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

Vipul Goyal[1], Abhishek Jain[2⋆], and Rafail Ostrovsky[3⋆⋆]

[1] Microsoft Research, India, vipul@microsoft.com
[2] UCLA, abhishek@cs.ucla.edu
[3] UCLA, rafail@cs.ucla.edu

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.

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) [1, 2] do not immediately translate to this setting; the reason being that known solutions for secure MPC require the existence of authenticated channels – which

⋆ Supported in Part by NSF grants 0830803, 0916574.
⋆⋆ 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..

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 [3]. This was followed by several additional works proposing protocols with only heuristic security arguments (see [4] 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 [5] and Boyko et al [6] 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 [7] gave the first construction for PAKE without random oracles based on the DDH assumption. Their result were subsequently improved by Gennaro and Lindell [8], and Genarro [9]. Again in the CRS model, Canetti, Halevi, Katz, Lindell and MacKenzie [10] proposed new definitions and constructions for a PAKE protocol in the framework of Universal Composability [11]. They further proved the *impossibility* of a Universally Composable PAKE construction in the plain model.

Goldreich and Lindell [12] 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 [13]. Recently, Barak et al [14] 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, [12, 13] and [14] remain the only known solutions for PAKE in the plain model. However, an important limitation of Goldreich and Lindell [12] (as well as Nguyen and Vadhan [13]) 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 [15]. 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 [14] 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 [7, 16, 8, 10, 4] solve this problem (where [7, 8] are secure under self-composition, and [16] also enjoy forward secrecy, while [10] 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 [17] and Micali and Pass [18].

Our construction is proven secure as per the definition of Goldreich and Lindell [12] in the concurrent setting. We stress that Lindell's impossibility result [19] 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[4], 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 [12]. The definition in [12], 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 [6]. 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). 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 [20], this improved definition implies the original definition of Goldreich and Lindell (see full version for a proof).

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[5]. An example of the same password case is

[4] 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 [12] allows these distributions to be $\mathcal{O}(1/|D|)$ apart (where D is the password dictionary).

[5] 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

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 [20]). 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 [21]. 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 [21] 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 [21]. Secondly, [21] 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 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 [17] (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 schedul-

constant in expectation but always bounded by a constant is left as an interesting open question.

ing 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 [18] 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 [22]. Specifically, Pandey et. al. [22] 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[6] 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 [22] 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 [17] 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 [17] 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

[6] 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*.

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 [22, 23] 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 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 [17] 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 [19]). 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 Our Model

We first summarize the main differences in our model with respect to [12]. 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 [12], see full version for more details). We stress that this modeling allows us to assume that the communication between protocol participants takes place over authenticated channels. Furthermore, in contrast to [12], we allow the adversary to have a-priori information on the password. 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 [12] (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 simulation paradigm for defining secure computation. 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 the static corruption model and probabilistic polynomial time (PPT) adversaries only. 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[7]. 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.

[7] 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.

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[8]. 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 $\text{REAL}^{\Sigma}_{M,\mathcal{A}}(\kappa, p, z)$.

We now give our definition of concurrently-secure password-authenticated key exchange protocol.

[8] 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 [12]).

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 $z \in \{0,1\}^*$, $p \in D$, $M \subset [n]$,*

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

We note that our security definition implies the original definition of Goldreich and Lindell [12] (adapted to the concurrent setting). We refer the reader to the full version for a formal proof. We now state our main result.

Theorem 1 (Main Result) *Assume the existence of 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries*[9]*. 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.

2.2 Building Blocks

We now briefly mention some of the main cryptographic primitives that we use in our construction. We refer the the reader to the full version of the paper for more details.

Statistically Binding Commitments. In our protocol, we shall use the 2-round statistically binding commitment scheme of Naor [25] based on one-way functions. Given a random string z from the receiver, let $\text{COM}_z(\cdot)$ denote the commitment function of the scheme.

Preamble from PPSTV [22]. 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

[9] Note that 1-out-of-2 oblivious transfer (OT) secure against honest but curious adversaries implies 1-out-of-2 OT secure against malicious adversaries [24].

other protocol [26]. However, as it has been demonstrated in [22] (see also [23, 27]) there is a "time-oblivious" rewinding strategy that the simulator can use to extract the preamble secrets from every concurrent cheating committer, with high probability. In the sequel, we will refer to the preamble simulator as *CEC-Sim*. For our purpose, we will use PPSTV preambles with linear (in the security parameter κ) number of rounds. Then, the simulation strategy in [22] 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.

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 [17], henceforth referred to as mBPS-CNMZK. In the modified version, we replace the PRS [23] 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 [28].

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 [29] 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 [30] in $O(k/\log(k))$ rounds using a one way permutation ([30] in turn builds on the *interactive hashing* technique introduced in [31]). Constructions based on one way functions were given in [32, 33].

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 2.1) in the stand-alone setting as per the standard definition of secure computation. The existence of such a protocol $\Pi_{\text{SH-PAKE}}$ follows from [1, 34].

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.

Notation. Let P_1 and P_2 be two parties with private inputs (password from dictionary D) x_1 and x_2 respectively. Given a random string z (from the receiver), let $\text{COM}_z(\cdot)$ denote the commitment function of Naor's commitment scheme [25]. By mBPS-CNMZK, we will refer to the modified version of the CNMZK protocol of [17] described in section 2.2. Let $\Pi_{m\text{BPS},P_i \rightarrow P_j}$ denote an instance of the

mBPS-CNMZK protocol where P_i and P_j play the roles of prover and verifier respectively. Let $\Pi_{\text{SH-PAKE}}$ be any *semi-honest* two party computation protocol that emulates the functionality $\mathcal{F}$ in the stand-alone setting. 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. $P_1 \leftrightarrow P_2 : P_1$ sends a random string z_2 (of appropriate length) to P_2 as the first message of Naor's commitment scheme. Similarly, P_2 sends a random string z_1 to P_1.
2. $P_1 \rightarrow P_2 : P_1$ creates a commitment $com_1 = \text{COM}_{z_1}(0)$ to bit 0 and sends it to P_2. P_1 and P_2 now engage in the execution of a mBPS-CNMZK argument $\Pi_{m\text{BPS},P_1 \rightarrow P_2}$ where P_1 proves that com_1 is a commitment to 0.
3. $P_2 \rightarrow P_1 : P_2$ now acts symmetrically. Specifically, it creates a commitment $com_2 = \text{COM}_{z_2}(0)$ to bit 0 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.

II. *m*PPSTV 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 function $\text{COM}_{z_1}(\cdot)$, 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 $\alpha_{1,j}^{z_{1,j}}, \ldots, \alpha_{k,j}^{z_{k,j}}$. Now, P_1 and P_2 engage in the execution of a sWI, where P_1 proves the following statement: either (a) commitments $A_{1,j}^{z_{1,j}}, \ldots, A_{k,j}^{z_{k,j}}$ can be decommitted to $\alpha_{1,j}^{z_{1,j}}, \ldots, \alpha_{k,j}^{z_{k,j}}$ 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.

III. Secure Computation Phase. In this phase, the parties run an execution of the semi-honest two party protocol $\Pi_{\text{SH-PAKE}}$ "compiled" with sWI.

Coin Flipping. P_1 and P_2 first engage in a coin-flipping protocol. More specifically, P_1 (resp., P_2) generates $r_2' \xleftarrow{\$} U_\eta$ (resp., $r_1' \xleftarrow{\$} U_\eta$) and sends it to P_2 (resp., P_1). Define $r_1'' = r_1 \oplus r_1'$ and $r_2'' = r_2 \oplus r_2'$. Now r_1'' and r_2'' respectively 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 where the proof statement in each 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. Due to lack of space, in this version, we only give the description of the simulator and bound its output queries. We refer the reader to the full version of the paper for a complete proof.

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 $\Pi_{m\text{PPSTV},H\rightarrow\mathcal{A}}$ (resp., $\Pi_{m\text{PPSTV},\mathcal{A}\rightarrow H}$) 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 section 2.2) 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 [22]). 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} sim-

ulates the *post-preamble* phase of $\Pi^{\ell}_{m\text{BPS},H\to\mathcal{A}}$ (see [17] for protocol description) in a "straight-line" fashion, as described below.

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.[10][11] The response from the trusted party is passed on to $S_{\Pi_{\text{SH-PAKE}}}$. If

[10] 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.

[11] 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,

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.[12]

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. If the reply of $\mathcal{A}$ is different from the (expected) message $\Delta^{\ell}_{2,j}$, then S_{CORE} outputs the abort symbol $\perp$.

This completes the description of our simulator $\mathcal{S} = \{S_{\text{CEC}}, S_{\text{CORE}}\}$.

4.2 Total Queries by $\mathcal{S}$

Lemma 1. *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 (see section 2.2), it follows 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 $\mathsf{special}$ 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

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.

[12] 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 look-ahead thread.)

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:

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

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 1 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.*

5 Acknowledgements

We thank Rafael Pass for pointing out that one of the arguments in an earlier draft of this paper was insufficient. We also thank Omkant Pandey, Rafael Pass and Akshay Wadia for useful discussions.

References

1. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. (1986)
2. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC. (1987)
3. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: IEEE Symposium on Security and Privacy. (1992)
4. Katz, J., Ostrovsky, R., Yung, M.: Efficient and secure authenticated key exchange using weak passwords. J. ACM **57**(1) (2009)
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: EUROCRYPT. (2000)
6. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. In: EUROCRYPT. (2000)
7. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: EUROCRYPT. (2001)
8. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: EUROCRYPT. (2003)
9. Genarro, R.: Faster and shorter password-authenticated key exchange. In: ACM Conference on Computer and Communications Security. (2008)
10. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: EUROCRYPT. (2005)

11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. (2001)
12. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. In: CRYPTO. (2001)
13. Nguyen, M.H., Vadhan, S.P.: Simpler session-key generation from short random passwords. In: TCC. (2004)
14. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: CRYPTO. (2005)
15. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC. (1990)
16. Katz, J., Ostrovsky, R., Yung, M.: Forward secrecy in password-only key exchange protocols. In: SCN. (2002)
17. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS. (2006)
18. Micali, S., Pass, R.: Local zero knowledge. In: STOC. (2006)
19. Lindell, Y.: Lower bounds for concurrent self composition. In: TCC. (2004)
20. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. J. Cryptology **19**(3) (2006)
21. Goyal, V., Sahai, A.: Resettably secure computation. In: EUROCRYPT. (2009)
22. Pandey, O., Pass, R., Sahai, A., Tseng, W.L.D., Venkitasubramaniam, M.: Precise concurrent zero knowledge. In: EUROCRYPT. (2008)
23. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. (2002)
24. Haitner, I.: Semi-honest to malicious oblivious transfer - the black-box way. In: TCC. (2008)
25. Naor, M.: Bit commitment using pseudorandomness. J. Cryptology (1991)
26. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC. (1998)
27. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC. (2001)
28. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. **30**(2) (2000)
29. Blum, M.: How to prove a theorem so no one else can claim it. In: International Congress of Mathematicians. (1987)
30. Naor, M., Ostrovsky, R., Venkatesan, R., Yung, M.: Perfect zero-knowledge arguments for np can be based on general complexity assumptions. In: CRYPTO. (1992)
31. Ostrovsky, R., Venkatesan, R., Yung, M.: 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 (1991)
32. Haitner, I., Nguyen, M.H., Ong, S.J., Reingold, O., Vadhan, S.P.: Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. SIAM J. Comput. (2009)
33. Haitner, I., Reingold, O., Vadhan, S.P., Wee, H.: Inaccessible entropy. In: STOC. (2009)
34. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC. (1988)