Session-Key Generation using Human Passwords Only

Oded Goldreich* and Yehuda Lindell

Department of Computer Science and Applied Math,
Weizmann Institute of Science, Rehovot, ISRAEL.
{oded,lindell}@wisdom.weizmann.ac.il

Abstract. We present session-key generation protocols in a model where the legitimate parties share *only* a human-memorizable password. The security guarantee holds with respect to probabilistic polynomial-time adversaries that control the communication channel (between the parties), and may omit, insert and modify messages at their choice. Loosely speaking, the effect of such an adversary that attacks an execution of our protocol is comparable to an attack in which an adversary is only allowed to make a constant number of queries of the form "is w the password of Party A". We stress that the result holds also in case the passwords are selected at random from a small dictionary so that it is feasible (for the adversary) to scan the entire directory. We note that prior to our result, it was not clear whether or not such protocols were attainable without the use of random oracles or additional setup assumptions.

1 Introduction

This work deals with the oldest and probably most important problem of cryptography: enabling *private and reliable* communication among parties that use a public communication channel. Loosely speaking, *privacy* means that nobody besides the legitimate communicators may learn the data communicated, and *reliability* means that nobody may modify the contents of the data communicated (without the receiver detecting this fact). Needless to say, a vast amount of research has been invested in this problem. Our contribution refers to a difficult and yet natural setting of two parameters of the problem: the *adversaries* and the *initial set-up*.

We consider only probabilistic polynomial-time adversaries. Still even within this framework, an important distinction refers to the type of adversaries one wishes to protect against: *passive* adversaries only eavesdrop the channel, whereas *active* adversaries may also omit, insert and modify messages sent over the channel. Clearly, reliability is a problem only with respect to active adversaries (and holds by definition w.r.t passive adversaries). *We focus on active adversaries.*

The second parameter mentioned above is the initial set-up assumptions. Some assumption of this form must exist or else there is no difference between

* Supported by the MINERVA Foundation, Germany.

the legitimate communicators, called Alice and Bob, and the adversary (which may otherwise initiate a conversation with Alice pretending to be Bob). We list some popular initial set-up assumptions and briefly discuss what is known about them.

Public-key infrastructure: Here one assumes that each party has generated a secret-key and deposited a corresponding public-key with some trusted server(s). The latter server(s) may be accessed at any time by any user.

It is easy to establish private and reliable communication in this model (cf. [15, 33]). (However, even in this case, one may want to establish "session keys" as discussed below.)

Shared (high-quality) secret keys: By *high-quality keys* we mean strings coming from distributions of high min-entropy (e.g., uniformly chosen 56-bit (or rather 192-bit) long strings, uniformly chosen 1024-bit primes, etc). Furthermore, these keys are selected by a suitable program, and cannot be memorized by humans.

In case a pair of parties shares such a key, they can conduct private and reliable communication (cf., [9, 36, 19, 4]).

Shared (low-quality) secret passwords: In contrast to high-quality keys, *passwords* are strings that may be easily selected, memorized and typed-in by humans. An illustrating (and simplified) example is the case in which the password is selected uniformly from a relatively small dictionary; that is, the password is uniformly distributed in $\mathcal{D} \subset \{0,1\}^n$, where $|\mathcal{D}| = \text{poly}(n)$.

Note that using such a password in the role of a cryptographic key (in schemes as mentioned above) will yield a totally insecure scheme. A more significant observation is that the adversary may try to guess the password, and initiate a conversation with Alice pretending to be Bob and using the guessed password. So nothing can prevent the adversary from successfully impersonating Bob with probability $1/|\mathcal{D}|$. *But can we limit the adversary's success to about this much?*

The latter question is the focus of this paper.

Session-keys: The problem of establishing private and reliable communication is commonly reduced to the problem of generating a secure session-key (a.k.a "authenticated key exchange"). Loosely speaking, one seeks a protocol by which Alice and Bob may agree on a key (to be used throughout the rest of the current communication session) so that this key will remain unknown to the adversary.[1] Of course, the adversary may prevent such agreement (by simply blocking all communication), but this will be detected by either Alice or Bob.

[1] We stress that many famous key-exchange protocols, such as the one of Diffie and Hellman [15], refer to a passive adversary. In contrast, this paper refers to active adversaries.

1.1 What security may be achieved based on passwords

Let us consider the related (although seemingly easier) task of *mutual authentication*. Here Alice and Bob merely want to establish that they are talking to one another. Repeating an observation made above, we note that if the adversary initiates $m \leq |\mathcal{D}|$ instances of the mutual authentication protocol, guessing a different password in each of them, then with probability $m/|\mathcal{D}|$ it will succeed in impersonating Alice to Bob (and furthermore find the password). The question posed above is rephrased here as follows:

> *Can one construct a password-based scheme in which the success probability of any probabilistic polynomial-time impersonation attack is bounded by $O(m/|\mathcal{D}|) + \mu(n)$, where m is the number of sessions initiated by the adversary, and $\mu(n)$ is a negligible function in the security parameter n?*

We resolve the above question in the affirmative. That is, assuming the existence of trapdoor one-way permutations, *we prove that schemes as above do exist* (for any $\mathcal{D}$ and specifically for $|\mathcal{D}| = \text{poly}(n)$). Our proof is constructive. We actually provide a protocol of comparable security for the more demanding goal of *authenticated session-key generation.*

Password-based authenticated session-key generation: Our definition for the task of authenticated session-key generation is based on the simulation paradigm. That is, we require that a secure protocol emulates an ideal execution of a session-key generation protocol (cf. [1, 29, 12]). In such an ideal execution, a trusted third party hands identical, uniformly distributed session-keys to the honest parties. The only power given to the adversary in this ideal model is to prevent the trusted party from handing keys to one of both parties. (We stress that, in this ideal model, the adversary learns *nothing* of the parties' joint password or output session-key).

Next, we consider a real execution of a protocol (where there is no trusted party and the adversary has full control over the communication channel between the honest parties). In general, a protocol is said to be secure if real-model adversaries can be emulated in the ideal-model such that the output distributions are computationally indistinguishable. Since in a password-only setting the adversary can always succeed with probability $1/|\mathcal{D}|$, it is impossible to achieve computational indistinguishability between the real model and above-described ideal model (where the adversary has zero probability of success). Therefore, in the context of a password-only setting, an authenticated session-key generation protocol is said to be secure if the above-mentioned ideal-model emulation results in an output distribution that can be distinguished from a real execution by (a gap of) at most $O(1/|\mathcal{D}|) + \mu(n)$.

Main result (informally stated): *Assuming the existence of trapdoor one-way permutations, there exists a secure authenticated session-key generation protocol in the password-only setting.*

The above (informal) definition implies the intuitive properties of authenticated session-key generation (e.g., security of the generated session-key and of the initial password). In particular, the output session-key can be distinguished from a random key by (a gap of) at most $O(1/|\mathcal{D}|)+\mu(n)$.[2] Similarly, the distinguishing gap between the parties' joint password and a uniformly distributed element in $\mathcal{D}$ is at most $O(1/|\mathcal{D}|)+\mu(n)$. (As we have mentioned, the fact that the adversary can distinguish with gap $O(1/|\mathcal{D}|)$ is an inherent limitation of password-based security.) The parties are also guaranteed that, except with probability $O(1/|\mathcal{D}|)+\mu(n)$, they either end-up with the same session-key or detect that their communication has been tampered with. Our definition also implies additional desirable properties of session-key protocols such as forward secrecy and security in the case of session-key loss (or known-key attacks). Furthermore, our protocol provides improved (i.e., negligible gap) security in case the adversary only eavesdrops the communication (during the protocol execution).

We mention that a suitable level of indistinguishability (of the real and ideal executions) holds when m sessions (referring to the same password) are conducted sequentially: in this case the distinguishing gap is $O(m/|\mathcal{D}|)+\mu(n)$ rather than $O(1/|\mathcal{D}|)+\mu(n)$ (which again is optimal). This holds also when any (polynomial) number of *other sessions w.r.t independently distributed passwords* are conducted concurrently to the above m sessions.

Caveat: Our protocol is proven secure only when assuming that the *same pair of parties* (using the same password) does not conduct several *concurrent* executions of the protocol. We stress that concurrent sessions of other pairs of parties (or of the same pair using a different password), are allowed. See further discussion in Sections 1.4 and 2.5.

1.2 Comparison to prior work

The design of secure mutual authentication and key-exchange protocols is a major effort of the applied cryptography community. In particular, much effort has been directed towards the design of *password-based* schemes that should withstand active attacks.[3] An important restricted case of the mutual authentication

[2] This implies that when using the session-key as a key to a MAC, the probability that the adversary can generate a valid MAC-tag to a message not sent by the legitimate party is small (i.e., $O(1/|\mathcal{D}|)$). Likewise, when using the session-key for private-key encryption, the adversary learns very little about the encrypted messages: for every partial-information function, the adversary can guess the value of the function applied to the messages with only small (i.e., $O(1/|\mathcal{D}|)$) advantage over the a-priori probability.

[3] A specific focus of this research has been on preventing *off-line dictionary attacks*. In such an off-line attack, the adversary records its view from past protocol executions and then scans the dictionary for a password consistent with this view. If checking consistency in this way is possible and the dictionary is small, then the adversary can derive the correct password. Clearly, a secure session-key generation protocol (as imformally defined above) withstands any off-line dictionary attack.

problem is the *asymmetric* case in which a human user authenticates himself to a server in order to *access* some service. The design of secure *access control* mechanisms based only on passwords is widely recognized as a central problem of computer practice and as such has received much attention.

The first protocol suggested for password-based session-key generation was by Bellovin and Merritt [5]. This work was very influential and became the basis for much future work in this area [6, 34, 24, 27, 31, 35]. However, these protocols have not been proven secure and their conjectured security is based on mere heuristic arguments. Despite the strong need for secure password-based protocols, the problem was not treated rigorously until quite recently. For a survey of works and techniques related to password authentication, see [28, 26] (a brief survey can be found in [23]).

A first rigorous treatment of the *access control* problem was provided by Halevi and Krawczyk [23]. They actually considered an *asymmetric* hybrid model in which one party (the server) may hold a high-quality key and the other party (the human) may only hold a password. The human is also assumed to have secure access to a corresponding public-key of the server (either by reliable access to a reliable server or by keeping a “digest” of that public-key, which they call a *public-password*). The Halevi–Krawczyk model capitalizes on the asymmetry of the access control setting, and is inapplicable to settings in which communication has to be established between two humans (rather than a human and a server). Furthermore, requiring the human to keep the unmemorizable public-password (although not secretly) is undesirable even in the access control setting. Finally, we stress that the Halevi–Krawczyk model is a *hybrid* of the “shared-key model” and the “shared-password model” (and so their results don’t apply to the “shared-password model”). Thus, it is of both theoretical and practical interest to answer the original question as posed above (i.e., without the public-password relaxation): *Is it possible to implement a secure access control mechanism* (*and authenticated key-exchange*) *based only on passwords*?

Positive answers to the original problem have been provided *in the random oracle model*. In this model, all parties are assumed to have oracle access to a totally random (universal) function [3]. Secure (password-based) access control schemes in the random oracle model were presented in [2, 11]. The common interpretation of such results is that *security is $\mathcal{LIKELY}$ to hold* even if the random oracle is replaced by a (“reasonable”) concrete function known explicitly to all parties. We warn that this interpretation is not supported by any sound reasoning. Furthermore, as pointed out in [14], there exist protocols that are secure in the random oracle model but become insecure if the random function is replaced by *any* specific function (or even a function uniformly selected from any family of functions).

To summarize, this paper is the first to present session-key generation (as well as mutual authentication) protocols *based only on passwords* (i.e., in the shared-password model), using only standard cryptographic assumptions (e.g., the existence of trapdoor one-way permutations, which in turn follows from the intractability assumption regarding integer factorization). We stress that prior to this work it was not clear whether such protocols exist at all (i.e., outside of the random oracle model).

Necessary conditions for mutual authentication: Halevi and Krawczyk [23] proved that mutual-authentication in the shared-password model implies (unauthenticated) secret-key exchange, which in turn implies one-way functions. Consequently, Boyarsky [10] pointed out that, in the shared-password model, mutual-authentication implies Oblivious Transfer.

1.3 Techniques

One central idea underlying our protocol is due to Naor and Pinkas [30]. They suggested the following protocol for the case of *passive* adversaries, using a secure protocol for polynomial evaluation.[4] In order to generate a session-key, party A first chooses a random linear polynomial $Q(\cdot)$ over a large field (which contains the dictionary of passwords). Next, A and B execute a secure polynomial evaluation in which B obtains $Q(w)$, where w is their joint password. The session-key is then set to equal $Q(w)$.

In [10] it was suggested to make the above protocol secure against *active* adversaries, by using non-malleable commitments. This suggestion was re-iterated to us by Moni Naor, and in fact our work grew out of his suggestion. In order to obtain a protocol secure against active adversaries, we augment the abovementioned protocol of [30] by several additional mechanisms. Indeed, we use non-malleable commitments [16], but in addition we also use a *specific* zero-knowledge proof [32], ordinary commitment schemes [7], a *specific* pseudorandom generator (of [9, 36, 8]), and message authentication schemes (MACs). The analysis of the resulting protocol is very complicated, *even when the adversary initiates a single session.* As explained below, we believe that these complications are unavoidable given the current state-of-art regarding *concurrent execution* of protocols.

Although not explicit in the problem statement, the problem we deal with actually concerns *concurrent* executions of a protocol. Even in case the adversary attacks a single session among two legitimate parties, its ability to modify messages means that it may actually conduct two *concurrent* executions of the protocol (one with each party).[5] Concurrent executions of some protocols were analyzed in the past, but these were relatively simple protocols. Although the high-level structure of our protocol can be simply stated in terms of a small number of modules, the currently known implementations of some of these modules are quite complex. Furthermore, these implementations are NOT known to be secure when two copies are executed *concurrently.* Thus, at the current state

[4] In the polynomial evaluation functionality, party A has a polynomial $Q(\cdot)$ over some finite field and Party B has an element x of the field. The evaluation is such that A learns nothing, and B learns $Q(x)$; i.e., the functionality is defined by $(Q, x) \mapsto (\lambda, Q(x))$.

[5] Specifically, the adversary may execute the protocol with Alice while claiming to be Bob, concurrently to executing the protocol with Bob while claiming to be Alice, where these two executions refer to the same joint Alice–Bob password.

of affairs, the analysis cannot proceed by applying some composition theorems to (two-party) protocols satisfying some concurrent-security properties (because suitable concurrently-secure protocols and composition theorems are currently unknown). Instead, we have to analyze our protocol directly. We do so by reducing the analysis of (two concurrent executions of) our protocol to the analysis of non-concurrent executions of *related* protocols. Specifically, we show how a successful adversary in the concurrent setting contradicts the security requirements in the non-concurrent setting. Such "reductions" are performed several times, each time establishing some property of the original protocol. Typically, the property refers to one of the two concurrent executions, and it is shown to hold even if the adversary is given some secrets of the legitimate party in the second execution. This is done by giving these secrets to the adversary, enabling him to effectively emulate the second execution internally. Thus, only the first execution remains and the relevant property is proven (in this standard non-concurrent setting). See Section 4 for an illustration of some of these proof techniques.

1.4 Discussion

We view our work as a theoretical study of the *very possibility* of achieving private and reliable communication among parties that share only a secret (low-quality) password and communicate over a channel that is controlled by an active adversary. Our main result is a demonstration of the *feasibility* of this task. That is, we demonstrate the *feasibility* of performing *session-key generation based only on* (low-quality) *passwords*. Doing so, this work is merely the first (rigorous) step in a research project directed towards providing a good solution to this practical problem. We discuss two aspects of this project that require further study.

Concurrent executions: Our protocol is proven secure only when the *same pair of parties* (using the same password) does not conduct several *concurrent* executions of the protocol. (We do allow concurrent executions that use different passwords.) Thus, actual use of our protocol requires a mechanism for ensuring that the same password is never used in concurrent executions. A simple mechanism enforcing the above is to disallow a party to enter an execution with a particular password if less than Δ units of time have passed since a previous execution with the same password. Furthermore, an execution must be completed within Δ units of time; that is, if Δ time units have elapsed then the execution is suspended. See Section 2.5 for further details. Indeed, it is desirable not to employ such a timing mechanism, and to prove that security holds also when many executions are conducted concurrently using the same password.

Efficiency: It is indeed desirable to have more efficient protocols than the one presented here. Some of our techniques may be useful towards this goal.

1.5 Independent work

Independently of our work, Katz, Ostrovsky and Yung [25] presented a protocol for session-key generation based on passwords. Their protocol is incomparable to ours. On one hand, their protocol uses a stronger set-up assumption (i.e., public parameters selected by a trusted party), and a seemingly stronger intractability assumption (i.e., the Decisional Diffie-Hellman). On the other hand, their protocol seems practical and is secure in an unrestricted concurrent setting. Recall that the thrust of our work is in demonstrating the feasibility of performing session-key generation based on passwords only (i.e., without any additional set-up assumptions).

2 Formal Setting

In this section we present notation and definitions that are specific to our setting, culminating in a definition of Authenticated Session-Key Generation. Given these, we state our main result.

2.1 Basic Notations

Typically, C denotes the *channel* (probabilistic polynomial-time adversary) via which parties A and B communicate. We adopt the notation of Bellare and Rogaway [4] and model the communication by giving C oracle access to A and B. We stress that, as in [4], these oracles have memory and model parties who participate in a session-key generation protocol. Unlike in [4], when A and B share a single password, C has oracle access to only a single copy of each party. We denote by $C^{A(x),B(y)}(\sigma)$, an execution of C (with auxiliary input σ) when it communicates with A and B, holding respective inputs x and y. Channel C's output from this execution is denoted by $\mathsf{output}\big(C^{A(x),B(y)}(\sigma)\big)$.

The password dictionary is denoted by $\mathcal{D} \subseteq \{0,1\}^n$, and is fixed for the entire discussion. We let $\epsilon = \frac{1}{|\mathcal{D}|}$. We denote by U_n the uniform distribution over strings of length n. For a set S, we denote $x \in_R S$ when x is chosen *uniformly* from S. We use "ppt" as shorthand for probabilistic polynomial time. We denote an unspecified negligible function by $\mu(n)$. That is, for every polynomial $p(\cdot)$ and for all sufficiently large n's, $\mu(n) < \frac{1}{p(n)}$. For functions f and g (defined over the integers), we denote $f \approx g$ if $|f(n) - g(n)| < \mu(n)$. Finally, we denote computational indistinguishability by $\stackrel{c}{\equiv}$.

A security parameter n is often implicit in our notation and discussions. Thus, for example, by the notation $\mathcal{D}$ for the dictionary, our intention is actually $\mathcal{D}_n$ (where $\mathcal{D}_n \subseteq \{0,1\}^n$). Recall that we make no assumptions regarding the size of $\mathcal{D}_n$, and in particular it may by polynomial in n.

2.2 $(1-\epsilon)$-indistinguishability and pseudorandomness

Extending the standard definition of computational indistinguishability [22, 36], we define the concept of $(1-\epsilon)$-indistinguishability. Two ensembles are $(1-\epsilon)$-

indistinguishable if for every ppt machine, the probability of distinguishing between them (via a single sample) is at most negligibly greater than ϵ. (Note that $(1-\epsilon)$-indistinguishability is not preserved under multiple samples, but for efficiently constructible ensembles $(1-\epsilon)$-indistinguishability implies $(1-m\epsilon)$-indistinguishability of sequences of m samples.) Thus, computational indistinguishability coincides with 1-indistinguishability. The formal definition is as follows.

Definition 1 ($(1-\epsilon)$-indistinguishability): *Let $\epsilon : \mathsf{N} \rightarrow [0,1]$ be a function, and let $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$ be probability ensembles, so that for any n the distribution X_n (resp., Y_n) ranges over strings of length polynomial in n. We say that the ensembles are* $(1-\epsilon)$-indistinguishable, *denoted $\{X_n\}_{n\in\mathsf{N}} \stackrel{\epsilon}{\equiv} \{Y_n\}_{n\in\mathsf{N}}$, if for every probabilistic polynomial time distinguisher D, and all auxiliary information $z \in \{0,1\}^{\mathrm{poly}(n)}$*

$$|\Pr[D(X_n, 1^n, z) = 1] - \Pr[D(Y_n, 1^n, z) = 1]| < \epsilon + \mu(n)$$

We say that $\{X_n\}_{n\in\mathsf{N}}$ is $(1-\epsilon)$-pseudorandom if it is $(1-\epsilon)$-indistinguishable from $\{U_n\}_{n\in\mathsf{N}}$. The definition of pseudorandom functions [19] is similarly extended to $(1-\epsilon)$-pseudorandom functions.

2.3 Authenticated Session-Key Generation: Definition and Discussion

The problem of password-based authenticated session-key generation can be cast as a three-party functionality involving honest parties A and B, and an adversary C. Parties A and B should input their joint password and receive identical, uniformly distributed session-keys. On the other hand, the adversary C should have no output (and specifically should not obtain information on the password or output session-key). Furthermore, C should have no power to maliciously influence the outcome of the protocol (and thus, for example, cannot affect the choice of the key or cause the parties to receive different keys). However, recall that in a real execution, C controls the communication line between the (honest) parties. Thus, it can block all communication between A and B, and cause any protocol to fail. This (unavoidable) adversarial capability is modeled in the functionality by letting C input a single bit b indicating whether or not the execution is to be successful. Specifically, if $b = 1$ (i.e., success) then both A and B receive the above-described session-key. On the other hand, if $b = 0$ then A receives a session-key, whereas B receives a special abort symbol $\perp$ instead.[6] We stress that C is given no ability to influence the outcome beyond determining this single bit (i.e., b). In conclusion, the problem of password-based session-key

[6] This lack of symmetry in the definition is inherent as it is not possible to guarantee that A and B both terminate with the same "success/failure bit". For sake of simplicity, we (arbitrarily) choose to have A always receive a uniformly distributed session-key and to have B always output $\perp$ when $b = 0$.

generation is cast as the following three-party functionality:

$$(w_A, w_B, b) \mapsto \begin{cases} (U_n, U_n, \lambda) & \text{if } b = 1 \text{ and } w_A = w_B, \\ (U_n, \perp, \lambda) & \text{otherwise.} \end{cases}$$

where w_A and w_B are A and B's respective passwords.

Our definition for password-based authenticated session-key generation is based on the "simulation paradigm" (cf. [1, 29, 12]). That is, we require a secure protocol to emulate an *ideal* execution of the above session-key generation functionality. In such an ideal execution, communication is via a trusted third party who receives the parties inputs and (honestly) returns to each party its output, as designated by the functionality.

An important observation in the context of password-based security is that, in a real execution, an adversary can always attempt impersonation by simply guessing the secret password and participating in the protocol, claiming to be one of the parties. If the adversary's guess is correct, then impersonation always succeeds (and, for example, the adversary knows the generated session-key). Furthermore, by executing the protocol with one of the parties, the adversary can verify whether or not its guess is correct, and thus can learn information about the password (e.g., it can rule out an incorrect guess from the list of possible passwords). Since the dictionary may be small, this information learned by the adversary in a protocol execution may not be negligible at all. Thus, we cannot hope to obtain a protocol that emulates an ideal-model execution (in which C learns *nothing*) up to computational indistinguishability. Rather, the inherent limitation of password-based security is accounted for by (only) requiring that a real execution can be simulated in the ideal model such that the output distributions (in the ideal and real models) are $(1-O(\epsilon))$-*indistinguishable* (rather than 1-indistinguishable), where (as defined above) $\epsilon = 1/|\mathcal{D}|$.

We note that the above limitation applies only to active adversaries who control the communication channel. Therefore, in the case of a passive (eavesdropping) adversary, we demand that the ideal and real model distributions be computationally indistinguishable (and not just $(1 - O(\epsilon))$-indistinguishable). We now define the ideal and real models and present the formal definition of security.

The ideal model: Let $\hat{A}$ and $\hat{B}$ be honest parties and let $\hat{C}$ be any ppt ideal-model adversary (with arbitrary auxiliary input σ). An ideal-model execution proceeds in the following phases:

Initialization: A password $w \in_R \mathcal{D}$ is uniformly chosen from the dictionary and given to both $\hat{A}$ and $\hat{B}$.

Sending inputs to trusted party: $\hat{A}$ and $\hat{B}$ both send the trusted party the password they have received in the initialization stage. The adversary $\hat{C}$ sends either 1 (denoting a successful protocol execution) or 0 (denoting a failed protocol execution).

The trusted party answers all parties: In the case $\hat{C}$ sends 1, the trusted party chooses a uniformly distributed string $k \in_R \{0,1\}^n$ and sends k to both $\hat{A}$

and $\hat{B}$. In the case $\hat{C}$ sends 0, the trusted party sends $k \in_R \{0,1\}^n$ to $\hat{A}$ and $\perp$ to $\hat{B}$. In both cases, $\hat{C}$ receives no output.[7]

The ideal distribution is defined as follows:

$$\mathsf{ideal}_{\hat{C}}(\mathcal{D},\sigma) \stackrel{\text{def}}{=} (w, \text{output}(\hat{A}), \text{output}(\hat{B}), \text{output}(\hat{C}(\sigma)))$$

where $w \in_R \mathcal{D}$ is the input given to $\hat{A}$ and $\hat{B}$ in the initialization phase. Thus,

$$\mathsf{ideal}_{\hat{C}}(\mathcal{D},\sigma) = \begin{cases} (w, U_n, U_n, \text{output}(\hat{C}(\sigma))) & \text{if send}(\hat{C}(\sigma)) = 1, \\ (w, U_n, \perp, \text{output}(\hat{C}(\sigma))) & \text{otherwise.} \end{cases}$$

where $\text{send}(\hat{C}(\sigma))$ denotes the value sent by $\hat{C}$ (to the trusted party), on auxiliary input σ.

The real model: Let A and B be honest parties and let C be any ppt real-model adversary with arbitrary auxiliary input σ. As in the ideal model, the real model begins with an initialization stage in which both A and B receive an identical, uniformly distributed password $w \in_R \mathcal{D}$. Then, the protocol is executed with A and B communicating via C.[8] The execution of this protocol is denoted $C^{A(w),B(w)}(\sigma)$ and we augment C's view with the $\mathsf{accept}/\mathsf{reject}$ decision bits of A and B (this decision bit denotes whether a party's private output is a session-key or $\perp$). This formal requirement is necessary, since in practice this information can be implicitly understood from whether or not the parties continue communication after the session-key generation protocol has terminated. (We note that in our specific formulation, A always accepts and thus it is only necessary to provide C with the decision-bit output by B.) The real distribution is defined as follows:

$$\mathsf{real}_C(\mathcal{D},\sigma) \stackrel{\text{def}}{=} (w, \text{output}(A), \text{output}(B), \text{output}(C^{A(w),B(w)}(\sigma)))$$

where $w \in_R \mathcal{D}$ is the input given to A and B in the initialization phase.

The definition of security: Loosely speaking, the definition requires that a secure protocol (in the real model) emulates the ideal model (in which a trusted party participates). This is formulated by saying that adversaries in the ideal model are able to simulate the execution of a real protocol, so that the input/output distribution of the simulation is $(1-O(\epsilon))$-indistinguishable from in a real execution. We further require that passive adversaries can be simulated in the ideal-model

[7] Since $\hat{A}$ and $\hat{B}$ are always honest, we need not deal with the case that they hand the trusted third party different passwords.

[8] We stress that there is a fundamental difference between the real model as defined here and as defined in standard multi-party computation. Here, the parties A and B do *not have the capability of communicating directly with each other.* Rather, A can only communicate with C and likewise for B. This is in contrast to standard multi-party computation where all parties have direct communication links or where a broadcast channel is used.

so that the output distributions are computationally indistinguishable (and not just $(1-O(\epsilon))$-indistinguishable).[9]

Definition 2 (password-based authenticated session-key generation): *A protocol for* password-based authenticated session-key generation *is secure if the following two requirements hold:*

1. Passive adversaries: *For every ppt real-model* passive *adversary C there exists a ppt ideal-model adversary $\hat{C}$ such that for every dictionary $\mathcal{D} \subseteq \{0,1\}^n$ and every auxiliary input $\sigma \in \{0,1\}^{\mathrm{poly}(n)}$*

$$\left\{\mathsf{ideal}_{\hat{C}}(\mathcal{D},\sigma)\right\}_{\mathcal{D},\sigma} \stackrel{\mathrm{c}}{\equiv} \left\{\mathsf{real}_{C}(\mathcal{D},\sigma)\right\}_{\mathcal{D},\sigma}$$

2. Arbitrary (active) adversaries: *For every ppt real-model adversary C there exists a ppt ideal-model adversary $\hat{C}$ such that for every dictionary $\mathcal{D} \subseteq \{0,1\}^n$ and every auxiliary input $\sigma \in \{0,1\}^{\mathrm{poly}(n)}$*

$$\left\{\mathsf{ideal}_{\hat{C}}(\mathcal{D},\sigma)\right\}_{\mathcal{D},\sigma} \stackrel{O(\epsilon)}{\equiv} \left\{\mathsf{real}_{C}(\mathcal{D},\sigma)\right\}_{\mathcal{D},\sigma}$$

where $\epsilon \stackrel{\mathrm{def}}{=} \frac{1}{|\mathcal{D}|}$. We stress that the constant in $O(\epsilon)$ is a universal one.

Properties of Definition 2: Definition 2 asserts that the joint input/output distribution from a real execution is at most "$O(\epsilon)$-far" from an ideal execution in which the adversary learns nothing (and has no influence on the output except to cause B to reject). This immediately implies that the output session-key is $(1-O(\epsilon))$-pseudorandom (which, as we have mentioned, is the best possible for password-based key generation). Thus, if such a key is used for encryption then for any (partial information) predicate P, the probability that an adversary learns $P(m)$ given the ciphertext $E(m)$ is at most $O(\epsilon)+\mu(n)$ greater than the a-priori probability (when the adversary is not given $E(m)$). Likewise, if the key is used for a message authentication code (MAC), then the probability that an adversary can generate a correct MAC-tag on a message not sent by A or B is at most negligibly greater than $O(\epsilon)$. We stress that the security of the output session-key does not deteriorate with its usage; that is, it can be used for polynomially-many encryptions or MACs and the security remains $O(\epsilon)$. Another important property of Definition 2 is that, except with probability $O(\epsilon)$, (either one party detects failure or) both parties terminate with the *same* session-key.

Definition 2 also implies that the password used remains $(1-O(\epsilon))$-indistinguishable from a randomly chosen (new) password $\tilde{w} \in_R \mathcal{D}$. (This can be seen from the fact that in the ideal model, the adversary learns nothing of the password w, which is part of the ideal distribution.) In particular, this implies that a secure

[9] A passive adversary is one that does not modify, omit or insert any messages sent between A or B. That is, it can only eavesdrop and thus is limited to analyzing the transcript of a protocol execution between two honest parties. Passive adversaries are also referred to as *semi-honest* in the literature (e.g., in [21]).

protocol is resistant to offline dictionary attacks (whereby an adversary scans the dictionary in search of a password that is "consistent" with its view of a protocol execution).

Other desirable properties of session-key protocols are also guaranteed by Definition 2. Specifically, we mention *forward secrecy* and security in the face of *loss of session-keys* (also known as *known-key attacks*). Forward secrecy states that the session-key remains secure even if the password is revealed after the protocol execution. Analogously, security in the face of loss of session-keys means that the password and the current session-key maintain their security even if prior session-keys are revealed. These properties are immediately implied by the fact that, in the ideal-model, there is no dependence between the session-key and the password and between session-keys from different sessions. Thus, learning the password does not compromise the security of the session-key and visa versa.[10]

An additional property that is desirable is that of *intrusion detection*. That is, if the adversary modifies any message sent in a session, then with probability at least $(1-O(\epsilon))$ this is detected and at least one party rejects. This property is not guaranteed by Definition 2 itself; however, it does hold for our protocol. Combining this with Item 1 of Definition 2 (i.e., the requirement regarding passive adversaries), we conclude that in order for C to take advantage of its ability to learn "$O(\epsilon)$-information" C must expose itself to the danger of being detected with probability $1-O(\epsilon)$.

Finally, we observe that the above definition also enables mutual-authentication. This is because A's output session-key is always $(1-O(\epsilon))$-pseudorandom to the adversary. As this key is secret, it can be used for explicit authentication via a (mutual) challenge/response protocol.[11] By adding such a step to any secure session-key protocol, we obtain explicit mutual-authentication.

Augmenting the definition: Although Definition 2 seems to capture all that is desired from authenticated session-key generation, there is a subtlety that it fails to address (as pointed out by Rackoff to the authors of [4]). The issue is that the two parties do not necesssarily terminate the session-key generation protocol simultaneously, and so one party may terminate the protocol and start using the session-key while the other party is still executing instructions of the session-key generation protocol (i.e., determining its last message). In this extended abstract, we note only that Definition 2 can be augmented to deal with this issue, and that our protocol is secure also with respect to the augmented definition. A full treatment of this issue is provided in the full version of the paper.

---

[10] The independence of session-keys from different sessions relates to the multi-session case, which is discussed in Section 2.5. For now, it is enough to note that the protocol behaves as expected in that after t executions of the real protocol, the password along with the outputs from all t sessions are $(1-O(t\epsilon))$-indistinguishable from t ideal executions.

[11] It is easy to show that such a key can be used directly to obtain a $(1-O(\epsilon))$-pseudorandom function, which can then be used in a standard challenge/response protocol.

2.4 Our Main Result

Given Definition 2, we can now formally state our main result.

Theorem 3 *Assuming the existence of trapdoor permutations, there exist secure protocols for password-based authenticated session-key generation.*

2.5 Multi-Session Security

The definition above relates to two parties executing a session-key generation protocol once. Clearly, we are interested in the more general case where many different parties run the protocol any number of times. It turns out that any protocol that is secure for a single invocation between two parties (i.e., as in Definition 2), is secure in the multi-party and sequential invocation case.

Many Invocations by Two Parties Let A and B be parties who invoke t sequential executions of a session-key generation protocol. Given that we wish that an adversary gains no more than $O(1)$ password guesses upon each invocation, the security upon the t'th invocation should be $O(t\epsilon)$. That is, we consider ideal and real distributions consisting of the outputs from all t executions. Then, we require that these distributions be $(1-O(t\epsilon))$-indistinguishable. It can be shown that any secure protocol for password-based authenticated session-key generation maintains $O(t\epsilon)$ security after t sequential invocations. Details are given in the full version of this work.

Sequential vs Concurrent Executions for Two Parties: Our solution is proven secure only if A and B do not invoke *concurrent* executions of the session-key generation protocol (with the same password). We stress that a scenario whereby the adversary invokes B twice or more (sequentially) during a single execution with A is not allowed. Therefore, in order to actually use our protocol, some mechanism must be used to ensure that such concurrent executions do not take place. This can be achieved by having A and B wait Δ units of time between protocol executions (where Δ is greater than the time taken to run a single execution). Note that parties do not usually need to initiate session-key generation protocols immediately one after the other. Therefore, this delay mechanism need only be employed when an attempted session-key generation execution fails. This means that parties not "under attack" by an adversary are not inconvenienced in any way.

We note that this limitation does *not* prevent the parties from opening a number of different (independently-keyed) communication lines. They may do this by running the session-key protocol *sequentially*, once for each desired communication line. However, in this case, they incur a delay of Δ units of time between each execution. Alternatively, they may run the protocol once and obtain a $(1-O(\epsilon))$-pseudorandom session-key. This key may then be used as a shared, high-quality key for (concurrently) generating any polynomial number of $(1-O(\epsilon))$-pseudorandom session-keys; one for each communication line (simple and efficient protocols exist for this task, see [4]).

Many Parties In the case where many parties execute the session-key protocol simultaneously, we claim that for m invocations of the protocol (which must be sequential for the same pair of parties and may be *concurrent* otherwise), the security is $O(m\epsilon)$. We assume that different pairs of parties (executing concurrently) have independently distributed passwords. Then, the security is derived from the single-session case by noting that sessions with independently distributed passwords can be perfectly simulated by an adversary.

3 Our Session-Key Generation Protocol

All arithmetic below is over the finite field $\mathrm{GF}(2^n)$ which is identified with $\{0,1\}^n$. In our protocol, we use a secure protocol for evaluating *non-constant, linear polynomials* (actually, we could use any 1–1 $\mathrm{Universal}_2$ family of hash functions). This protocol involves two parties A and B; party A has a non-constant, linear polynomial $Q(\cdot) \in \{0,1\}^{2n}$ and party B has a string $x \in \{0,1\}^n$. The functionality is $(Q, x) \mapsto (\lambda, Q(x))$; that is, A receives nothing and B receives the value $Q(x)$ (and nothing else). The fact that A is supposed to input a non-constant, linear polynomial can be enforced by simply mapping all possible input strings to the set of such polynomials (this convention is used for all references to polynomials from here on). We actually augment this functionality by having A also input a commitment to the polynomial Q (i.e., $c_A \in \mathrm{Commit}(Q)$) and its corresponding decommitment r (i.e., $c_A = C(Q, r)$). Furthermore, B also inputs a commitment value c_B. The augmentation is such that if $c_A \neq c_B$, then B receives a special failure symbol. This is needed in order to tie the polynomial evaluation to a value previously committed to in the main (higher level) protocol. The functionality is defined as follows:

Definition 4 (augmented polynomial evaluation):

- **Input:** *Party A inputs a commitment c_A and its corresponding decommitment r, and a linear, non-constant polynomial Q. Party B inputs a commitment c_B and a value x.*
- **Output:**
  1. Correct Input Case: *If $c_A = c_B$ and $c_A = C(Q, r)$, then B receives $Q(x)$ and A receives nothing.*
  2. Incorrect Input Case: *If $c_A \neq c_B$ or $c_A \neq C(Q, r)$, then B receives a special failure symbol, denoted $\perp$, and A receives nothing.*

We note that by [37, 21], this functionality can be securely computed (observe that the input conditions can be checked in polynomial time because A also provides the decommitment r).

3.1 The Protocol

Let f be a one-way permutation and b a hard-core of f.

Protocol 5 (password-based authenticated session-key generation)

- **Input:** Parties A and B begin with a joint password w, which is supposed to be uniformly distributed in $\mathcal{D}$.
- **Output:** A and B each output an accept/reject bit as well as session-keys k_A and k_B respectively (where k_A "should" equal k_B).

- **The Protocol:**
  1. **Stage 1: (Non-Malleable) Commit**
     (a) A chooses a random, linear, non-constant polynomial Q over $\mathrm{GF}(2^n)$.
     (b) A and B engage in a *non-malleable* (perfectly binding) commitment protocol in which A commits to the string $(Q, w) \in \{0,1\}^{3n}$. Denote the random coins used by B in the commitment protocol by r_B and denote B's view of the execution of the commitment protocol by $NMC(Q, w)$.[12]
     Following the commitment protocol, B sends his random coins r_B to A. (This has no effect on the security, since the commitment scheme is perfectly binding and the commitment protocol has already terminated.)
  2. **Stage 2: Pre-Key Exchange** – In this stage the parties "exchange" strings τ_A and τ_B, from which the output session-keys (as well as validation checks) are *derived*. Thus, τ_A and τ_B are called pre-keys.
     (a) A sends B a commitment $c = C(Q, r)$, for a randomly chosen r.
     (b) A and B engage in an augmented polynomial evaluation protocol. A inputs Q and (c, r); B inputs w and c.
     (c) We denote B's output by τ_B. (Note that τ_B is supposed to equal $Q(w)$.)
     (d) A internally computes $\tau_A = Q(w)$.
  3. **Stage 3: Validation**
     (a) A sends the string $y = f^{2n}(\tau_A)$ to B.
     (b) A proves to B in zero-knowledge that she input the same polynomial in both the non-malleable commitment (performed in Stage 1) and the ordinary commitment (performed in Stage 2(a)), and that the value y is "consistent" with the non-malleable commitment. Formally, A proves the following statement:
     There exists a string $(X_1, x_2) \in \{0,1\}^{3n}$ and random coins $r_{A,1}, r_{A,2}$ (where $r_{A,1}$ and $r_{A,2}$ are A's random coins in the non-malleable and ordinary commitments, respectively) such that
        i. B's view of the non-malleable commitment, $NMC(Q, w)$, is identical to the receiver's view of a non-malleable commitment to

[12] Recall that B's view consists of his random coins and all messages received during the commitment protocol execution.

(X_1, x_2), where the sender and receiver's respective random coins are $r_{A,1}$ and r_B. (Recall that r_B denotes B's random coins in the non-malleable commitment.)[13]

ii. $c = C(X_1, r_{A,2})$, and

iii. $y = f^{2n}(X_1(x_2))$.

The zero-knowledge proof used here is the *specific* zero-knowledge proof of Richardson and Kilian [32], with a *specific* setting of parameters.[14]

(c) Let t_A be the entire session transcript as seen by A (i.e., the sequence of all messages sent and received by A) and let MAC_k be a message authentication code, keyed by k. Then, A computes $k_1(\tau_A) \stackrel{\text{def}}{=} b(\tau_A) \cdots b(f^{n-1}(\tau_A))$, and sends $m = MAC_{k_1(\tau_A)}(t_A)$ to B.

4. **Decision Stage**

(a) A always accepts and outputs $k_2(\tau_A) \stackrel{\text{def}}{=} b(f^n(\tau_A)) \cdots b(f^{2n-1}(\tau_A))$.

(b) B accepts if and only if all the following conditions are fulfilled:

- $y = f^{2n}(\tau_B)$, where y is the string sent by A to B in Step 3(a) above and τ_B is B's output from the polynomial evaluation. (Note that if $\tau_B = \perp$ then no y fulfills this equality, and B always rejects.)
- B accepts the zero-knowledge proof in Step 3(b) above, and
- $\text{Verify}_{k_1(\tau_B)}(t_B, m) = 1$, where t_B is the session-transcript as seen by B, the string m is the alleged MAC-tag that B receives, and verification is with respect to the MAC-key defined by $k_1(\tau_B) = b(\tau_B) \cdots b(f^{n-1}(\tau_B))$.

If B accepts, then he outputs $k_2(\tau_B) = b(f^n(\tau_B)) \cdots b(f^{2n-1}(\tau_B))$, otherwise he outputs $\perp$. (Recall that the accept/reject decision bit is considered a public output.)

We stress that A and B always accept or reject based solely on these criteria, and that they do not halt (before this stage) even if they detect malicious behavior.

See Figure 1 below for a schematic diagram of Protocol 5.

[13] The view of a protocol execution is a function of the parties' respective inputs and random strings. Therefore, (X_1, x_2), $r_{A,1}$ and r_B define a single possible view. Furthermore, recall that B sent r_B to A following the commitment protocol. Thus A has $NMC(Q, w)$ (which includes r_B), the committed-to value (Q, w) and $r_{A,1}$, enabling her to efficiently prove the statement.

[14] The setting of parameters referred to relates to the number of iterations m in the first part of the Richardson-Kilian proof. We set m to equal the number of rounds in all other parts of our protocol plus any non-constant function of the security parameter.

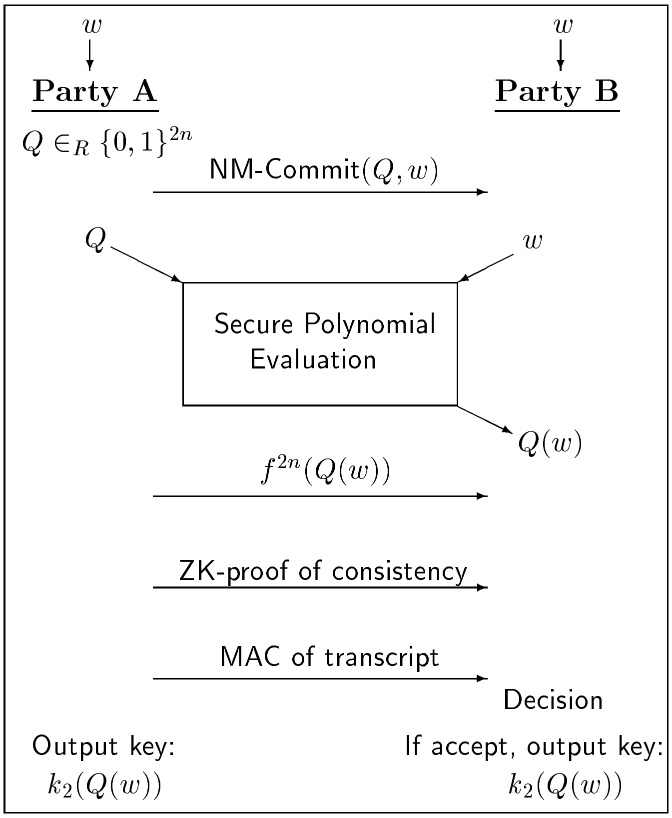

Fig. 1. Schematic Diagram of the Protocol.

In our description of the protocol, we have referred only to parties A and B. That is, we have ignored the existence (and possible impact) of the channel C. That is, when A sends a string z to B, we "pretend" that B actually received z and not something else. In a real execution, this may not be the case at all. In the actual analysis we will subscript every value by its owner, as we have done for τ_A and τ_B in the protocol. For example, we shall say that in Step 3(a), A sends a string y_A and the string received by B is y_B.

3.2 Motivation for the security of the protocol

The central module of Protocol 5 is the secure polynomial evaluation. This, in itself, is enough for achieving security against *passive* channels only. Specifically, consider the following protocol. Party A chooses a random, linear polynomial Q and inputs it into a secure polynomial evaluation with party B who inputs the joint password w. By the definition of the polynomial evaluation, B receives $Q(w)$ and A receives nothing. Next, A internally computes $Q(w)$ (she can do this as she knows both Q and w), and both parties use this value as the session-key. The key is uniformly distributed (since Q is random and linear) and due to the secrecy requirements of the polynomial evaluation, the protocol reveals nothing of w or $Q(w)$ to a passive eavesdropper C (since otherwise this would also be revealed to party A who should learn nothing from the evaluation).

One key problem in extending the above argument to our setting (where C may be active) is that the security definitions of two-party computation guarantee nothing about the simulatability of C's view in this concurrent setting. We now provide some intuition into how simulation of our protocol is nevertheless achieved. First, assume that the MAC-value sent by A at the conclusion of the protocol is such that unless C behaved passively (and relayed all message without modification), then B rejects (with some high probability). Now, if C behaves passively, then B clearly accepts (as in the case of honest parties A and B that execute the protocol without any interference). On the other hand, if C does not behave passively, then (by our assumption regarding the security of the MAC) B rejects. However, C itself knows whether or not it behaved passively and therefore can predict whether or not B will reject. In other words, the accept/reject bit output by B is simulatable (by C itself). We proceed by observing that this bit is the only meaningful message sent by B during the protocol: apart from in the polynomial evaluation, the only messages sent by B are as the *receiver* in a non-malleable commitment protocol and the *verifier* in a zero-knowledge proof (clearly, no knowledge of the password w is used by B in these protocols). Furthermore, the polynomial evaluation is such that only B receives output. Therefore, intuitively, the input used by B is not revealed by the execution; equivalently, the view of C is (computationally) independent of B's input w (this can be shown to hold even in our concurrent setting). We conclude that all messages sent by B during the execution can be simulated without knowledge of w. Therefore, by indeed simulating B, we can reduce the *concurrent* scenario involving A, C and B to a (standard) two-party setting between A and C. In this setting, we can then apply standard tools and techniques for simulating C's view in its interaction with A, and conclude that the entire real execution is simulatable in the ideal model.

Thus, the basis for simulating C's view lies in the security of the MAC in our scenario. Indeed, the MAC is secure when the parties using it (a priori) share a random MAC-key; but in our case the parties establish the MAC-key during the protocol, and it is not clear that this key is random nor the same in the view of both parties. In order to justify the security of the MAC (in our setting), we show that two properties hold. Firstly, we must show that with high probability either A and B hold the same MAC key or B is going to reject anyhow (and C knows this). Secondly, we need to show that this (identical) MAC-key held by A and B has "sufficient pseudorandomness" to prevent C from successfully forging a MAC. The proof of these properties (especially the first one) is very involved and makes up a major part of the proof, which is presented in the full version of this work.

3.3 Properties of Protocol 5

The main properties of Protocol 5 are captured by the following theorem.

Theorem 6 *Protocol 5 constitutes a secure protocol for password-based authenticated session-key generation (as defined in Definition 2).*

All the cryptographic tools used in Protocol 5 can be securely implemented assuming the existence of trapdoor permutations. Thus, at the very least, Theorem 6 implies the feasibility result captured by Theorem 3.

Unfortunately, due to lack of space in this abstract, we do not provide a proof of Theorem 6. However, a demonstration of some of the proof techniques used to prove Theorem 6 is provided in Section 4.

4 An Illustration of Our Proof Techniques

In this section, we illustrate our proof techniques for a simplified scenario in which A and B execute a secure polynomial evaluation only, while communicating via an adversarial channel C. Recall that the polynomial evaluation functionality is defined (in the stand-alone setting) by $(Q, x) \mapsto (\lambda, Q(x))$. That is, A has a polynomial $Q(\cdot)$ over some finite field and B has an element x in that field. The evaluation is such that A learns nothing while B obtains $Q(x)$. In the scenario that we are considering, A's input is a random, linear polynomial and B's input is a random password $w \in_R \mathcal{D}$ (as is the case in Protocol 5).

Recall that in this setting C may omit, insert and modify any message sent between A and B. Thus, in a sense C conducts two *separate* executions of the polynomial evaluation: one with A in which C impersonates B (called the (A, C)-execution), and one with B in which C impersonates A (called the (C, B)-execution). These two executions are carried out *concurrently* (by C), and there is no explicit execution between A and B.

We remind the reader that the definition of (stand-alone) secure two-party computation *does not apply* to the concurrent setting that we consider here. Furthermore, there are currently no tools for dealing with (general) *concurrent* computation in the two-party case. Therefore, our analysis of these executions uses specific properties of the protocol to remove the concurrency and obtain a reduction to the stand-alone setting. That is, we show how an adversarial success in the concurrent setting can be translated into a related adversarial success in the stand-alone setting. This enables us to analyze the adversary's capability in the concurrent setting, based on the security of two-party stand-alone protocols.

In order to demonstrate our proof techniques, we show that C learns "little" of w and $Q(w)$ from the above concurrent execution. Our formal statement of this has an ideal-model/real-model flavor. Specifically, we show that for every ppt adversary C interacting with A and B, there exists a *non-interactive* ppt machine $\hat{C}$ (who receives no input or output), such that $\{w, Q(w), \mathsf{output}(C^{A(Q),B(w)}\}$

is $(1-\epsilon)$-indistinguishable from $\{w, U_n, \mathsf{output}(\hat{C})\}$.[15] (Recall that $C^{A(Q),B(w)}$ denotes an execution of C with A and B holding respective inputs Q and w.) One can think of C as being a real-model adversary and $\hat{C}$ an ideal-model adversary, where in this ideal model $\hat{C}$ sends no input to the trusted third party and likewise receives no output. We note that such a view is rather simplistic as we claim nothing here regarding the outputs of A and B from the execution (as is usually required in secure computation). In other words, here we prove a statement regarding privacy, but make no claims to correctness; for example, there is no guarantee that C does not maul or skew the parties' outputs in some undesired way. Formally, we prove the following:

Theorem 7 (illustration): *For every ppt adversarial channel C interacting with A and B, there exists a ppt machine $\hat{C}$ (interacting with nobody) such that for every dictionary $\mathcal{D} \subseteq \{0,1\}^n$,*

$$\left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w)})\right\} \stackrel{\epsilon}{\equiv} \left\{w, U_n, \mathsf{output}(\hat{C})\right\}$$

where $w \in_R \mathcal{D}$, Q is a random linear polynomial, and $\epsilon = \frac{1}{|\mathcal{D}|}$.

Proof: We prove the theorem by first showing how the (C, B) execution can be simulated so that C's view in the simulation is negligibly close to in a real interaction. Then, we remain with a stand-alone execution between A and C only. In this scenario, we apply the standard definition of secure two-party computation to conclude that C learns at most "ϵ-information" about w and $Q(w)$. The fact that the (C, B) execution can be simulated is formally stated as follows (in the statement of the lemma below, $C'^{A(Q)}$ denotes a stand-alone execution of C with A upon input Q):

Lemma 8 (simulating the (C, B) execution): *For every ppt adversary C interacting with both A and B, there exists a ppt adversary C' interacting with A only, such that for every dictionary $\mathcal{D} \subseteq \{0,1\}^n$,*

$$\left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w)})\right\} \stackrel{c}{\equiv} \left\{w, Q(w), \mathsf{output}(C'^{A(Q)})\right\}$$

where $w \in_R \mathcal{D}$ and Q is a random linear polynomial.

Proof: Loosely speaking, we prove this lemma by showing that B's role in the (C, B) execution can be simulated without any knowledge of w. Thus, C' is able to simulate B's role for C and we obtain the lemma. We begin by showing that C learns nothing of B's input w from the (C, B) polynomial evaluation. This is trivial in a stand-alone setting by the definition of the functionality; here we claim that it also holds in our concurrent setting. Formally, we show that

[15] As in Definition 2, this implies that following the execution, with respect to C's view, the password w is $(1-\epsilon)$-indistinguishable from a (new) randomly chosen password $\tilde{w}$. It also implies that the value $Q(w)$ (used in Protocol 5 to derive the MAC and session keys) is $(1-\epsilon)$-pseudorandom with respect to C's view.

if B were to use some fixed $w' \in \mathcal{D}$ instead of the password w, then this is indistinguishable to C (when also interacting concurrently with A). That is,

$$\left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w)})\right\} \stackrel{c}{\equiv} \left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w')})\right\} \qquad (1)$$

where $w \in_R \mathcal{D}$ is a random password and $w' \in \mathcal{D}$ is fixed. Later, we will use Eq. (1) in order to show how C' simulates the (C, B) execution for C. First, we prove Eq. (1) by reducing C's concurrent execution with A and B to a stand-alone two-party setting between C and B only. This reduction is obtained by giving the adversary C the polynomial Q. Now, C has A's entire input and can *perfectly emulate* the (A, C) execution by itself. Formally, there exists an adversary C'', given auxiliary input Q, and interacting with B only, such that the following two equations hold:

$$\left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w)})\right\} \equiv \left\{w, Q(w), \mathsf{output}(C''^{B(w)}(Q))\right\} \qquad (2)$$

$$\left\{w, Q(w), \mathsf{output}(C^{A(Q),B(w')})\right\} \equiv \left\{w, Q(w), \mathsf{output}(C''^{B(w')}(Q))\right\} \qquad (3)$$

where $C''^{B(w)}(Q)$ denotes a stand-alone execution of C'' (given input Q) with B (who has input w). Machine C'' works by playing A's role in the (A, C)-execution and forwarding all messages belonging to the (C, B)-execution between C and B (notice that C'' can play A's role because it knows Q). We therefore remain with a stand-alone setting between C'' (given auxiliary input Q) and B, in which B inputs either w or w' into the polynomial evaluation. In this stand-alone setting, the security of the polynomial evaluation guarantees that C'' can distinguish the input cases with at most negligible probability. That is, for every ppt adversary C'', it holds that

$$\left\{w, Q(w), \mathsf{output}(C''^{B(w)}(Q))\right\} \stackrel{c}{\equiv} \left\{w, Q(w), \mathsf{output}(C''^{B(w')}(Q))\right\} \qquad (4)$$

Eq. (1) then follows by combining Equations (2), (3) and (4). In summary, we have shown that even in our concurrent setting where C interacts with both A and B, the adversary C cannot distinguish the cases that B inputs w or w'.

We are now ready to show how C' works (recall that C' interacts with A only and its aim is to simulate a concurrent execution with A and B for C). Machine C' begins by selecting an arbitrary $w' \in \mathcal{D}$. Then, C' perfectly emulates an execution of $C^{A(Q),B(w')}$ by playing B's role in the (C, B) execution and forwarding all messages belonging to the (A, C) execution between A and C (C' can play B's role here because w' is known to it). By Eq. (1) we conclude that this emulation is computationally indistinguishable from a real execution of $C^{A(Q),B(w)}$. This completes the proof of the lemma. ■

(We remark that the proof of Lemma 8 is typical of many of our proofs. Our goal is to obtain a reduction from the concurrent setting to the stand-alone setting between A and C, and we obtain this reduction by simulating B. However, in order to show that this simulation is "good" we first reduce the concurrent setting to a stand-alone setting *between C and B by simulating A*.)

It remains to show that C''s view of its (stand-alone) interaction with A can be simulated and that in this interaction, C' learn at most "ϵ-information" about w and $Q(w)$. Formally,

Lemma 9 (simulating the (A, C') stand-alone execution): *For every ppt adversary C' interacting with A, there exists a ppt machine $\hat{C}$ (interacting with nobody) such that for every dictionary $\mathcal{D} \subseteq \{0,1\}^n$,*

$$\left\{w, Q(w), \mathsf{output}(C'^{A(Q)})\right\} \stackrel{\epsilon}{\equiv} \left\{w, U_n, \mathsf{output}(\hat{C})\right\}$$

where $w \in_R \mathcal{D}$, Q is a random linear polynomial and $\epsilon = \frac{1}{|\mathcal{D}|}$.

Proof: The setting of this lemma is already that of standard two-party computation. Therefore, the security definition of two-party computation can be applied directly in order to prove the lemma. We sketch this more standard proof for the sake of completeness. We begin by showing that

$$\left\{w, Q(w), \mathsf{output}(C'^{A(Q)})\right\} \stackrel{\epsilon}{\equiv} \left\{w, U_n, \mathsf{output}(C'^{A(Q)})\right\} \quad (5)$$

In order to prove Eq. (5), recall that the security of the polynomial evaluation implies that the receiver (here played by C') can learn nothing beyond the value of $Q(\cdot)$ at a single point selected by C'. We denote this point by w_C. Then, in the case that $w_C \neq w$, the values $Q(w)$ and U_n are identically distributed (by the pairwise independence of random linear polynomials). That is, unless $w_C = w$, machine C' learns nothing of the value $Q(w)$. However, since w is uniformly distributed in $\mathcal{D}$, the probability that $w_C = w$ is at most ϵ. This means that, given C''s view, $Q(w)$ can be distinguished from U_n with probability at most ϵ.

We are now ready to define the (non-interactive) machine $\hat{C}$. Machine $\hat{C}$ works by first choosing a random linear polynomial $\hat{Q}$. Next, $\hat{C}$ perfectly emulates $C'^{A(\hat{Q})}$ by playing A's role in the execution with C ($\hat{C}$ uses the polynomial $\hat{Q}$ as A's input). Finally $\hat{C}$ outputs whatever C' does. Since w and U_n are independent of the polynomials Q and $\hat{Q}$, it follows that

$$\left\{w, U_n, \mathsf{output}(C'^{A(Q)})\right\} \equiv \left\{w, U_n, \mathsf{output}(\hat{C})\right\} \quad (6)$$

The lemma follows by combining Equations (5) and (6). ■

Combining Lemmas 8 and 9, we obtain Theorem 7. ■

We reiterate that Theorem 7 relates only to the secrecy of the password w and value $Q(w)$. Unlike Definition 2, it does *not say anything* about the outputs of the parties A and B. Furthermore, the model is significantly simplified by the fact that there is no public accept/reject bit output by the parties (as discussed in Section 3.2, simulating this bit is the most involved part of our proof). Thus, unfortunately, the above proof is merely an illustration of some of our techniques used in proving Theorem 6.

Acknowledgements

We would like to thank Moni Naor for suggesting this problem to us and for his valuable input in the initial stages of our research. We are also grateful to Alon Rosen for much discussion and feedback throughout the development of this work. We also thank Jonathan Katz for helpful discussion. Finally, we would like to thank Ran Canetti, Shai Halevi and Tal Rabin for discussion that led to a significant simplification of the protocol.

References

1. D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Fault Minority. *Journal of Cryptology*, Vol. 4, pages 75–122, 1991.
2. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *EuroCrypt 2000*, Springer-Verlag (LNCS 1807), pages 139–155, 2000.
3. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *1st Conf. on Computer and Communications Security*, ACM, pages 62–73, 1993.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, Springer-Verlag (LNCS 773), pages 232–249, 1994.
5. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the ACM/IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.
6. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pages 244–250, 1993.
7. M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, February 1982.
8. M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme which hides all partial information. In *CRYPTO'84*, Springer-Verlag (LNCS 196), pages 289–302.
9. M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SICOMP*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.
10. M. Boyarsky. Public-key Cryptography and Password Protocols: The Multi-User Case. In *Proceedings of the 6th ACM Conference on Computer and Communication Security*, 1999.
11. V. Boyko, P. MacKenzie and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In *EuroCrypt 2000*, Springer-Verlag (LNCS 1807), pages 156–171, 2000.
12. R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, Vol. 13, No. 1, pages 143–202, 2000.
13. R. Canetti. A unified framework for analyzing security of protocols. Cryptology ePrint Archive, Report No. 2000/067, 2000. Available from `http://eprint.iacr.org`.
14. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc. of the 30th STOC*, pages 209–218, 1998.

15. W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22 (Nov. 1976), pages 644–654.
16. D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, January 2000.
17. U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.
18. O. Goldreich. *Secure Multi-Party Computation.* Manuscript. Preliminary version, 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.
19. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *JACM*, Vol. 33, No. 4, pages 792–807, 1986.
20. O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. Journal of Cryptology, Vol. 9, pages 167–189, 1996.
21. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987. For details see [18].
22. S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984.
23. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. In *ACM Conference on Computer and Communications Security*, 1998.
24. D. P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, Vol 26, No. 5, pages 5–26, 1996.
25. J. Katz, R. Ostrovsky and M. Yung. Practical Password-Authenticated Key Exchange Provably Secure under Standard Assumptions. In *Eurocrypt 2001.*
26. C. Kaufman, R. Perlman and M. Speciner. *Network Security.* Prentice Hall, 1997.
27. S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, Ecole Normale Superieure, 1997.
28. A. Menezes, P. Van Oorschot and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
29. S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *Crypto'91*, Springer-Verlag (LNCS 576), 1991.
30. M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In *31st STOC*, pages 245-254, 1999.
31. S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 236–247, 1997.
32. R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, pages 415–431.
33. R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *CACM*, Vol. 21, Feb. 1978, pages 120–126.
34. M. Steiner, G. Tsudi and M. Waidner. Refinement and extension of encrypted key exchange. *ACM SIGOPS Oper. Syst. Rev.*, Vol. 29, 3, pages 22–30, 1995.
35. T. Wu. The secure remote password protocol. In *1998 Internet Society Symposium on Network and Distributed System Security*, pages 97–111, 1998.
36. A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.
37. A.C. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.