A}$ be an adversary who controls an arbitrary number of parties. In order to prove theorem 2, we will first construct a simulator $\mathcal{S}$ that will simulate the view of $\mathcal{A}$ in the ideal world. We will then show that $\mathcal{S}$ makes only a constant number of queries per session while simulating the view of $\mathcal{A}$. Finally, 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 remove this assumption.

We describe the construction of our simulator in section 4.1 and argue the correctness of the simulation in section 4.2 and section 4.3. We first give some notation.

NOTATION. In the sequel, for any session $\ell \in [m]$, we will use the notation H to denote the honest party and $\mathcal{A}$ to denote the corrupted party. Let $\Pi_{m\text{BPS},H \rightarrow \mathcal{A}}$ (resp., $\Pi_{m\text{BPS},\mathcal{A} \rightarrow H}$) denote an instance of mBPS-CNMZK

where H (resp., $\mathcal{A}$) plays the role of the prover and $\mathcal{A}$ (resp., H) plays the verifier. Similarly, let $\Pi_{m\text{PPSTV},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{PPSTV},\mathcal{A}\to H}$) denote an instance of mPPSTV 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.

Consider any session between H and $\mathcal{A}$. Consider the last message from $\mathcal{A}$ before H sends a message to $\mathcal{A}$ during the coin-flipping sub-phase in the secure computation phase. Note that this message could either be the first message of the coin-flipping phase or the last message of the mPPSTV phase, depending upon whether $\mathcal{A}$ or H sends the first message in the coin-flipping phase. In the sequel, we will refer to this message from $\mathcal{A}$ as the special message. Intuitively, this message is important because our simulator will need to query the ideal functionality every time it receives such a message from $\mathcal{A}$. Looking ahead, in order to bound the number of queries made by our simulator, we will be counting the number of special messages sent by $\mathcal{A}$ during the simulation.

4.1 Description of Simulator $\mathcal{S}$

The simulator $\mathcal{S}$ consists of two parts, S_{CEC} and S_{CORE}. Informally speaking, S_{CEC} is essentially the simulator CEC-Sim (see appendix A.1) whose goal is to extract the *preamble secret* in each instance of the PPSTV preamble where $\mathcal{A}$ acts as the committer. These extracted values are passed on to S_{CORE}, who uses them crucially to simulate the view of $\mathcal{A}$. We now give more details.

Description of S_{CEC}. S_{CEC} is essentially the main simulator in that it handles all communication with $\mathcal{A}$. However, for each session $\ell \in [m]$, S_{CEC} by itself only answers $\mathcal{A}$'s messages in those instances of the PPSTV preamble where $\mathcal{A}$ plays the role of the committer; S_{CEC} in turn communicates with the core simulator S_{CORE} to answer all other messages from $\mathcal{A}$.

Specifically, recall that our protocol consists of two instances of the PPSTV preamble where $\mathcal{A}$ plays the role of the committer. Consider any session $\ell \in [m]$. The first instance is inside the mBPS-CNMZK instance $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ in the trapdoor creation phase, while the second instance is in fact the mPPSTV preamble $\Pi^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$ in the second phase. Then, S_{CEC} is essentially the simulator CEC-Sim that interacts with $\mathcal{A}$ in order to extract the preamble secret in each of the above instances of the PPSTV preamble. Specifically, in order to perform these extractions, S_{CEC} employs the time-oblivious rewinding strategy of CEC-Sim for an imaginary adversary (see next paragraph). During the simulation, whenever S_{CEC} receives a message from $\mathcal{A}$ in any of the above instance of the PPSTV preamble, then it answers it on its own in the same manner as CEC-Sim does (i.e., by sending a random challenge string). However, on receiving any other message, it simply passes it to the core simulator S_{CORE} (described below), and transfers its response to $\mathcal{A}$. Whenever S_{CEC} extracts a preamble secret from $\mathcal{A}$ at any point during the simulation, it immediately passes it to S_{CORE}. If S_{CEC} fails to extract any of the preamble secrets from $\mathcal{A}$, then it outputs the abort symbol $\perp$.

Message generation timings of $\mathcal{A}$. We note that in order to employ the time-oblivious rewinding strategy of CEC-Sim, S_{CEC} needs to know the amount of time that $\mathcal{A}$ takes to send each message in the protocol (see [PPS+08]). We remark that we do not seek precision in simulation time (guaranteed by the rewinding strategy of CEC-Sim); instead we only require that the number of queries made by the simulator in the look-ahead threads is only within a constant factor of the number of the number of sessions. To this end, we consider an imaginary experiment in which $\mathcal{A}$ takes a disproportionately large amount of time in generating the message after which our simulator has to query the trusted party. Then the rewinding strategy of S_{CEC} is determined by running CEC-Sim using the next message generation timings of such an (imaginary) adversary, explained as follows.

Consider all the messages sent by $\mathcal{A}$ during a protocol execution. We will assign q time units to the special message, where q is the round complexity (linear in the security parameter) of our protocol; any other message from $\mathcal{A}$ is simply assigned one time unit. Intuitively, by assigning more weight to the special message, we ensure that if the running time of our simulator is only within a constant factor of the running time of $\mathcal{A}$ in the real execution, then the number of special messages sent by $\mathcal{A}$ during the simulation must be a constant as well. Looking ahead, this in turn will allow us to prove that the number of queries made

by the simulator are only a constant.

Description of S_{CORE}. We describe the strategy of S_{CORE} in each phase of the protocol, for each session $\ell \in [m]$. We stress that S_{CORE} uses the same strategy in the main-thread as well as all look-ahead threads (unless mentioned otherwise).

Trapdoor Creation Phase. S_{CORE} first sends a commitment to bit 1, instead of committing to bit 0. Now, recall that S_{CEC} interacts with $\mathcal{A}$ during the preamble phase in $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ and extracts the preamble secret $\sigma^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ from $\mathcal{A}$ at the conclusion of the preamble. Then, on receiving $\sigma^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ from S_{CEC}, S_{CORE} simulates the *post-preamble* phase of $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ (see section A.2 for a description of the mBPS-CNMZK protocol) in the following manner[7].

Let y^{ℓ} be the proof statement in $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$. Then, in phase II of $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$, S_{CORE} creates a statistically hiding commitment (sCOM) to $\sigma^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ (instead of a string of all zeros) and follows it up with an honest execution of statistical zero knowledge argument of knowledge (sZKAOK) to prove knowledge of the decommitment. In phase IV of $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$, S_{CORE} creates a non-malleable commitment (NMCOM) to an all zeros string (instead of a valid witness to y^{ℓ}). Finally, in phase V, S_{CORE} proves the following statement using sZKAOK: (a) the value committed to in phase IV is a valid witness to y^{ℓ}, or (b) the value committed to in phase II is $\sigma^{\ell}_{m\text{BPS},H\to\mathcal{A}}$. Here it uses the witness corresponding to the *second* part of the statement.

Now, consider the mBPS-CNMZK instance $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$, where H plays the role of the verifier. Here, S_{CORE} simply uses the honest verifier strategy to interact with $\mathcal{A}$.

mPPSTV Preamble Phase. Consider the execution of the mPPSTV instance $\Pi^{\ell}_{m\text{PPSTV},H\to\mathcal{A}}$. Here, S_{CORE} commits to a random string and answers $\mathcal{A}$'s challenges with random strings. Note that the trapdoor condition is true for each instance of sWI in $\Pi^{\ell}_{m\text{PPSTV},H\to\mathcal{A}}$ since S_{CORE} committed to bit 1 (instead of 0) in the trapdoor creation phase. Therefore, S_{CORE} uses the trapdoor witness in order to successfully simulate each instance of sWI in $\Pi^{\ell}_{m\text{PPSTV},H\to\mathcal{A}}$.

Now consider the mPPSTV instance $\Pi^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$. Note that in this preamble, S_{CEC} interacts with $\mathcal{A}$ without the help of S_{CORE}. As explained earlier, S_{CEC} extracts the preamble secret (that contains the input and randomness of $\mathcal{A}$ in session ℓ) and passes it to S_{CORE}.

Secure Computation Phase. Let $S_{\Pi_{\text{SH-PAKE}}}$ denote the simulator for the semi-honest two-party protocol $\Pi_{\text{SH-PAKE}}$ used in our construction. S_{CORE} internally runs the simulator $S_{\Pi_{\text{SH-PAKE}}}$ on adversary's input in session ℓ. $S_{\Pi_{\text{SH-PAKE}}}$ starts executing, and, at some point, it makes a call to the trusted party in the ideal world with some input (say) x. S_{CORE} uses the following strategy to manage queries to the trusted party.

S_{CORE} maintains a counter c to count the total number of queries (including all sessions) made to the trusted party on the look-ahead threads so far in the simulation (note that there will be exactly m queries on the main thread). Now, when $S_{\Pi_{\text{SH-PAKE}}}$ makes a call to the trusted party, S_{CORE} computes a session index s in the following manner. If the query corresponds to the main thread, then S_{CORE} sets $s = \ell$, else it computes $s = c \mod m$. Now, if S_{CORE} has already queried the trusted party at least once for session s, then it first sends the $(\mathsf{new\text{-}query}, s)$ message to the trusted party. Otherwise, it simply sends the message (s, x) to the trusted party.[8][9] The response from the trusted party is passed on to $S_{\Pi_{\text{SH-PAKE}}}$. If the query corresponds to the main thread, S_{CORE} sends the message (output, s) to the trusted party, indicating it to send the output to the honest party in session s.[10]

[7] Note that given the preamble secret, the post-preamble phase of mBPS-CNMZK can be simulated in a *straight-line* manner. See section A.2 for more details.

[8] We stress that the simulator is able to "trade" the ideal functionality calls in one session for another since the inputs of the honest parties are the same across all the sessions.

[9] Note that by choosing the session index for the output query in the above fashion, S_{CORE} is able to equally distribute the queries across all the sessions. Looking ahead, in the next subsection, we will argue that the total number of queries across all the sessions are only within a constant factor of the number of sessions. Then, this strategy of distributing the queries will ensure that the queries per session are also a constant.

[10] Note that $s = \ell$ in this case. We stress that by setting $s = \ell$ for a query on the main thread, S_{CORE} ensures that the honest party in session ℓ receives the correct output. (Note that an honest party does not receive any output for an output query on a

Having received the trusted party's response from S_{CORE}, $S_{\Pi_{\text{SH-PAKE}}}$ runs further, and finally 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 $\Pi_{\text{SH-PAKE}}$, 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, S_{CORE} 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 $\Pi_{\text{SH-PAKE}}$, S_{CORE} sends $\tilde{r}^{\ell}_{\mathcal{A}}$ to $\mathcal{A}$ during the coin-flipping phase prior to the execution of $\Pi_{\text{SH-PAKE}}$. Finally, S_{CORE} 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 $\Pi_{\text{SH-PAKE}}$. This is done as follows. Without loss of generality, let us assume that the honest party sends the first message in this instance of $\Pi_{\text{SH-PAKE}}$. Then, in round j, $1 \leq j \leq t$, S_{CORE} 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). S_{CORE} uses the trapdoor witness to complete the associated sWI. As we establish later in the proof of Lemma 3, 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 our simulator $\mathcal{S} = \{S_{\text{CEC}}, S_{\text{CORE}}\}$. In the next subsection, we bound the total number of queries made by $\mathcal{S}$.

4.2 Total Queries by $\mathcal{S}$

Lemma 2 *Let m be the total number of sessions of Σ being executed concurrently. Then, the total number of queries made by $\mathcal{S}$ to the trusted party is within a constant factor of m.*

Proof. Let T be the total running time of the adversary in the real execution, as per the time assignment strategy described in section 4.1. Now, since $\mathcal{S}$ employs the time-oblivious rewinding strategy of CEC-Sim, it follows from lemma 5 (see section A.1) that the total running time of $\mathcal{S}$ is within a constant factor of T. Let us now assume that our claim is false, i.e., the total number of queries made by $\mathcal{S}$ is a super-constant multiple of m. We will show that in this case, the running time of $\mathcal{S}$ must be super-constant multiple of T, which is a contradiction. We now give more details.

Let q be the round complexity of Σ. Then, as per the time assignment strategy given in section 4.1, $T = (q - 1 + q) \cdot m$ (recall that the special message is assigned a weight of q time units, while each of the remaining $q - 1$ messages is assigned one time unit). Now, let λ be a value that is super-constant in the security parameter such that $\mathcal{S}$ makes $\lambda \cdot m$ total queries during the simulation. Note that each output query corresponds to a unique special message. Let T' be the total running time of $\mathcal{S}$. We calculate T' as follows:

$$\begin{aligned} T' &\geq q \cdot (\lambda \cdot m) + (q-1) \cdot m \\ &> q \cdot (\lambda \cdot m) \\ &> \frac{\lambda \cdot q}{(q-1+q)} \cdot (q-1+q) \cdot m \\ &> \frac{\lambda \cdot q}{(q-1+q)} \cdot T \end{aligned}$$

Since $\frac{\lambda \cdot q}{(q-1+q)}$ is a super-constant in the security parameter, we have that T' is a super-constant multiple of T, which is a contradiction. Hence the claim follows. ∎

The corollary below immediately follows from lemma 2 and the description of $\mathcal{S}$ in section 4.1.

Corollary 1 *$\mathcal{S}$ makes a constant number of queries per session to the trusted party.*

4.3 Indistinguishability of the Outputs

We consider two experiments H_0 and H_1, where H_0 corresponds to the real execution of Σ while H_1 corresponds to the ideal computation of $\mathcal{F}$, as described below.

look-ahead thread.)

Experiment H_0: The simulator $\mathcal{S}$ is given the inputs of all the honest parties. By running honest programs for the honest parties, it generates their outputs along with $\mathcal{A}$'s view. The simulation is perfect.

Experiment H_1: $\mathcal{S}$ simulates all the sessions without the inputs of the honest parties (in the same manner as explained in the description of $\mathcal{S}$) and outputs the view of $\mathcal{A}$. Each honest party outputs the response it receives from the trusted party.

Let v^i be a random variable that represents the output (including the view of the adversary and the outputs of the honest parties) of H_i. We now claim that the output distributions of H_0 and H_1 are indistinguishable, as stated below:

Lemma 3 $v^0 \stackrel{c}{\equiv} v^1$

This is the main technical lemma in our paper. The proof of this lemma requires a careful hybrid argument. A detailed and self contained proof is given in the next subsection.

5 Proof of Lemma 3

We will prove this lemma using a carefully designed series of intermediate hybrid experiments. 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 $\Pi_{m\text{PPSTV},H\to\mathcal{A}}$ (resp., $\Pi_{m\text{PPSTV},\mathcal{A}\to H}$) denote the instance of mPPSTV 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 mBPS-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 $\Pi_{\text{SH-PAKE}}$ 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 4.3). Hybrid $\mathcal{H}_{2m:6}$ will be the ideal execution (i.e., H_1, as defined in section 4.3). 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 mBPS-CNMZK instance $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ in session ℓ. Let y^{ℓ} denote the proof statement for this mBPS-CNMZK instance[11]. 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 $v^{i:j}$.

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

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 4 *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 mBPS-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.

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

5.1 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 PPSTV simulator CEC-Sim (see appendix A.1). 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*[12].

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 $v^{i:1}$.

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

We now claim that,

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

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

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:

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

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

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

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 PPSTV preamble inside the mBPS-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 mPPSTV preamble $\Pi^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$. 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 6 holds for such an adversary (i.e. the probability that the 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 6 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 $\Pi^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$ if $\Pi^{\ell}_{m\text{PPSTV},\mathcal{A}\to H}$ 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,

$$v^{i:1} \stackrel{c}{\equiv} v^{i:2} \tag{5}$$

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

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:

$$v^{i:1} \stackrel{c}{\equiv} v^{i:1:1} \tag{7}$$

$$\forall \ell \quad \alpha^{i:1,\ell}_{\mathcal{A}} \stackrel{c}{\equiv} \alpha^{i:1:1,\ell}_{\mathcal{A}} \tag{8}$$

$$v^{i:1:1} \stackrel{c}{\equiv} v^{i:1:2} \tag{9}$$

$$\forall \ell \quad \alpha^{i:1:1,\ell}_{\mathcal{A}} \stackrel{c}{\equiv} \alpha^{i:1:2,\ell}_{\mathcal{A}} \tag{10}$$

$$v^{i:1:2} \stackrel{c}{\equiv} v^{i:1:3} \tag{11}$$

$$\forall \ell \quad \alpha^{i:1:2,\ell}_{\mathcal{A}} \stackrel{c}{\equiv} \alpha^{i:1:3,\ell}_{\mathcal{A}} \tag{12}$$

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 $\alpha^{i:1,\ell}_{\mathcal{A}}$ and $\alpha^{i:1:1,\ell}_{\mathcal{A}}$ 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 $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, M_ℓ takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$. Additionally, M_ℓ "exposes" the NMCOM in $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ 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 $\sigma^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$ to C. Now, when C starts the execution of $\overline{\text{sCOM}}$ in $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, 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 $\sigma^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, 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 $\Pi^{s(i)}_{m\text{BPS},H\to\mathcal{A}}$, 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 $\alpha^{i:1:2,\ell}_{\mathcal{A}}$ and $\alpha^{i:1:3,\ell}_{\mathcal{A}}$ 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 $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ 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 $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$ denote this commitment.

We now claim that,

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

$$\forall \ell \quad \alpha^{i:2,\ell}_{\mathcal{A}} \stackrel{c}{\equiv} \alpha^{i:3,\ell}_{\mathcal{A}} \tag{14}$$

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 mBPS-CNMZK. Let us first assume that equation 14 is false, i.e., $\exists \ell \in [m]$ such that $\alpha^{i:2,\ell}_{\mathcal{A}}$ and $\alpha^{i:3,\ell}_{\mathcal{A}}$ are distinguishable by a PPT distinguisher. Note that it cannot be the case that the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ concludes *before* $\mathcal{S}$ sends the non-interactive commitment $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$ in session $s(i)$, since in this case, the execution of NMCOM is independent of $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$. Now consider the case when the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ concludes *after* $\mathcal{S}$ sends $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$.

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 $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$, it takes this commitment from an external sender C and forwards it internally to $\mathcal{A}$. Additionally, it "exposes" the NMCOM inside $\Pi^{\ell}_{m\text{BPS},\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Note that if C 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,

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

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i:3,\ell} \stackrel{c}{\equiv} \alpha_{\mathcal{A}}^{i:4,\ell} \tag{16}$$

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 $\Pi^{s(i)}_{m\text{PPSTV},H\rightarrow\mathcal{A}}$ 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,

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

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i:4,\ell} \stackrel{c}{\equiv} \alpha_{\mathcal{A}}^{i:5,\ell} \tag{18}$$

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:

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

$$\forall \ell \quad \alpha_{\mathcal{A}}^{i:4,\ell} \stackrel{c}{\equiv} \alpha_{\mathcal{A}}^{i:4:1,\ell} \tag{20}$$

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

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

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 $\Pi^{s(i)}_{m\text{PPSTV},H\rightarrow\mathcal{A}}$. In this case, we can leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in mBPS-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 $\Pi^{s(i)}_{m\text{PPSTV},H\to\mathcal{A}}$.

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 $\Pi_{\text{SH-PAKE}}$ in session $s(i)$, in the following manner. Let $S_{\Pi_{\text{SH-PAKE}}}$ be the simulator for the semi-honest two party protocol $\Pi_{\text{SH-PAKE}}$ used in our construction. $\mathcal{S}$ internally runs the simulator $S_{\Pi_{\text{SH-PAKE}}}$ for the semi-honest two party protocol $\Pi_{\text{SH-PAKE}}$ on $\mathcal{A}$'s input in session $s(i)$ that was extracted earlier. When $S_{\Pi_{\text{SH-PAKE}}}$ 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 4.1. The response from the trusted party is passed on to $S_{\Pi_{\text{SH-PAKE}}}$. Further, $\mathcal{S}$ decides whether the output must be sent to the honest party in the same manner as explained earlier. $S_{\Pi_{\text{SH-PAKE}}}$ finally halts and outputs a transcript of the execution of $\Pi_{\text{SH-PAKE}}$, 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 4.1. We claim that during the execution of $\Pi_{\text{SH-PAKE}}$, 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:

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

$$\forall \ell \quad \alpha^{i:5,\ell}_{\mathcal{A}} \stackrel{c}{\equiv} \alpha^{i:6,\ell}_{\mathcal{A}} \tag{24}$$

Proving Equation 23. Informally speaking, equation 23 follows from the semi-honest security of the two-party computation protocol $\Pi_{\text{SH-PAKE}}$ 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 $\Pi_{\text{SH-PAKE}}$ 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 $x^{s(i)}_{\mathcal{A}}$ denote the extracted input of $\mathcal{A}$. Let $x^{s(i)}_{H}$ 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 $x^{s(i)}_{H}$ along with $x^{s(i)}_{\mathcal{A}}$ and K to C and receives from C a transcript for $\Pi_{\text{SH-PAKE}}$ 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 - $x^{s(i)}_{H}$ and $x^{s(i)}_{\mathcal{A}}$ - along with the output K. In this case, the transcript output by C is a real transcript of an honest execution of $\Pi_{\text{SH-PAKE}}$.
2. C computed the transcript and randomness by using only adversary's input $x^{s(i)}_{\mathcal{A}}$, and the output K. In this case C simply ran the simulator $S_{\Pi_{\text{SH-PAKE}}}$ on input $x^{s(i)}_{\mathcal{A}}$ and answered its query with K. The transcript output by C in this case is a simulated transcript for $\Pi_{\text{SH-PAKE}}$.

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 $\Pi_{\text{SH-PAKE}}$, 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 $\Pi_{\text{SH-PAKE}}$ and the extractability property of the non-malleable commitment scheme in mBPS-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 $\Pi_{\text{SH-PAKE}}$ 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 $\Pi_{\text{SH-PAKE}}$ 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 $\Pi_{\text{SH-PAKE}}$, which is a contradiction.

paragraphAcknowledgements We thank Rafael Pass for bringing to our attention that an argument in an earlier draft of this paper was insufficient. We also thank Omkant Pandey and Akshay Wadia for useful discussions.

References

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

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

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Security and Privacy*, 1992.

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT*, pages 156–171, 2000.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.

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

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[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.

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

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

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[Gen08] Rosario Genarro. Faster and shorter password-authenticated key exchange. In *ACM Conference on Computer and Communications Security*, 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.

[GL06] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. *J. Cryptology*, 19(3):241–340, 2006.

[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.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 54–71. Springer, 2009.

[Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2008.

[HNO+09] Iftach Haitner, Minh-Huyen Nguyen, Shien Jin Ong, Omer Reingold, and Salil P. Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM J. Comput.*, 39(3):1153–1218, 2009.

[HRVW09] Iftach Haitner, Omer Reingold, Salil P. Vadhan, and Hoeteck Wee. Inaccessible entropy. In Michael Mitzenmacher, editor, *STOC*, pages 611–620. ACM, 2009.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

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

[KOY02] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Forward secrecy in password-only key exchange protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 29–44. Springer, 2002.

[KOY09] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient and secure authenticated key exchange using weak passwords. *J. ACM*, 57(1), 2009.

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

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC*, pages 306–315, 2006.

[NOVY92] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for np can be based on general complexity assumptions. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 196–214. Springer, 1992.

[NV04] Minh-Huyen Nguyen and Salil P. Vadhan. Simpler session-key generation from short random passwords. In *TCC*, pages 428–445, 2004.

[OVY91] Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Fair games against an all-powerful adversary. In *DIMACS workshop presentation, 1990. Extended abstract in proceedings of Sequences II, June 1991, Positano, Italy, R.M. Capocelli, A. De-Santis and U. Vaccaro (Eds.), Springer-Verlag.*, Journal version in AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 13. (Jin-Yi Cai ed.) pp. 155-169, 1991.

[PPS+08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Precise concurrent zero knowledge. In *EUROCRYPT*, pages 397–414, 2008.

[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.

[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 cryptographic primitives that we use in our construction.

A.1 Preamble from PPSTV [PPS+08]

In this subsection, we describe the preamble from [PPS+08] some of its main properties useful for our context. Let k be a parameter that determines the round-complexity of the protocol. Let COM be a statistically binding commitment scheme. The PPSTV preamble 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 k^2 pairs of secret shares $\{\alpha^0_{i,\ell}, \alpha^1_{i,\ell}\}^k_{i,\ell=1}$ 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 k^2 commitments to the receiver.

Challenge-Response Phase This phase consists of k iterations where in the ℓ^{th} iteration, the receiver sends a random k-bit string $b_\ell = b_{1,\ell}, \ldots, b_{k,\ell}$, and the committer decommits to $\mathrm{COM}(\alpha^{b_{1,\ell}}_{1,\ell}), \ldots, \mathrm{COM}(\alpha^{b_{k,\ell}}_{k,\ell})$. 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.

Now consider the scenario where multiple sessions of the PPSTV preamble are being executed concurrently honest receivers and a cheating committer. The simulator for the PPSTV preamble is a program that uses a rewinding schedule to "simulate" the concurrent sessions (i.e., produce a transcript indistinguishable from the real execution) and simultaneously extract the committed value β (referred to as the **preamble secret**) in each session with high probability. In the concurrent setting, such an extraction can be difficult since while rewinding for a specific session, a simulator may rewind past the start of another session [DNS98]. However, as it has been demonstrated in [PPS+08]), there is a fixed *time-oblivious* rewinding strategy that the simulator can use to extract the preamble secret for every concurrent cheating committer with high probability. We now give more details.

Simulator CEC-Sim. We call the simulator for the PPSTV preamble CEC-Sim, where CEC stands for concurrently-extractable commitment (intuitively, the PPSTV preamble can be viewed as a concurrently-extractable commitment). It was shown in [PPS+08] that there exists a simulator CEC-Sim that uses a time-oblivious rewinding strategy such that when the preamble contains a *linear* (in the security parameter) number of rounds, CEC-Sim is able to simulate the concurrent sessions (and extract the preamble secret in

[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 PPSTV commitment, and it is this value that the receiver accepted as the committer's secret in the preamble.

each session) in time that is only within a constant factor of the running time of the concurrent committer. In this paper, we do not focus on precision in running time of the simulator. However, we shall crucially use the precision in running time of the simulator CEC-Sim in order to argue that the total number of output queries made by our simulator (that internally uses CEC-Sim) are only a constant per session. Below we introduce some terminology and summarize two main properties of CEC-Sim for our context.

Consider polynomially many concurrent sessions of the PPSTV 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. Furthermore, due to the time-oblivious rewinding strategy, the running time of CEC-Sim is only within a constant factor of the running time of the adversarial committer. We recall the following two properties of CEC-Sim that are useful to our context.

Lemma 5 *(Informal statement [PPS$^+$08]) For any concurrent adversarial committer, there exists a simulator algorithm CEC-Sim such that the running time of CEC-Sim is within a constant factor of T, where T is the running time of the adversarial committer.*

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 6 *(implicit in [PPS$^+$08]) Consider a concurrent adversarial committer and a receiver running polynomially many (in the security parameter) sessions of a protocol with the PPSTV preamble. Then except with negligible probability, in every thread of execution output by CEC-Sim; if the receiver accepts a PPSTV 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 PPSTV preamble. In our construction, we shall additionally make use of a *modified* version of the PPSTV preamble (referred to as mPPSTV) 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 6 is applicable to the mPPSTV 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 6.

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 minor 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 mBPS-CNMZK).

At a high level, the mBPS-CNMZK protocol consists of two main phases - (a) a *preamble phase*, where the verifier commits to a random secret (say) σ using a PPSTV [PPS+08] 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 mBPS-CNMZK protocol proceeds as follows.

Phase I. P and V engage in an execution of the PPSTV preamble[14] with linear (in the security parameter) number of rounds, 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] [DDN00].

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.

B Our Definition Implies the Definition of [GL01]

Goldreich and Lindell [GL01] model the problem of PAKE as a three-party functionality $\mathcal{F}$ involving honest parties P_1 and P_2 and an adversary $\mathcal{A}$. They define appropriate "ideal" and "real" models of computation, and require that any adversary in the real model can be *emulated* (in a specific sense described below) by an adversary in the ideal model. We refer the reader to section 2.1 for a brief description of the ideal and real models.

Goldreich and Lindell define the stand-alone security of a PAKE protocol by requiring the ideal and real distributions to be at most $\mathcal{O}(1/|D|) + \mu(\kappa)$ apart, where D is the dictionary of passwords and μ is

[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 non-malleable commitment protocol of [DDN00] only consists of standard perfectly binding commitments and zero knowledge proof of knowledge. Therefore, we can easily instantiate the DDN construction with public-coin versions of these primitives such that the resultant protocol is public-coin.

a negligible function in the security parameter κ. Further, it was noted in [GL01] that in the case of m sequential sessions (referring to the same password), the former definition can be suitably modified to allow a distinguishing gap of $\mathcal{O}(m/|D|) + \mu(\kappa)$ rather than $\mathcal{O}(1/|D|) + \mu(\kappa)$. We note that this new definition works even for the case of m *concurrent* sessions (referring to the same password). We restate this definition below (assuming suitable definitions of the ideal and real models for the case when m sessions are being executed concurrently).

Definition 2 (adapted from [GL01]) *Let $\mathcal{F}$ be as above. Let D be the dictionary of passwords. A protocol Σ for password-authenticated key exchange is concurrently secure if for every probabilistic polynomial-time real model adversary A, there exists a probabilistic polynomial time ideal model adversary S such that for every password $p \in D$, and every auxiliary input z,*

$$\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(p, z) \overset{\mathcal{O}(\frac{m}{|D|})}{\equiv} \text{REAL}_{\mathcal{A}}^{\Sigma}(p, z),$$

where $\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(p, z)$ and $\text{REAL}_{\mathcal{A}}^{\Sigma}(p, z)$ are the output distributions in the ideal and real worlds respectively.

We stress that definition 2 is meaningful only if the adversary has no a priori information on the password. That is, the auxiliary input z in the above definition must not contain any information on the password. We now claim that definition 1 (given in section 2.2) implies definition 2, as stated below.

Lemma 7 *If a PAKE protocol is concurrently secure as per definition 1, then it is also secure as per definition 2.*

Before we give a proof of lemma 7, we first make the following observations. The definition of Goldreich and Lindell cannot be satisfied if an adversary has a priori information on the password; in particular, the real and ideal distributions may be distinguishable with probability 1 in this case. In contrast, our definition (see definition 1) can still be realized for such an adversary (and provides meaningful guarantees even for such a case), as evident in theorem 2. Therefore, in order to prove lemma 7, we will consider weaker adversaries for our definition; in particular, we will only consider adversaries that have no a-priori information on the password[16]. We now give a proof sketch.

Proof of Lemma 7. Let Σ be a PAKE protocol that is concurrently secure as per definition 1. Let $m = \text{poly}(\kappa)$ be the total number of sessions. Then, given a real world adversary $\mathcal{A}$ for Σ, there exists an ideal world adversary $\mathcal{S}$ such that $\mathcal{S}$ makes a constant number of queries per session, and produces an ideal distribution that is computationally indistinguishable from the real distribution. We will use $\mathcal{S}$ to construct another ideal world adversary $\mathcal{S}'$ that makes no queries and produces an ideal distribution that is $\mathcal{O}(m/|D|) + \mu(\kappa)$ apart from the real distribution. We note that this is sufficient to prove lemma 7. We stress that here we are only considering adversaries that have no a-priori information on the password.

Description of $\mathcal{S}'$. The ideal world adversary $\mathcal{S}'$ works by running $\mathcal{S}$. Whenever $\mathcal{S}$ makes any query in the ideal world, $\mathcal{S}'$ returns $\perp$ (i.e., $\mathcal{S}'$ replies that the password is incorrect). Finally, when $\mathcal{S}$ stops and outputs a value (its view), $\mathcal{S}'$ outputs the same value.

Let λ be a constant such that $\mathcal{S}$ makes a total of $m \cdot \lambda$ queries. Let E_i denote the event that the answer to the ith query of $\mathcal{S}$ is wrong. In other words, E_i is the event that the password guessed by $\mathcal{S}$ in the ith query is correct. Then, since $\mathcal{S}$ has no prior information on the password, $\Pr[E_i] = \frac{1}{|D|}$ (where probability is over the random coins of $\mathcal{S}$). We can use the union bound to compute an upper bound on the probability that at least one of the total $m \cdot \lambda$ answers is wrong. Specifically, we have,

$$\Pr[E_1 + \ldots, +E_{m \cdot \lambda}] \leq \frac{m \cdot \lambda}{|D|}.$$

[16] Note that these are the only valid adversaries as per definition of [GL01].

Therefore, all the answers of $\mathcal{S}'$ must be correct with probability at least $1 - \frac{m \cdot \lambda}{|D|}$.

Now, from definition 1, the ideal distribution produced by $\mathcal{S}$ must be computationally indistinguishable from the real distribution conditioned on the event that all the answers of $\mathcal{S}'$ are correct. Then, it follows that the distinguishing gap between these distributions is at most $\mathcal{O}(m/|D|) + \mu(\kappa)$, where μ is a negligible function in the security parameter κ. ■