Key Exchange Using Passwords and Long Keys*

Vladimir Kolesnikov and Charles Rackoff

Dept. Comp. Sci., University of Toronto, Toronto, ON, M5S 3G4, Canada,
{vlad|rackoff}@cs.utoronto.ca

Abstract. We propose a new model for key exchange (KE) based on a combination of different *types* of keys. In our setting, *servers* exchange keys with *clients*, who memorize short passwords and carry (stealable) storage cards containing long (cryptographic) keys. Our setting is a generalization of that of Halevi and Krawczyk [17] (HK), where clients have a password and the public key of the server.
We point out a subtle flaw in the protocols of HK and demonstrate a practical attack on them, resulting in a full password compromise. We give a definition of security of KE in our (and thus also in the HK) setting and discuss many related subtleties. We define and discuss protection against *denial of access* attacks, which is not possible in any of the previous KE models that use passwords. Finally, we give a very simple and efficient protocol satisfying all our requirements.

1 Introduction

We consider the goal of enabling multiple independent secure conversations between pairs of parties over an insecure network. The most convenient and natural way to achieve this is to perform a *key exchange* (KE), that is to provide the parties with matching randomly chosen keys that can be used for securing (only) a particular conversation. Of course, each player wants to communicate with a particular person, and even a powerful adversary *Adv* should not be able to match him up with a wrong partner. Therefore, players must possess some secret information with which they can authenticate themselves. The kind of information that is available to players determines the setting of KE. The simplest KE setting is when players have a shared random string. KE is more complicated in the public key setting, where parties have public/private key pairs with the public keys securely published. The most difficult setting is the pure password setting, where parties only have a short (presumably memorizable) shared password. We note that pure password KE protocols, at least in the standard model, are currently rather complicated and inefficient, due to the complexity of the setting.

1.1 Our Setting

Consider the client-server setting where both long keys and short keys (passwords) are used for KE. Assume that the server's (e.g. bank's) keys are securely

* A shorter version of this paper appears in the Theory of Cryptography Conference 2006 [18].

stored. We take advantage of the inherent *logistical* differences in how keys are stored by the client (password in memory, long key on a storage card), to achieve more robust security than what is possible by using either type of key alone. Indeed, possession of long keys allows strong security guarantees against an online attacker. However, long keys can not be memorized, and thus must be stored, perhaps on a convenient plastic storage card. This is the vulnerability of this solution – the card may be (relatively) easily stolen by a physical attacker. On the other hand, passwords may be memorized, need not be stored, and thus can not be stolen. However, the protection against an online attacker one can hope to achieve with passwords is rather weak – passwords can always be guessed with relatively high probability. The only (somewhat satisfactory) protection against guessing attacks is recognizing them and refusing connection after a predetermined number of password failures[1].

Combining the benefits of both settings allows us to obtain a system, secure against both *types* of attack, and thus suitable for protection of sensitive information. This model is even more appealing due to its wide acceptance – it is natural for us to think of a card and a password, when we do, say, personal banking. More motivation is given in Sect. 3.

1.2 Our Contributions

We demonstrate a dangerous practical attack on the Halevi and Krawczyk (HK) [17] protocols, resulting in full compromise of any client's password (Sect. 2). The elegance, simplicity and practicality of the HK model and protocols resulted in their widespread practical use (e.g. their variants are being considered for parts of the IETF key exchange standard [13, 11]). Therefore, the discovery of our attack may also have an important practical impact.

We propose and advocate the above *Combined Key* model of key exchange (ckKE). To the best of our knowledge, it has never been formally discussed. ckKE is a generalization of the HK model.

We give a formal definition of security of ckKE (Sect 3). Defining KE even in simpler settings has proven to be notoriously difficult, with a variety of (only seemingly!) innocuous decisions to be made. We discuss the subtleties of many of our choices, such as the necessity of tightness in the allowed success of the adversary, distinguishing the types of failures and reporting them, etc. Much of our discussion (e.g. on tightness of allowed success of the adversary Adv) also applies to and benefits pure password models.

We aim to make our definition as simple and natural as possible. For example, we require the server to explicitly indicate in its output whether a password failure occurred. We find this more intuitive than defining password guessing attack as an act of interference by the adversary (e.g. a successful impersonation!), as done in previous formalizations, such as [17, 2]. Moreover, in previous

[1] We mention (but do not explicitly address) a variation of this defense against "too many" password guessing attacks. There the server limits the rate with which logins can be made, e.g. by exponentially increasing wait times between unsuccessful logins.

formalizations, such as [17, 2, 5], the attacks are accounted by the environment; the server may not even "know" they occurred (e.g. in case of successful impersonation), which makes attack recognition in practice less intuitive. We also find the game style of definitions (used in this paper) generally simpler and less prone to error than the simulation style (see discussion on the style of definition in Sect. 3.1 for more details).

We discuss unique security features available in ckKE "for free", such as the possibility of protection against the following *Denial of Access* (DoA) attack. *Adv*, attacking a player P, tries to connect to P's partner Q, using any password *pwd*. If *pwd* is correct, *Adv* wins; if not, *Adv* continues until he wins or Q refuses to connect to P. Then a legitimate P can no longer connect to Q. This easy to mount attack is unavoidable in any password-based setting (including HK) and is highly disruptive. We are not aware of the prevention of this attack being previously formalized. We formalize this attack and show how to prevent it in our model.

Finally, we give a very simple and efficient *two flow* KE protocol and prove its security (Sect. 4). An important feature of our protocol is that its flows are *independent* of each other, and thus can be sent in any order (or simultaneously), allowing for more flexibility and round efficiency.

1.3 Related Work

The problem of key exchange has deservedly received a vast amount of attention (e.g. [12, 3, 19, 1, 22, 9, 10]). The more complicated setting of pure password-based KE (pwKE) was first considered by Bellovin and Merritt [4]. Formal definitions (and protocols) in this setting were given by Bellare, Pointcheval and Rogaway [2], Boyko, Mackenzie and Patel [6], Goldreich and Lindell [14], and, recently, by Canetti et al. [8], as well as by many others.

Most relevant to our work is the problem of password-based KE in the asymmetric client-server setting, where the client has a password and the public key of the server. The question of resistance to off-line password-guessing attacks in this setting was first raised by Gong, et al. [15]. Later, Halevi and Krawczyk [17] formalized the notion of *one-way password authentication* in this setting and gave very simple and efficient protocols realizing it. They also extended their protocols to achieve key exchange with mutual authentication and perfect forward secrecy. The HK model is much simpler than the pure password model. The work of HK was the inspiration of our paper.

Further, Boyarsky [5] criticised the protocols of the earlier version [16] of [17] and suggested his own formalization of the same model. He showed several ways to amend a variant of protocols of [16] to satisfy his definition. We stress that he does not criticize protocols of the later version [17] we are considering.

Pinkas and Sander [21] consider heuristic approaches to securing password-only based authentication. They increase the cost of password-guessing and DoA attacks by using reverse Turing tests (RTT), that is, problems that are easy to solve for humans, but not for computers. We approach a different problem. In

particular, RTT techniques can not increase security of a particular client against a determined attacker.

2 Attacking the Protocols of Halevi and Krawczyk [17]

Halevi and Krawczyk give four versions of their protocol (suitable for different tasks: password transmission, one-way authentication, and key exchange in two settings). Three of the four versions (with the exception of the Encrypted Password Transmission protocol) are (similarly) affected. We demonstrate our attack on their key exchange protocol.

The Halevi-Krawczyk protocol. Let S be a server with the public key pk_S, and p be the password shared between S and the client C. Let function $f(\cdot;\cdot)$ be *one-to-one on its components*, i.e. for every fixed strings p, x, functions $f(p;\cdot)$ and $f(\cdot;x)$ are one-to-one. Let $E = (Gen, Enc, Dec)$ be a CCA2 secure encryption scheme.

Construction 1 *(The Halevi-Krawczyk Mutual Authentication and Key Exchange Protocol (Π_{HK}))*

S		C
pick a nonce n	$n, pk_S \rightarrow$	*verify* pk_S
		pick random long key k
	$\leftarrow C, n, Enc_{pk_S}(k, f(p; C, S, k, n))$	
decrypt and verify		
$y := PRF_k(n, S, C)$	$y \rightarrow$	*check* $y = PRF_k(n, S, C)$
set $K = PRF_k(y)$		*set* $K = PRF_k(y)$

The "decrypt and verify" step outputs "FAIL" if the encryption is invalid or the received value of f does not match what S computes himself. The *nonces* must satisfy the *only* requirement that they never repeat.

Our Attack exploits the structure of f. We show that the conditions imposed on f are insufficient.The flaw of the proof of security of the protocol seems to be in the incorrect conclusion in Footnote 9 on p. 258 of [17]. We note that it is possible to make the proof (of security of one-way password authentication protocol) of Halevi and Krawczyk go through by additionally requiring that $f(\cdot; C, \cdot) \neq f(\cdot; C', \cdot)$ for any unequal client names C, C'.

For simplicity, we describe our attack on a specific instantiation of Π_{HK}. We stress that natural variants of our attack apply to many choices for f, and for nonce strategies, as well as for other parameter settings.

Let client names and passwords be 10 bits long, and nonces be 30 bits long. For a variable V, let v_i be the i-th bit of V. For example, $C = \langle c_1, c_2, ..., c_{10}\rangle$ is the name of the honest player, and $n = \langle n_1, n_2, ..., n_{30}\rangle$ is the nonce. Let the function be $f(p; C, S, k, n) = \langle c_1, ...c_9, c_{10} \oplus p_1, n_1, ...n_{21}, n_{22} \oplus p_2, ..., n_{30} \oplus p_{10}, S, k\rangle$. Finally, let nonces be chosen sequentially starting from 0. Note that this is a valid configuration of Π_{HK}.

The attack proceeds as follows. *Adv* creates an honest server S, an honest client C with any name $C = \langle c_1, c_2, ..., c_{10}\rangle$, and a bad client B with the name

$B = \langle c_1, c_2, ..., c_{10} \oplus 1 \rangle$ and a randomly chosen password $p' = \langle p'_1, ..., p'_{10} \rangle$. Let p be C's password. Suppose for now that $p_1 \neq p'_1$, i.e. passwords of C and B differ in the high order bit. *Adv* observes one execution of KE between S and C. *Adv* records the encryption e sent by C and the nonce n (for concreteness, say $n =$ 00..00, e.g. n is the first nonce). Now, B logs into S as himself, as follows. S sends the nonce $n' = n+1 =$ 00..01, and B replies with $\langle B, n', e \rangle$. Now, if S doesn't fail, the password of C is computed as $pwd = \langle p'_1 \oplus 1, n_{22} \oplus n'_{22} \oplus p'_2, ..., n_{30} \oplus n'_{30} \oplus p'_{10} \rangle$ (since for $i = 22, ..., 30$, it must be that $n_{i-20} \oplus p_i = n'_{i-20} \oplus p'_i$). Also, if $p = pwd$, then S must accept, since $f(p'; B, S, k, n') = f(pwd, C, S, k, n)$. Thus, if S fails, *pwd* is eliminated from the possible passwords list.

B proceeds logging in as himself another $2^9 - 2$ times, eliminating different passwords one by one, until S accepts and that fact determines C's password. If S does not accept after B logged in $2^9 - 1$ times, B changes the first bit of his password with the server, and repeats the above entire attack (say, starting with a nonce ending with nine zeros), searching the other half-space. Finally, the two possible unchecked passwords can be verified by the same approach (and changing the password of B).

We stress that there were no attempts at impersonating C or S, and *all* failures are attributed to B. Neither C nor S know that C was attacked, thus C's account is never blocked. If B's account is blocked due to failures, B can claim mistyping and restore access. Moreover, there is no need to attack from only B's account; the attack can be easily distributed to try only a few passwords from each of many bad accounts. Again, it is easy to see that our attack is naturally generalizable to many practical instantiations of Π_{HK}.

On Boyarsky's [5] amendments of HK. The earlier version [16] of [17] had essentially the same protocol as [17], with the exception of the imposed requirements on the encryption scheme ([16] only required so-called *one-ciphertext verification attack* resistance, vs *ciphertext verification attack* resistance in [17]). Boyarsky [5] (independently from the revision resulting in the current version [17]) discovered the insufficiency of the weaker encryption. He gives his own formalization of the model and suggests three different amendments (see Sect. 5 of [5]) of the protocols of [16]. Boyarsky limits his consideration to the case where f is a concatenation function; thus our attack is not applicable to his protocols.

3 Key Exchange in the Combined Keys Model

Recall from the discussion in the Introduction that our setting (client carrying a plastic storage card and remembering a password) allows the advantage of robustness, that is graceful degradation of security in case one of the two types of keys is compromised. In particular, if the client's password is compromised, the security of KE should not suffer. On the other hand, if the card is compromised (e.g. copied), the remaining security should be that of the HK password model.

On resistance to server compromise. Halevi and Krawczyk briefly discuss resistance to insider attacks, i.e. attacks by rogue server employees who have access to some, but not all, private information stored on the server (see Sect.

3.3 in [17] for discussion of heuristic defense approaches). As another advantage of our setting, we mention that it allows stronger protection against server compromise. For example, since our clients have storage cards, public/private key pairs for each client C_i can be set up and used appropriately. Of course, an attacker who steals all the server data would now be able to successfully pose as the server. However, he can be prevented from posing as a client, as long as the client's private key remains secret. We note that such protection will require significant additional complexity of the definition and the protocol, and we leave it outside the scope of this paper. Therefore, for clarity, as do Halevi and Krawczyk, in our main exposition we assume that the server's private information is never compromised. For completeness, after presenting our protocols in Sect. 4, we briefly discuss how to modify them to protect against some consequences of server compromise.

On Denial of Access (DoA) attacks resistance. Recall that in the HK (and also in the pure password) setting, security critically depends on the ability of servers to suspend clients' accounts if there are "too many" password failures. At the same time, it is all too easy for *Adv* to cause them, making systems unusable by a trivial and easily mounted attack. In our combined key setting, it is natural to introduce protection against such DoA attacks. This can be done by requiring that polytime attackers can not cause password failures (and thus account suspension) without possession of long keys, stored on the card of the client. Of course, *Adv* may attempt attacks even without having the long keys, and furthermore, such attacks may be noticed by the servers. However, it is not hard to ensure that *Adv* does not learn anything from such attacks. This can be done, for example, by server first verifying possession of the long key (e.g. in form of a MAC), and immediately failing, if such verification failed. Then *Adv* does not learn anything about *pwd*, since it was not even used by the server. Therefore, such password guessing attacks are not a threat, and can be ignored. We formalize resistance to DoA attacks in our definition.

In our view, the main reason for using two types of keys is the two qualitatively different layers of protection against compromise. DoA resistance, although an important bonus, may not alone justify the cost of long key storage and management.

The reader may ask why one can't simply do two KE's in the two relevant models (one with parties sharing long keys, and the HK model) and combine the keys to obtain a KE protocol in our model. There are a number of issues to be addressed there. Firstly, a definition of security has to be given anyway – which is the bulk of our work. We note that some of the definitional subtleties arise specifically due to the use of both keys simultaneously. These subtleties cannot be addressed in either of the simpler models separately. See, e.g., the discussion on password updates in Sect. 4. Secondly, natural ways of combining the two KE protocols (such as establishing a secure session using long keys, and sending the password over it) result in less efficient protocols.

3.1 Pre-definition Discussion

We start by briefly recalling the general setting for KE. There is a number of players (in our case, they are divided into two types – clients and servers) who have associated credentials, and pairs of whom may have shared common information. We think of a player as an *identity*, which may have many *instantiations*. Whenever a player P wishes to talk to another player Q, an instance of P is created with the required credentials passed. Thus an instance can be thought of as a participant of a particular conversation.

It is convenient to separate the notions of identity and instance for several reasons. Firstly, it is easier to talk about the independence of instances. Independence is highly desirable to avoid maintaining state and worry about communication and synchronization between instances. Secondly, a need often arises to have several channels of communication open between two or more parties simultaneously. Then the notion of instance makes it easier to implement and model concurrent executions of KE by a player.

We do not discuss how a player P knows that he wants to talk to a player Q. This may be done as part of previous (possibly insecure) communication, scheduled to happen at some predetermined time, or be requested by a higher level protocol. We give *Adv* the power to initiate conversations between players to model all possible scenarios.

Our goal is to enable a secure conversation, or *session*, between the instances of two players. Key exchange provides corresponding pairs of participants with matching keys that can be used for securing their communication. Of course, the keys of honest parties must appear random to the adversary *Adv*, and *Adv* must not be able to cause instances to match up in an inconsistent way[2].

To formalize the latter requirement, we need to define the notion of *partners* – instances who end up having a (n intended) conversation. We use session IDs (SID) to partner instances of players. There are several ways of using SID for this purpose, and we choose what we find to be the most natural – requiring each party that output a key to have an additional output *sid*. The other ways (e.g. requiring *sid* to be an input to parties, or requiring existence of a partnering function) seem to be less intuitive. We note that many natural protocols can be naturally modified to produce session ids. The *sid* output is not necessary in real protocols; it is only used for the purpose of defining and analyzing security of KE protocols.

Definition 1. *(KE Partners) Let P be a player. We denote by P_i the i-th instance of P. We write P_i^Q to emphasize that P_i intends to do KE with (some instance of) player Q. We say that an instance C_i^S of a client C and an instance S_j^C of a server S are* partners*, if they have output the same session id sid.*

Note that no two instances are partners when they are created; they may become partners once they've executed their KE protocols. We stress that P_i

[2] We note that *Adv* can cause confusion by mismatching instances of players and making them output *unrelated* keys. We don't regard this as a problem.

and P_i^Q refer to the *same* instance of P. We may omit the superscript in P_i^Q, when it is clear from the context.

Mutual authentication (MA) is an assurance that, if P_i^Q successfully completed and output a key, there must have been a Q_j^P "communicating" with him. We choose not to require it, because it can be achieved at the cost of two additional "key confirmation" flows (and refreshing the session key). Moreover, P_i^Q can never be sure that Q_j^P "is there" anyway, since Q_j^P may go offline at any time. Note, it is rather common and accepted to not require explicit mutual authentication for these reasons (e.g. [8]). Further, if we required MA, we must use a special $\perp$ output symbol to denote failure. In our definition we allow $\perp$, but don't insist on its use.

On the notions of attacks and failures. We first note that a special kind of failure – the *password failure* – must be introduced in our model to allow protection against DoA attacks. Intuitively, if *Adv*'s attack is such that the act of failure of the server may reveal some information about the client's password, then such failure is a password failure.

A natural approach to define adversary's ability to attack the system is by counting password checking *attempts*. However, it is less natural to define what an "attempt" is. Indeed, previous works on password-based key exchange (e.g. [17, 2]) define "attempt" essentially as the act of *Adv*'s interference with the exchange of messages between two parties. However, it is less clear, for example, whether an act of *Adv* changing an insignificant bit of a message or an act of successful impersonation is such an attempt. Moreover, previously, the number of attempts was counted not by the server instances (they are not required to "know" whether a password guessing attack occurred), but by the environment.

An important feature of our definition is that servers themselves determine when, whether and what type of failure occurred. This explicates the notion of a *failed password attempt*, and ensures server's ability to identify a threat and react to it. Therefore, depending on the kind of failure, we allow servers to output either a *failure* symbol $\perp$, or a *password failure* symbol P$\perp$. We count password failures as P$\perp$'s reported by the servers, and clients accounts are suspended (to prevent further password guessing) based solely on that information and a predetermined threshold q. Therefore, a misidentification of an attack by the server is an omission of the protocol (opening a possibility of either password checking or DoA attacks), and we deem such protocols insecure.

We note that previous definitions, such as those of [17, 2, 5], can be similarly amended to ensure "explicit authentication" by additionally requiring that the server output P$\perp$ when he thinks a password attack has occurred. However, as discussed above, it seems to be cleaner to use the server's output as the only criterion for determining whether such an attack took place. Further, to ensure that the server does not misidentify the attacks, his output would need to be incorporated into the definitions, further complicating them.

The advantage of using smart cards vs storage cards is briefly discussed in Sect. 4.

On the style of definition. As mentioned earlier, we prefer the game style of KE definitions in this paper. We find it easier to understand, since the game of the definition naturally corresponds to the actions and abilities of the adversary. We don't seem to need the complexity of simulation style definitions. An exception seems to be the very complex universally composable (UC) definitions, which can model very subtle issues such as password mistyping (see [8] and discussion in Sect 3.3). In addition to their complexity, UC-secure protocols currently are significantly less efficient than protocols in other frameworks. From another point of view, it is highly desirable to have different styles of definitions to discuss their relative strengths and, hopefully, prove equivalence in some settings.

On modelling the adversary. We consider a powerful *Adv*, who schedules events (such as creation of players and their instances) and controls all communications. This latter is modelled by the parties not sending messages to each other, but giving them to *Adv* for delivery. *Adv* is allowed to arbitrarily modify the messages (including dropping and injecting them) and schedule delivery. We allow *Adv* to create and arbitrarily initialize a polynomial number of accounts for corrupted clients. Note that in this model the actions of corrupt players need not be discussed separately from the actions of *Adv*, since *Adv* can simulate all their actions. For example, a message sent by a corrupted party can be viewed as a message injected by *Adv*.

Recall, *Adv* steals either the long key or the password of a client, and attacks one of the several security features of the protocol. We describe the (five) possible settings as games the attacker plays. (These games cover all cases – the cases that are not discussed explicitly are implicitly covered by stronger settings.)

Game KE_1 models the most complicated setting where *Adv* stole the long key of the client, and is attacking a server (that is trying to distinguish server's session key from random). This is the only game where *Adv* can benefit from guessing a password. Thus, in KE_1 *Adv* is allowed a limited number of P$\perp$'s.

Game KE_2 models the setting where *Adv* stole the long key and the password of the client, but is attacking a client.

Game KE_3 models the setting where *Adv* stole only the password of the client, and is attacking a server.

Game DOA models the inability of *Adv* to cause password failures without stealing the long key.

Game SID models the inability of *Adv* to cause two honest parties output different session keys, and is included for technical reasons (see discussion before the game's definition in Sect. 3.2 below).

One way to define security is to describe one adversary who, at some point in his attack, decides which of the five games above he really wants to play. However, since *Adv*'s breaking abilities vary significantly among the games, defining allowed success of *Adv* in a "combined" game would be unnecessarily complicated. Therefore, we choose to describe five adversaries, each playing the corresponding game. We define the security of ckKE by inability of any of adversaries to win any of these games "too often". We note that it is possible to define the

"combined" adversary model carefully, and to prove that any protocol that is secure with respect to the five adversaries would also be secure with respect to one "combined" adversary.

Liveness. Note that protocols may never terminate (e.g. when *Adv* cuts the communication channels). Instances may also output special failure symbols instead of (*sid*, key) pairs (e.g. when they detect *Adv*'s interference). To ensure usability of KE protocols, we disallow these exceptional cases, unless *Adv* indeed attacks the system. Thus, we require that in the absence of an adversary, when processes communicate as intended, all sessions terminate, and intended partners output the same session id and key.

3.2 Formal Definition of Security of Key Exchange in the Combined Keys Model

Let n be a security parameter, and m be the number of bits in the password. In general, m can be a function of n; interesting cases are when m is constant or logarithmic in n. WLOG, say, the password domain is $D = \{0,1\}^m$. All players (*Adv*, clients and servers) are p.p.t. machines. Recall, the notion of partnering is defined in Def. 1.

We start by presenting the KE games. Recall, the first game models the setting where *Adv* obtained the long key of the client, is attacking a server, and is allowed a limited number of P$\perp$'s.

Game $\mathbf{KE}_1$. *The adversary Adv starts by deterministically choosing the active attack threshold $q \in 1..|D|$ (based on the security parameter n) and creating an (honest) server S. Adv chooses S's name; then S's public and private keys are set up, and only the public key revealed to Adv. Adv then runs the parties by executing steps 1-5 multiple times, in any order:*

1. *Adv creates an honest client C. Adv is allowed to pick any unused name for the client; the client C is registered with S, and long key ℓ and password pwd are set up and associated with C. Only one honest client can be created. Adv is given the long key ℓ, but not pwd.*
2. *Adv creates a corrupt client B^i. Adv is allowed to initialize him in any way, choosing any unused name, long key and password for him.*
3. *Adv creates an instance C_i of the honest client C. C_i is given (secretly from Adv) as input: his name C, the partner server's name S, the public key of S, the long key and the password of C.*
4. *Adv creates an instance S_j of the honest server S. S_j is given (secretly from Adv) as input: his name S, the private key of S, the partner client's name (C or B^i) and that client's long key and password.*
5. *Adv delivers a message m to an honest party instance. That instance immediately responds with a reply (by giving it to Adv) and/or terminates and outputs the result (either a (sid,session key) pair or the* failure symbol $\perp$*) according to the protocol. The server instance can additionally output the* password failure symbol P$\perp$*. If the total number of P$\perp$ for the honest client*

is equal to the threshold q, Adv becomes restricted – he can not deliver messages to any instances S_j^C.
Adv learns the output, with the exception of its session key part. Additionally, at any time Adv may "open" any completed honest instance – then Adv is given the session key output by that instance.

Then Adv asks for a challenge on an instance S_j^C of the server S. S_j^C, who has been instantiated to talk to the honest client C, must have completed and not failed. The challenge is, equiprobably, either the key output by S_j^C or a random string of the same length. Adv must not have opened S_j^C or a partner of S_j^C, and is not allowed to do it in the future.

Then Adv continues to run the game as before (execute steps 2-5). Finally, Adv outputs a single bit b which denotes Adv's guess at whether the challenge string was random. Adv wins if he makes a correct guess, and loses otherwise. Adv cannot "withdraw" from a challenge, and must produce his guess.

Note the following technicality of KE_1. It is possible that *Adv* may find himself unable to complete the game. This may happen only when he had just caused the q-th P$\perp$ (and hence he is not allowed to deliver messages to servers) and he has no completed instances whom he is allowed to challenge. One way to handle this would be to require *Adv* flip a coin to determine whether he won or lost. We prefer to simply disallow, by this discussion, such behaviour of *Adv*, since the stalemate can be easily avoided by *Adv* having a "safety instance" complete before he risks the q-th P$\perp$.

In all other KE games (KE_2, KE_3, SID and DOA) below, it is possible (and natural) to require that the knowledge of *pwd* does not help *Adv*. We thus choose to reveal the password to *Adv* and remove restrictions on the number of P$\perp$'s (thus removing the definition of q). These games are presented by modifying KE_1. All of the above three modifications are included in all games below (and the last two are omitted in individual descriptions for conciseness).

Game KE_2 models the setting where *Adv* stole the long key and the password of the client, but is attacking a client.

Game KE_2. *This game is identical to KE_1, with the following additional exceptions.*

- *Adv is given pwd (in addition to ℓ) and must challenge an honest client instance C_i^S, who is talking to S.*

Game KE_3 models the setting where *Adv* stole only the password of the client, and is attacking a server.

Game KE_3. *This game is identical to KE_1, with the following additional exceptions.*

- *Adv is given pwd, but not the long key ℓ.*

Game SID enforces a non-triviality condition, preventing parties from improperly partnering up (e.g. by unnecessarily outputting the same session ids).

Recall, *Adv* is not allowed to challenge parties whose partner has been opened, and we need to ensure that *Adv* is not unfairly restricted.

Game SID. *This game is identical to KE_1, with the following additional exceptions.*

- *Adv is given pwd (in addition to ℓ) and does not ask for (nor answers) the challenge.*
- *Adv wins if any two honest partners output different session keys.*

Finally, game DOA models resistance to the Denial of Access (DoA) attacks.

Game DOA. *This game is identical to KE_1, with the following additional exceptions.*

- *Adv is given pwd, but not the long key ℓ.*
- *Adv does not ask for (nor answers) the challenge.*
- *Adv wins if a server instance S_j^C outputs $P\perp$.*

Definition 2. *(Secure Key Exchange in the Combined Keys Model.) We say that a key exchange protocol Π is* secure in the Combined Keys model*, if for every polytime adversaries Adv_1, Adv_2, Adv_3, Adv_{sid} and Adv_{doa} playing games KE_1, KE_2, KE_3, SID and DOA, their probabilities of winning (over the randomness used by the adversaries, all players and generation algorithms) is at most only negligibly (in n) better than:*

- $1/2 + \frac{q}{2|D|}$*, for KE_1,*
- $1/2$*, for KE_2 and KE_3,*
- 0*, for SID and DOA.*

KE definition for the HK setting. We note that Halevi and Krawczyk do not formally define the full notion of KE in their setting, but concentrate on the *one-way password authentication* of the client to the server. Because ckKE is a generalization of the HK setting and thanks to the modularity of our presentation, it is not hard to extract the KE definition for the HK setting from Def. 2. The only difference between our and the HK settings is that we additionally allow for the use of the long shared key ℓ. It turns out that it suffices to remove the games that do not allow *Adv* to know ℓ from Def. 2, to obtain a definition for the HK setting. (Of course, we also need to remove the uses of the long key ℓ from the remaining games.) Indeed, it is not hard to verify that the remaining games cover all possible attacks *Adv* can do in the HK setting. We explicate this definition below.

Definition 3. *(Secure Key Exchange in the HK Model.) We say that a key exchange protocol Π is* secure in the Halevi-Krawczyk, or *hybrid*, model*, if for every polytime adversaries Adv_1, Adv_2 and Adv_{sid} playing (amended as described above) games KE_1, KE_2 and SID, their probabilities of winning (over the randomness used by the adversaries, all players and generation algorithms) is at most only negligibly (in n) better than:*

– $1/2 + \frac{q}{2|D|}$, *for* KE_1,
– $1/2$, *for* KE_2,
– 0, *for SID.*

We note that although the pre- and post-definition discussion (of Sect. 3.1 and 3.3) discusses the ckKE setting, much of it applies to the HK setting as well.

3.3 Post-definition Discussion

On the sufficiency of only one honest server and one honest client. We note that definition of security is not strengthened by allowing *Adv* to create additional (good or bad) servers or good clients. The reason for this is that we assume independence in the initialization procedures of each pair of identities, and each instance is initialized only with information relevant to its partner. More detail follows.

Consider an adversary who wishes to attack a particular player – a client C or a server S. Suppose we allowed creation of additional good or bad servers. Note that initialization of a client C proceeds independently for servers S^1 and S^2, and, further, $C_{i_1}^{S^1}$ has no information about $C_{i_2}^{S^2}$, that is not known to *Adv*. Therefore, creating accounts for C with more than one server and instances of C talking to them does not help *Adv*, since it can be simulated by *Adv*. On the other hand, the ability to create many clients with a server is essential, since server instances talking to different clients do share common information among themselves – the secret key of the server. In fact, we exploit that in our attack on Π_{HK}. Only one honest client is sufficient, however, since additional honest clients can be played by *Adv*. We note that had we allowed clients to possess information common to two or more servers, we would have to allow *Adv* to create additional bad servers.

Addressing Boyarsky's criticism of the single-user case ([5]), we note that our definition allows *Adv* to determine whether two honest clients have the same password, causing at least an (expected) one P$\perp$ on each of the two clients. However, we don't see it as a problem, since, with high probability, clients' passwords differ. Therefore, determining a large clique of users with the same password would cause a large number of system-wide password failures and not cause bigger than expected "bang for the buck".

On the order of creation of good client and revealing the long key ℓ. *Adv* should first create the good client, and only then be allowed to see ℓ. This is the way the attack works in real life. Had we reversed the order, it would be easy to construct good protocols that would be defined insecure (e.g., a server leaks some information, if the client's name is the same as ℓ.)

On the allowed success of *Adv* in $\mathbf{KE}_1$. Consider the success an adversary can always achieve (and therefore must be allowed in our definition). After q queries, *Adv* can guess the password with probability $q/|D|$, and if he fails to guess it, he can distinguish the key from random with probability $1/2$. Therefore, we should allow *Adv*'s probability of success of at least $\frac{q}{|D|} + \frac{1}{2}\frac{|D|-q}{|D|} = \frac{1}{2}\frac{q+|D|}{|D|} = \frac{1}{2} + \frac{q}{2|D|}$.

On independence of the states of instances. In our model, there is no global information, and state is not preserved between executions of instances of players. Therefore, for example, it is not possible for an instance to know exactly how many P⊥'s occurred. Nevertheless, some communication and preservation of state can be achieved with the help of the adversary, as follows. The private key of S now additionally includes an n-bit MAC key k_M. Whenever S_j wants to publish a message m, he gives $(m, MAC_{k_M}(m))$ to Adv. The server's protocol has an optional field in one of the expected messages. S_j only accepts the properly MAC'ed messages in that field (this is essential, so that Adv cannot forge messages). We stress that communication may only happen if it is in the interest of Adv. Therefore, it can not be used to increase security of protocols, but mainly to uncover weaknesses of definitions (see example in the next topic).

On continuing the game after q P⊥'s. In the real world, at least ideally, after q P⊥'s, the server knows there is an attack on C, and will not accept new connections and will terminate all incomplete instances. How should we model this in our KE games? Although S may have cut communication with C, old sessions may still exist, and we need to ensure that they remain secure. That is why we allow the game to continue as before, but disallow sending messages to the server instances after q P⊥'s occurred.

Observe that once Adv got the challenge, "trying" another password may not help him much. Therefore, in particular, it is crucial to allow to challenge instances *after* q P⊥'s occurred.

It is not hard to design a concrete protocol demonstrating the necessity of our choice. Take a secure protocol Π. Modify it as follows to obtain Π'. Once a P⊥ of an honest client C occurred in the game (see above discussion on independence of states), in all future sessions with instances of C the all-zero session key is chosen with fixed small, but non-negligible probability (say $prob = \frac{1}{|D|^3}$). Clearly, this is a bad protocol, since after performing only one active attack, an attacker certainly breaks into one of the next few sessions. However, Π' would be deemed secure according to the definition, if Adv is not allowed to challenge after q P⊥'s (this is because Adv is allowed only one challenge, and he does not know which is the weak session. The expected advantage of Adv is less than what he gets from the q-th password try.)

On the necessity of tightness in defining the allowed success of Adv. Note that for every non-negligible slack allowed in Adv's success, there is a natural variant of Π' above, deemed secure by such definition. While one may be tempted to not be very careful in denying Adv "a few extra password tries", Π' has a much more dangerous vulnerability, which really should be prevented. We remark that in the password-only setting, if an indistinguishability of challenge based security definition does not require tightness, a simpler variant of Π', where players *always* output an all zero key with sufficiently small (yet non-negligible) probability, would be deemed secure.

On clients mistyping the passwords. How should we model the case when an honest client mistypes the password and causes P⊥? Consider the following protocol. Take a secure protocol, and modify it, so that S_j^C reveals ℓ once P⊥

occurred. It is easy to see that the new protocol remains secure in our definition, since we implicitly assume that C never mistypes the password. Indeed, in our definition, if a P$\perp$ occurred, it must have been caused by Adv. Since Adv cannot cause P$\perp$ without possession of ℓ, it is OK if S_j^C reveals ℓ. However, intuitively, we would not want to call such a protocol secure.

The only way to formally address the issue in our model is to allow C to mistype the password. A natural first idea is to allow Adv to instantiate clients with the password of his choice. However, it is not clear that this models real life – most often clients mistype their passwords to something related.

A natural next idea is to instantiate clients with the password being $f(pwd)$, where the deterministic function f is specified by Adv. Only such an f that does not allow to check more than one password at a time may be allowed, and therefore strong restrictions on f are necessary. Indeed, setting $f(pwd) = 0$ on the first half of password domain D, and $f(pwd) = pwd$ on the second half, allows Adv to check half of password domain in one try. Restricting f to be a permutation does not work either, since applying such f allows to check whether pwd is a fixed point of f. Therefore functions f that have more than one or fewer than $|D| - 1$ fixed points are not allowed. At the same time, it is not hard to see that functions with $0, 1, |D|-1$ or $|D|$ fixed points do not allow Adv to check more than one password at a time when server is running a secure protocol, and thus may be allowed in our definition. Indeed, a function with 0 fixed points always causes S_j^C to P$\perp$; one with 1 fixed point fp always causes P$\perp$, unless $p = pwd$, and thus allows to check precisely one password; one with $|D|$ fixed points (identity) never causes P$\perp$; one with $|D| - 1$ fixed points always succeeds, unless pwd is the non-fixed point, and thus allows to check precisely one password.

At the same time, the most natural mistyping functions (e.g. confusing the order of digits) do not satisfy the requirements on f and do help the adversary (e.g. Adv can quickly test if the pin consists of the same decimal digits). More generally, Adv may infer a lot from simply observing a large volume of traffic, noting the patterns of honest clients mistyping their passwords, and matching them with expected patterns. However, it is not clear how to analyze this advantage, so we choose not to include password mistypes in our model at all, with the understanding that protocol designers take this discussion into account.

This subtlety also arises in KE in the pure password model, when passwords need not be chosen uniformly from D. Indeed, let $D_1 \subset D$ be all elements of D that end with a 0, and $pwd \in D$ is chosen uniformly from D_1. Then a protocol Π that reveals pwd iff pwd is mistyped only in the last digit, would be secure under a natural definition that does not allow mistyping. This is because pwd would not be revealed, unless Adv already had tried it. At the same time, such protocol Π should not be deemed secure. We note that the recent definition of password based KE in the complex Universal Composability model ([8]) addresses the issue of mistyping by allowing the environment both *choose* and *type* passwords.

On reporting failures to Adv immediately after failing. Consider a modification of Π_{HK}, where, upon a password failure, the server does not report it to Adv, but produces a random key and simulates successful completion of KE.

This change would have prevented our attack of Sect. 2. However, the achieved security would be illusory, since, in practice, it is hard to simulate successful completion well. Indeed, the fact of P$\perp$ must be somehow registered and used by S. This changes the state of S (in particular, the counter of active attacks is incremented). Since C can login after $q-1$, but not after q P$\perp$'s, Adv is able to infer some information about S' outputs. To account for such "side channels", we require that players don't have private failure outputs (either $\perp$ or P$\perp$), and Adv is informed of failure as soon as it output. Note that this discussion relates to the *Additional discussion* in Sect. 2.1 of [8], where the authors argue that Adv need not know whether the passwords of two honest partners matched.

To further illustrate this point, suppose S_j at some point "knows" he is going to output P$\perp$, that is, S_j entered a state from which all execution paths lead to outputting P$\perp$, and Adv learned this fact. Suppose S_j does not terminate yet, but is waiting to receive another message. Then Adv can delay the delivery of the message indefinitely, S_j would never report P$\perp$, and we don't count it. In particular, adding an extra round of communication to a secure protocol Π, in which parties say whether they failed, makes Π insecure. This is consistent with our desire to force a server to correctly and timely report active attacks.

4 Our Protocol

Let n be a security parameter. To simplify discussion, we present our constructions with the domains and ranges of PRFG and MAC equal to $\{0,1\}^n$. Let $E = (Gen, Enc, Dec)$ be a CCA2 secure public key encryption scheme, $F : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^n$ be a PRFG, and $MAC : \{0,1\}^n \times \{0,1\}^* \mapsto \{0,1\}^n$ be a message authentication code. Let N_C be the name of the client C, drawn from $\{0,1\}^n$. Shorter names can be used for efficiency, if desired.

Consider the following KE protocol Π, with two types of players, a server S and a client C who have secretly agreed on a password $pwd \in_R D$, a long secret key $\ell \in_R \{0,1\}^n$. Also, S has generated public/private key pair (pk_S, sk_S), and gave pk_S to C.

Construction 2 *(KE in the Combined Key Model (Π).)*

S^C		C^S
choose $r \in_R \{0,1\}^n$		*choose* $k \in_R \{0,1\}^n$, *set* $\alpha = Enc_{pk_S}(N_C, pwd, k)$
	$r \rightarrow \cdots \leftarrow \alpha, MAC_\ell(\alpha)$	
verify $MAC_\ell(\alpha)$ *and* N_C*;* *if fail, output* $\perp$ *and halt*		*output* $K = F_k(r), sid = (r, \alpha)$
verify pwd; *if fail, output P*$\perp$ *and halt*		
else output $K = F_k(r), sid = (r, \alpha)$		

WLOG, we assume that all protocol messages are formed properly (i.e. values are drawn from the appropriate domains, etc.). Then a client instance never fails, while a server instance may. Note that Adv may cause non-partnered parties to output unrelated keys. This is not a problem (see Sect. 3.1 and Footnote 2).

We stress that the two flows of the protocol are independent, and thus either of the parties can be the initiator. The DoA attacks are prevented if Adv does not have ℓ, even though, in particular, Adv is able to resend old messages of the client. The latter causes a server to output a random (from the point of view of Adv) session key, thus Adv is not able to take advantage of it. This also does not enable Adv to "reset" the fail counter in real executions (and thus try many passwords undetected), since the same effect can be achieved by Adv executing a KE between honest S_j^C and C_i^S, and then cutting the communication.

We treat the policies of account suspension and resetting of failure counters as external to our discussion, but stress that care should be taken in designing and implementing them. In particular, the client's explicit consent (communicated over a secure session) should be necessary for resetting the failed attempts counter, since otherwise Adv can be undetected when trying passwords between legitimate client logins. A natural scenario would be that the server asks the client whether he mistyped the password a certain number of times, and when client confirms, the fail counter is reset.

We further note that we can prevent Adv from resending C's old replies to S (e.g. if it is undesirable to have "hanging" sessions) by including r in the encryption of the client's reply and adding the corresponding verification step to S. We chose not to include it because it disallows the independence of flows of KE, and it is unclear whether hanging sessions are "worse" than hanging KE.

An alert reader will notice that smart cards may be gainfully used in place of client's storage cards. A smart card may hide the long key ℓ, only exposing the MAC'ing interface. An interesting setting is when Adv can "borrow" and return (but not copy) the card, obtaining only a period of ability to MAC strings of his choice. Our protocol will not benefit from such security improvements: C's messages are independent of S's, and thus Adv can MAC all the strings he might possibly need for an attack (e.g. strings containing all possible passwords) in one batch. Again, including r in the encryption of C's reply resolves this problem.

Π is secure. We first observe that for every Adv_{sid} and Adv_{doa} playing games SID and DOA, their probability of winning is negligible. Indeed, in our protocol, partners never output different keys (since the session key is determined by sid). As for Adv_{doa}, for a server to output P$\perp$, it is necessary to forge a MAC on an encryption not produced by any of the honest clients. This is only possible with negligible probability without the knowledge of the long key ℓ, assuming security of MAC.

We formally consider the remaining games KE_i and adversaries in the appendix. The structure of our proof is as follows. We start by reducing the KE adversaries to ones playing much simpler games. As a second step, we show that existence of new adversaries implies insecurity of either of the employed primitives. We consider several adversarial behaviours separately. Appendix A.1

discusses the most interesting setting, where the adversary sees the long key and challenges a server instance, and we formally and carefully show the precise quantitative security of Π. Discussion of this section contains the main ideas of our entire proof. Appendix A.2 addresses the remaining two games. Altogether, we've proven

Theorem 1. *The protocol Π of Constr. 2 is a secure key exchange protocol in the combined keys model.*

On generalizing Constr. 2. Consider creating a family of protocols parameterized by a function f similarly to the approach of Halevi and Krawczyk. The goal is to shorten the plaintext of the encryption α sent by C, which may improve the performance of the protocol. We note that we already reduce the amount of data under the CCA2-secure encryption – it is smaller than in any member of the HK families of KE protocols (but note that HK KE additionally achieve mutual authentication). We do not see how to further significantly increase efficiency by applying the HK idea to our protocols.

KE protocols for the HK setting. It is easy to see that removing the uses of the long key ℓ from the protocol of Constr. 2 casts it into the HK setting. The obtained protocol (explicated in Constr. 3 below) is a secure KE protocol in the HK setting, according to Def. 3. This conclusion immediately follows from the method of construction and Theorem 1.

Construction 3 *(KE in the HK setting.)*

S^C		C^S
choose $r \in_R \{0,1\}^n$		*choose* $k \in_R \{0,1\}^n$, *set* $\alpha = Enc_{pk_S}(N_C, pwd, k)$
	$r \rightarrow \cdots \leftarrow \alpha$	
verify N_C*;* *if fail, output* $\perp$ *and halt*		*output* $K = F_k(r), sid = (r, \alpha)$
verify pwd; *if fail, output* $P\perp$ *and halt*		
else output $K = F_k(r), sid = (r, \alpha)$		

Protection against the compromise of the server's password file. Recall that we previously assumed that the server's private information is never compromised. We now briefly discuss how storing passwords in a hashed form helps maintain a reasonable level of security even if *Adv* steals the password file. This discussion is informal, since, even though a password may be forced to be sufficiently long (40-60 bits), due to human memory limitations, it contains only a small amount of entropy. Consequently, it is hard, if at all possible, to formally justify the advantage of the server storing hashes of passwords instead of their plaintext values. Indeed, if *Adv* steals a file of hashed user passwords (or any other information allowing S to verify a password), he can compute the corresponding passwords in polytime.

However, in practice it is often unclear how to exploit the low entropy of passwords. See Narayanan and Shmatikov [20] for recent results and background in password cracking. Further, most of the attacks (including that of [20]) employ expensive precomputation, after which they can attack multiple passwords at a much lower cost per password. Recall, the benefits of precomputation are removed by "salting", which can be viewed as using a different hash function for each user's password. In addition, slow hash functions may be used to increase the cost of the attack, as discussed in [7] and is done in the UNIX `crypt()` implementation.

Thus, storing passwords only in the salted hashed form on the server seems to provide additional significant level of protection, at least with the current state of the art of password cracking. We note that our protocols can be trivially modified to allow for this second line of defense, e.g., as follows. The server S will store a randomly chosen salt s and a hash $h = H(pwd, s)$, instead of the C's password pwd. The client C stores s on the card. When computing α above, C includes $H(pwd, s)$ instead of the plaintext pwd. The password verification procedure of S^C is amended correspondingly. We envision the above modifications for most practical situations. As previously discussed, other second-line defense techniques, e.g. those described in [17], can be used to also achieve heuristic security against the compromise of the secret key of the server. Finally, we note that a compromise of the long key ℓ of the client (which is also stored on the server) is already addressed in our definition by allowing Adv to steal ℓ.

How to update passwords. In practice, throughout the life cycle of a client-server system, it might be necessary to update passwords of clients. Usually, in the KE literature this need is treated as external to the definition and protocols. It turns out that in our setting it requires special care. We now briefly describe the subtle problem and informally suggest several solutions.

Suppose client C securely (e.g. in a private meeting with the server S) updates his password from pwd to pwd'. Then Adv can perform a DoA attack by simply sending to S^C C's old messages, containing (properly encrypted and MAC'ed) pwd. Intuitively, the problem arises from the fact that only a part of the key of C is modified when the password is updated. In other words, clients with related credentials would exist in the system, violating our assumption on the independence of key generation of players (see first item in Sect. 3.3 for more discussion).

A natural solution is to disallow password-only updates, and to require regeneration of the long key ℓ as well. Such updates will not cause problems, since the new key (pwd', ℓ') is fully independent from the old one, and all previous transcripts obtained by Adv become useless[3].

We also note that it is possible to allow password-only updates, at the cost of complicating the protocol (and the definition). This may be desirable when updates of the client's storage are inconvenient or costly. Security in this setting can be achieved, for example, by a modification of our protocol into a

[3] Note a technicality that incomplete instances S^C and C^S should be terminated at the time of key update.

challenge-response one. Alternatively, it is possible to preserve the property of independence of flows in the KE protocol. This can be done at the cost of keeping (and appropriately using) password update counters by both S and C[4].

Our definitions also would need to be modified if password-only updates are allowed. Indeed, our protocol of Constr. 2 is secure according to Def. 2, yet it is clearly vulnerable to the DoA attack in the current setting. We leave the resulting update to the definitions outside the scope of this paper.

Acknowledgements. We thank Shai Halevi and the anonymous referees of TCC 2006 for many great comments on earlier versions of this work. We are grateful to Ian F. Blake for several stimulating discussions. We also thank Vitaly Shmatikov for a discussion on password cracking and the importance of avoiding storing passwords in plaintext. The authors were in part supported by Natural Sciences and Engineering Research Council of Canada (NSERC) grants. The first author was also supported by Ontario Graduate Scholarship (OGS).

References

1. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 419–428, New York, NY, USA, 1998. ACM Press.
2. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, pages 139–155, 2000.
3. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
4. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secureagainst dictionary attacks. In *SP '92: Proceedings of the 1992 IEEE Symposium on Security and Privacy*, page 72, Washington, DC, USA, 1992. IEEE Computer Society.
5. Maurizio Kliban Boyarsky. Public-key cryptography and password protocols: the multi-user case. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 63–72, New York, NY, USA, 1999. ACM Press.
6. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-hellman. In B. Preneel, editor, *Proceedings EUROCRYPT 2000*, pages 156–171, 2000.
7. Samuel R. Buss and Peter N. Yianilos. Secure short key cryptosystems: 40 bits are enough. Technical report, NEC Research Institute, Princeton, NJ, 1999.

[4] While storage on the server side is readily available, we may have a read-only client's storage medium. If so, the counter may be considered as part of the password, with the assumption that the upper bound on the number of password updates is small. We stress that in this case, a non-matching counter value will cause S^C output $\perp$, and not P$\perp$.

8. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, pages 404–421, 2005.
9. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 453–474, London, UK, 2001. Springer-Verlag.
10. Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–351, London, UK, 2002. Springer-Verlag.
11. T. Clancy. Eap password authenticated exchange, draft archive. http://www.cs.umd.edu/ clancy/eap-pax/, 2005.
12. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
13. Internet Engineering Task Force. Eap password authenticated exchange. http://www.ietf.org/internet-drafts/draft-clancy-eap-pax-03.txt, 2005.
14. Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 408–432, London, UK, 2001. Springer-Verlag.
15. L. Gong, M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
16. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 122–131, New York, NY, USA, 1998. ACM Press.
17. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.*, 2(3):230–268, 1999.
18. Vladimir Kolesnikov and Charles Rackoff. Key exchange using passwords and long keys. In *Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 5-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2006.
19. H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *SNDSS '96: Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96)*, page 114, Washington, DC, USA, 1996. IEEE Computer Society.
20. Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM Conference on Computer and Communications Security*, pages 364–372, 2005.
21. Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170, New York, NY, USA, 2002. ACM Press.
22. Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120 (#93166), IBM, 1999.

A Proof of security of the protocol of Constr. 2 (Theorem 1)

We first prove that, assuming security of the underlying primitives of Π, there does not exist an adversary winning the game KE_1 too often. The proof of this case is delicate due to handling precise quantitative advantage of Adv; it presents main ideas for the proof of the other cases.

Proposition 1. *If the PRFG F and the CCA2 encryption scheme E used in Π are secure, then for every polytime Adv, the probability p of Adv winning the game KE_1 is no more than $1/2 + \frac{q}{2|D|} + \epsilon$, where ϵ is negligibly small (in the security parameter n).*

Prop. 1 follows from lemmas 1 and 2, presented in the Appendix A.1.
The other cases are handled by

Proposition 2. *If the PRFG F, MAC, and the CCA2 encryption scheme E used in Π are secure, then for every polytime Adv_1 and Adv_2 the probabilities of them winning the games KE_2 and KE_3 respectively are no more than $p > 1/2+\epsilon$, where ϵ is negligibly small (in the security parameter n).*

Prop. 2 follows from lemmas 3, 4 and 5, presented in the Appendix A.2.
Theorem 1 follows from Prop. 1 and 2.

A.1 Proof for the case when the adversary is given the long key and challenges the server

Consider the following game (parameterized by n). that a distinguisher $Dist_1$ plays. We suggest looking at the game briefly at the first reading – the motivation behind it would be clear in the proof of the reduction from game KE_1 (Lemma 1).

Game G_1. *A maximum number of "password tries" q is deterministically (based on n) chosen by $Dist_1$ and fixed. The game initializes a CCA2 secure encryption scheme (by generating public and private keys pk_S and sk_S) and randomly chooses the password $pwd \in_R D$. Only the public key pk_S is given to $Dist_1$. $Dist_1$ queries the* decryption oracle $O_D(e') = Dec_{sk_S}(e')$ *to obtain decryptions of chosen strings. Then $Dist_1$ chooses a "client name" N_C. Then, for $i = 1, ..., u$, $Dist_1$ queries the* encryption oracle O_E *that produces random encryptions $e_i = Enc_{pk_S}(N_C, pwd, k_i)$, where $k_i \in_R \{0,1\}^n$ are chosen randomly and unknown to $Dist_1$. Here u is chosen by $Dist_1$. Then $Dist_1$ proceeds by executing Steps 1 - 2 multiple times, in any order:*

1. *$Dist_1$ queries the* PRFG oracle $O_F(i, r) = F_{k_i}(r)$, *where k_i was chosen (but not revealed) by O_E during it's i-th query. Here $r \in \{0,1\}^n$ and $i \in \{1..u\}$ are chosen by $Dist_1$.*
2. *$Dist_1$ queries the* decryption oracle $O_D(e')$, *where e' is chosen by $Dist_1$. He is not allowed to query O_D on any e_i obtained from O_E.*

Then $Dist_1$ chooses $i \in \{1, ..., u\}$ and $r_0 \in \{0,1\}^n$ and queries the challenge oracle $O_C(i, r_0)$. *O_C produces a challenge as follows: it randomly chooses a bit b and a string $\rho \in_R \{0,1\}^n$. Then $O_C(i, r_0) = F_{k_i}(r_0)$ if $b = 0$, and $O_C(i, r_0) = \rho$ if $b = 1$. $Dist_1$ is not allowed to query $O_C(i, r_0)$, if he queried $O_F(i, r_0)$.*

Then, $Dist_1$ continues running Steps 1-2, with the exception that he is not allowed to query $O_F(i, r_0)$.

Finally, $Dist_1$ generates a list of q password guesses $PL = \{p_1, ..., p_q\}$ and outputs a bit b'. $Dist_1$ wins if $pwd \in PL$ or if $b = b'$.

Lemma 1. *Suppose there exists an adversary Adv that asks for the long key ℓ, always challenges a server instance, and breaks the protocol Π. Then there exists $Dist_1$ winning the game G_1 with probability non-negligibly greater than $1/2 + \frac{q}{2|D|}$, where G_1 is run with the same encryption scheme E and PRFG F as Π.*

Proof: We prove the theorem by constructing $Dist_1$ that wins G_1, essentially whenever *Adv* wins the KE game. $Dist_1$ simulates an environment (i.e. KE players and their actions), in which he runs *Adv*, answers *Adv*'s queries and uses *Adv*'s decisions to make decisions in G_1. We say "$Dist_1$ stops", meaning "$Dist_1$ finishes processing *Adv*'s request and returns control to *Adv*", and "$Dist_1$ sends (outputs) m", meaning "$Dist_1$ simulates the given player sending (outputting) m, by giving m to *Adv*".

$Dist_1$ starts up *Adv*, who outputs the threshold q and requests to create (the only) server S. $Dist_1$ then starts the game G_1 with q, and obtains the public key pk_S for *Enc*. $Dist_1$ sends pk_S to *Adv* as the public key of the server. $Dist_1$ initializes its password list PL to empty.

$Dist_1$ then runs *Adv* and satisfies its requests for information as follows. Note that a client C must have been created to create its instances C_i or server instances S_j^C.

1. *Adv creates a bad client B^i:*
   Adv chooses the password and the long key, and reveals them to S (thus giving them to $Dist_1$).
2. *Adv creates (the only) honest client C with the name N_C:*
   $Dist_1$ chooses the name N_C for G_1 to be the name of the client. Let u be the upper bound on the number of client instances *Adv* creates. Then, for $i = 1, ..., u$, $Dist_1$ queries oracle O_E and obtains random encryptions $e_i = Enc_{pk_S}(N_C, pwd, k_i)$, where $k_i \in_R \{0,1\}^n$ are chosen randomly and are unknown to $Dist_1$. (We note that *Adv* did not cause any calls to O_F or O_C yet, although he may have created and run server with corrupt clients. Therefore, there is no conflict with G_1's scheduling.) Then $Dist_1$ randomly chooses $\ell \in_R \{0,1\}^n$ to be C's long key. *Adv* asks for it, so $Dist_1$ reveals ℓ to *Adv*.
3. *Adv creates an instance S_j^C or $S_j^{B^i}$ of S and starts the protocol:*
   $Dist_1$ randomly chooses $r_j \in_R \{0,1\}^n$ and sends it.

4. *Adv creates new (i-th) instance C_i of the honest client C.*
   Recall that $Dist_1$ already obtained e_i from O_E. $Dist_1$ computes $mac_i = MAC_\ell(e_i)$ and sends (e_i, mac_i).
5. *Adv delivers a message m_{C_i} to an instance C_i of honest client C (allegedly) from server S:*
   $Dist_1$ gives to *Adv* the session id $sid_i = (m_{C_i}, e_i)$. Recall, e_i is the encryption previously sent by C_i.
6. *Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C (recall, C is honest):*
   If $m' \neq MAC_\ell(e')$, $Dist_1$ outputs $\perp$and stops. Otherwise $Dist_1$ proceeds as follows.
   If $e' = e_i$ was obtained from O_E, $Dist_1$ gives to *Adv* the session id $sid_j = (r_j, e_i)$. Recall, r_j is the message previously sent by S_j^C.
   Otherwise, if e' was not obtained from O_E, $Dist_1$ continues and decrypts e' by querying the decryption oracle $O_D(e')$ to obtain (N'_C, pwd', k'). If $N'_C \neq N_C$, $Dist_1$ outputs $\perp$and stops. Otherwise, i.e. if the client's name matches, $Dist_1$ adds pwd' to the list PL of passwords to try, unless this causes $|PL| > q$. (Since *Adv* cannot communicate with S_j^C after q P$\perp$'s, the only case when *Adv* causes the $q+1$-st execution of this clause is when *Adv* had produced a valid guess at C's password. If so, *pwd* has already been added to PL, and there is no benefit in adding anything to PL.) Finally, $Dist_1$ outputs P $\perp$ and stops. (Note if this response is incorrect, then *pwd* has been added to PL, and $Dist_1$ wins, so we don't worry about properly simulating the game anymore.)
7. *Adv delivers a message $m_{S_j} = (e', m')$ to $S_j^{B^i}$ (allegedly) from client B^i (recall, B^i is corrupt):*
   Recall that $Dist_1$ knows B^i's long key and password. $Dist_1$ verifies MAC; if MAC failed, $Dist_1$ outputs $\perp$and stops. If $e' = e_i$ was obtained by any oracle call to O_E, $Dist_1$outputs $\perp$and stops(since the client name would not verify[5].)
   Otherwise (if MAC checked and e' was not obtained from O_E) $Dist_1$ proceeds as follows. $Dist_1$ decrypts e' by querying the decryption oracle $O_D(e') = (N'_C, pwd', k')$ and acts according to the Server's protocol, as follows. $Dist_1$ verifies whether N'_C equals to the name of B^i. If not, $Dist_1$ outputs $\perp$and stops. Then $Dist_1$ verifies whether pwd' is the B^i's password; if not, $Dist_1$ outputs P$\perp$ and stops. Otherwise, $Dist_1$ gives to *Adv* the session id $sid_j = (r_j, e')$.
8. *Adv sends an* open *request on a (completed and not failed or challenged) client instance C_i of C:*
   Note that C_i output $sid_i = (m_{C_i}, e_i)$. $Dist_1$ queries oracle $O_F(i, m_{C_i}) = F_{k_i}(m_{C_i})$, and gives the answer to *Adv*. Note that there are restrictions on

[5] This conclusion cannot be made when attempting to reduce the KE game of the Halevi-Krawczyk protocol to G_1 in a natural way, and thus $Dist_1$ cannot answer correctly without calling $O_D(e_i)$. However, it is crucial that $O_D(e_i)$ is not called here.

when $Dist_1$ is allowed to call O_F (O_F and O_C cannot be called with the same parameters). We argue later that we are not violating them.

9. *Adv sends an* open *request on a (completed and not failed or challenged) server instance S_j of S:*
   Recall that S_j received $m_{S_j} = (e', m')$ and S_j output $sid_j = (r_j, e')$. If $e' = e_i$ was generated by O_E, then $Dist_1$ queries oracle $O_F(i, r_j)$ and outputs the answer. As in 8, we will later argue that we are not violating G_1's restrictions. Otherwise, $Dist_1$ decrypts e' by calling $O_D(e')$ and outputs $F_{k'}(r_j)$, where k' is the key inside e'. Note that this is the case corresponding to the last paragraph of case 7 above, since $Dist_1$ always reports failure when S_j^C receives e' not generated by O_E. No O_F call is made in this clause.
10. *Adv sends a* challenge *request on a (completed and not failed or opened) server instance S_j^C of S:*
    Recall, S_j^C sent r_j, received $m_{S_j} = (e', m')$ and output $sid_j = (r_j, e')$. If $e' = e_i$ was generated by O_E (i.e. sent by a client C_i), $Dist_1$ queries the challenge oracle $ch = O_C(i, r_j)$, gives ch to Adv and (later, after submitting the list PL) submits Adv's output as his answer to the challenge of G_1. As in 8 and 9, we will later argue that we are not violating G_1's restrictions when querying $O_C(i, r_j)$.
    Note that the case when e' of m_{S_j} was not generated by O_E cannot happen, since $Dist_1$ would have reported to Adv that S_j^C failed.

We note that $Dist_1$ always ensures legality of calls to $O_D(e)$ by checking that e was not generated by O_E. We now argue that all calls to O_F in 8–9, and to O_C in 10 will be legal requests in G_1, that is that $Dist_1$ never calls both $O_F(i, r)$ and $O_C(i, r)$, for any pair (i, r).

First note that O_F and O_C are only called when Adv opens or challenges instances, respectively. Adv always challenges a server instance. Suppose, he challenged S_j^C, and thus caused the call $O_C(i, r_j)$, where e_i was generated by O_E and sent by some client C_i. Consider two possible cases. First, for $k \neq j$, Adv opens (either earlier or later) a server instance S_k, causing a call $O_F(i', r_k)$. This call is legal, since $Prob(r_j = r_k)$ is negligible. Second, Adv opens a client instance C_k^S, thus causing a call $O_F(k, m_{C_k})$. Suppose this call is illegal, i.e. $i = k$ (implying that $e_i = e_k$) and $r_j = m_{C_k}$. However, in this case, the session ids output by the parties match. Then S_j^C and C_k^S are partners, and such Adv's behaviour is not allowed in KE_1.

Now it is easy to see that the simulated messages provided by $Dist_1$ are distributed almost identically to those generated in a real execution, until the point when Adv does guess the password correctly, and $Dist_1$ incorrectly returns P$\perp$. What happens after that point, however, does not matter, since $Dist_1$ had already won the game.

By assumption of the lemma, Adv wins with probability non-negligibly more than $1/2 + \frac{q}{2|D|}$. It is easy to see that $Dist_1$ wins whenever Adv wins, except for a negligible fraction of the time. Therefore, the constructed $Dist_1$ wins the game G_1 with probability non-negligibly more than $1/2 + \frac{q}{2|D|}$.

□

We now show that the adversary $Dist_1$ described in Lemma 1 cannot exist, if secure primitives are used.

Lemma 2. *If the PRFG F and the CCA2 encryption scheme E used in G_1 are secure, then for every polytime $Dist_1$, the probability p of $Dist_1$ winning the game G_1 is no more than $1/2 + \frac{q}{2|D|} + \epsilon$, where ϵ is negligibly small (in the security parameter n).*

Proof (sketch): Consider a polytime $Dist_1$. We first argue that he cannot produce a password list PL containing pwd with probability significantly more than $q/|D|$. To prove this, we strengthen $Dist_1$ by allowing him choose k_i used in the calls to O_E. Then G_1 can be further simplified – $Dist_1$ does not need access to O_F (he can evaluate it himself). It is now easy to see that if $Dist_1$ can produce a list PL of q passwords with probability significantly more than $q/|D|$, he can be used to break the security of E (since he must have obtained some information about pwd from playing essentially the game of the CCA2 security.)

Now, return to the original $Dist_1$ and G_1. Let E_1 be the event of $Dist_1$ producing PL containing pwd, and E_2 be the event of $Dist_1$ winning by answering the challenge correctly. Then the probability of $Dist_1$ winning G_1 is $p = prob(E_1) + (1 - prob(E_1))prob(E_2|\neg E_1)$. Note that the lemma trivially holds for n, where $q \geq |D|$.

From now on, consider n, such that $q < |D|$ (q is polynomially bounded). Then, $Prob(\neg E_1)$ is bounded away from 0 by a polynomial (in n) fraction. We now show that for $Dist_1$, $prob(E_2|\neg E_1) < 1/2 + \epsilon_2$, where ϵ_2 is negligible. Suppose otherwise. Then we construct a polytime D' who, with the knowledge of pwd, answers the challenge of G_1 with probability significantly better than $1/2$. D' proceeds as follows. He runs $Dist_1$ up to the point when $Dist_1$ produces PL. D' checks whether $pwd \in PL$. If so, he flips a coin to answer the challenge. If not (and this happens non-negligibly often), he continues running $Dist_1$ (and obtains non-negligible advantage). At the same time, it can be easily shown by standard hybrid techniques that such D' cannot exist. Thus $prob(E_2|\neg E_1) < 1/2 + \epsilon_1$.

Therefore, if all the employed primitives are secure, $p = prob(E_1) + (1 - prob(E_1))prob(E_2|\neg E_1) < \frac{q}{|D|} + \epsilon_1 + (1 - \frac{q}{|D|})(1/2 + \epsilon_2) = 1/2 + \frac{q}{|D|} + \epsilon$.

□

A.2 Other cases

In all other cases, we reduce the KE game to a simpler variant G_2 of the game G_1.

Game G_2. *G_2 proceeds exactly as G_1 with the following two exceptions. First, the client's password pwd is revealed to the distinguisher $Dist_2$ as soon as $Dist_2$ sets the name C. Second, $Dist_2$ is not allowed to win by presenting PL (thus PL generation is omitted).*

Lemma 3. *If there exists an adversary Adv breaking the protocol Π that challenges a client and is given the long key ℓ and the password pwd, then there exists $Dist_2$ winning the game G_2 with probability non-negligibly greater than $1/2$.*

Proof (sketch): The construction of $Dist_2$ and the following discussion proceed almost identically to construction of $Dist_1$ of Lemma 1. Here we only point out the differences in construction and discussion.

- PL is not created nor used in any way by $Dist_2$.
- In Step 2, when the honest client is created, both the long key ℓ and the password pwd (obtained from G_2) are given to Adv.
- In Step 6 (Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C) $Dist_2$ proceeds like $Dist_1$, with the following exception. If e' (an encryption of (N'_C, pwd', k')), was not obtained from O_E, and the client name matches ($N'_C = N_C$), then instead of modifying PL, $Dist_2$ does the following. Recall, $Dist_2$ knows the password pwd of C. If $pwd' \neq pwd$, $Dist_2$ outputs P$\perp$, otherwise $Dist_2$ outputs $sid = (r_j, e')$. Recall, r_j is the message previously sent by S_j^C.
- $Dist_2$ handles a new type of request: *Adv sends a* challenge *request on a (completed and not failed or opened) client instance C_i^S of C:*
  Note that C_i^S previously received m_{C_i} and output $sid_i = (m_{C_i}, e_i)$. $Dist_2$ queries the challenge oracle $ch = O_C(i, m_{C_i})$, gives ch to Adv and submits Adv's output as the answer to the challenge of G_2. Note that there are restrictions on when $Dist_2$ is allowed to call O_C (O_F and O_C cannot be called with the same parameters). We argue later that we are not violating them.
- Request 10 (challenging a server instance) is now not allowed.

We note that all oracle calls made by $Dist_2$ are legal requests in G_2. The argument is also similar to that of Lemma 1. Indeed, as in construction of $Dist_1$, we always ensure that e was not generated by O_E before calling $O_D(e)$.

Further, O_F and O_C are only called when Adv opens or challenges instances, respectively. Consider the two possible cases (there are only two since Adv always challenges a client). First, Adv opened and challenged client instances C_{i_1} and C_{i_2}. Then, for the conflict to happen, it must be that $e_{i_1} = e_{i_2}$, which happens with negligible probability. Second, Adv opened a server instance S_j^C and a challenged a client instance C_i^S. For the conflict to happen, it must be that the client instance received r_j, and the server instance received e_i during the game. However, in this case, the sid output by the instances would match, and thus C_i and S_j would be partners, and Adv would not be allowed to challenge C_i^S and open S_j.

Now it is easy to see that the simulated messages provided by $Dist_2$ are distributed almost identically to those generated in a real execution. By assumption of the lemma, Adv wins with probability non-negligibly more than 1/2. It is easy to see that case $Dist_2$ wins whenever Adv wins, except for the negligible fraction of the time. Therefore, the constructed $Dist_2$ wins the game G_2 with probability non-negligibly more than 1/2.

□

Finally, we consider the adversary who is not given the long key ℓ, and is attacking the server.

Lemma 4. *Suppose the employed MAC scheme is secure. Then, if there exists an adversary Adv breaking the protocol Π who is not given the long key ℓ and is attacking the server, then there exists $Dist_2$ winning the game G_2 with probability non-negligibly greater than* $1/2$.

Proof (sketch): The construction of $Dist_2$ and the following discussion proceed almost identically to construction of $Dist_1$ of Lemma 1. Here we only point out the differences in construction and discussion.

- PL is not created nor used in any way by $Dist_2$.
- In Step 2, when the honest client is created, the long key ℓ is not revealed to Adv. The password pwd (obtained from G_2) is given to Adv.
- In Step 6 (Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C) $Dist_2$ proceeds like $Dist_1$. We note that e' was not obtained from O_E only with negligible probability (since otherwise we can construct a forger for MAC), and thus we don't handle the corresponding clause.
- In Step 10 (Adv sends a *challenge* request on a (completed and not failed or opened) server instance S_j^C of S:) $Dist_2$ proceeds like $Dist_1$. (Note that e' was not obtained from O_E only with negligible probability, due to the security of MAC; thus we don't handle the corresponding clause.)

We note that all oracle calls made by $Dist_2$ are legal requests in G_2. The argument is analogous to that of Lemma 1. Thus, the simulated messages provided by $Dist_2$ are distributed almost identically to those Adv sees in a real execution. By assumption of the lemma, Adv wins with probability non-negligibly more than $1/2$. It is easy to see that case $Dist_2$ wins whenever Adv wins, except for a negligible fraction of the time. Therefore, the constructed $Dist_2$ wins the game G_2 with probability non-negligibly more than $1/2$.

□

We now show that the adversary described in Lemmas 3 and 4 cannot exist, if secure schemes are used.

Lemma 5. *If the PRFG F and the CCA2 encryption scheme E used in G_2 are secure, then there does not exist a polytime $Dist_2$ winning the game G_2 with probability $p > 1/2 + \delta$, where δ is not negligibly small (in the security parameter n).*

The proof of Lemma 5 is done by a standard hybrid argument, and is omitted.

□