Password Mistyping in Two-Factor-Authenticated Key Exchange*

Vladimir Kolesnikov[1] and Charles Rackoff[2]

[1] Bell Labs, Murray Hill, NJ 07974,USA kolesnikov@research.bell-labs.com
[2] Dept. Computer Science, University of Toronto, Canada rackoff@cs.utoronto.ca

Abstract. We study the problem of Key Exchange (KE), where authentication is two-factor and based on both electronically stored long keys and human-supplied credentials (passwords or biometrics). The latter credential has low entropy and may be *adversarily* mistyped. Our main contribution is the first formal treatment of mistyping in this setting.
Ensuring security in presence of mistyping is subtle. We show mistyping-related limitations of previous KE definitions and constructions (of Boyen et al. [7, 6, 10] and Kolesnikov and Rackoff [16]).
We concentrate on the practical two-factor authenticated KE setting where *servers* exchange keys with *clients*, who use short passwords (memorized) and long cryptographic keys (stored on a card). Our work is thus a natural generalization of Halevi-Krawczyk [15] and Kolesnikov-Rackoff [16]. We discuss the challenges that arise due to mistyping. We propose the first KE definitions in this setting, and formally discuss their guarantees. We present efficient KE protocols and prove their security.

1 Introduction

The problem of securing communication over an insecure network is generally solved using *key exchange* (KE). KE provides partners with matching randomly chosen keys, which are used for securing their conversation. Of course, no adversary *Adv* should be able to mismatch players. Therefore, players must possess secrets with which they can authenticate themselves. The kind of secrets that are available to players determines the setting of KE. In the simplest KE setting players have a long shared random string. KE is more complicated if parties establish key pairs with the public keys securely published. Using weak and/or fuzzy credentials, such as passwords or biometrics, further complicates the design of KE. Finally, using a combination of credentials may make certain aspects of KE easier (such as incorporating password authentication), but increases the overall complexity of the solution, as discussed in [16].

Our setting. Two-factor authentication is critical and is used extensively in secure applications such as banking, VPN, etc. Stored long keys protect against online adversaries, but are vulnerable against theft. The extra layer of security is achieved with additional use of a theft-resistant credential, e.g. a short password or a biometric. Unfortunately, neither password nor biometric can be expected to be read reliably into the computer.

* A shorter version of this paper appears in ICALP 2008 [17]

We give foundation to this setting by generalizing the work of Halevi-Krawczyk (HK) [15] and Kolesnikov-Rackoff (KR) [16]. Recall, they address the client-server setting where both long key and a short password are used for KE. The servers are incorruptible, but client's card or password can be compromised.

Motivated by real scenarios, we study the effects of password mistyping. Mistyping need not be random, but may be skewed by the adversary, e.g. by technical means or social engineering manipulation. We thus consider security against adversaries who can *arbitrarily* affect user's mistyping. This consideration is especially relevant in case biometric credentials are used for authentication, since, due to technology limitations, biometric readings are *expected* to be misread.

Mistyping opens subtle vulnerabilities and raises complex definitional issues. In the sequel, we use terms "password" and "mistype", although our work applies to passwords, biometrics, and other short noisy credentials, as noted in Sect. 5.

1.1 Our contributions and outline of work

Our main contribution is the first formal treatment of mistyping of passwords in KE that uses a combination of credentials.

We discuss recent definitions that consider mistyping-related settings and issues – robust fuzzy extractors of [7, 6, 10]. We point out a limitation of the definitions of [7, 6, 10] with respect to robust handling of biometric misreading/mistyping and discuss possible remedies. We demonstrate and correct a vulnerability of the definition and protocol of [16], which can only be exploited when users mistype. These observations further emphasize the subtleties of mistyping and the need for its formal treatment and deeper understanding.

In Sect. 3, we introduce our setting and the framework of [16] which we build upon. Then, with simple protocols we illustrate mistyping-related issues, discuss natural definitional approaches to handling mistyping and their shortcomings. Most of the mistyping-related subtleties we uncover arise due to the simultaneous use of both long keys and passwords. In Sect. 4, we formalize our discussion in a definition, and prove that it prevents attacks that exploit mistyping.

In Sect. 5 we discuss applications of our work in biometric authentication.

In Sect. 6, we give efficient protocols and prove their security.

1.2 Related work

The problem of key exchange has deservedly received a vast amount of attention. Password KE was first considered by Bellovin and Merritt [4]. Foundations – formal definitions and protocols – were laid in [3, 8, 13, 9], and other works.

The use of combined keys in authentication, where the client has a password and the public key of the server, was introduced by Gong et al. [14] and first formalized by Halevi and Krawczyk [15]. Kolesnikov and Rackoff [16] extended this setting by allowing the client to also share a long key with the server, and gave first definitions of KE in their (and thus in the Gong et al. and HK) setting.

Password mistyping in KE. Despite the large research effort, the definitional issues of KE password mistyping are formally approached only in the

UC definition of Canetti et al. [9]. In their password-only setting, mistyping is modelled by Environment $\mathcal{Z}$ providing players' inputs. Additional use of long key makes our setting significantly different (and more subtle with respect to mistyping) from that of [9]. Mistyping was also considered in different settings: related-key attacks on blockciphers [2] and signing authority delegation [18].

Biometric authentication and fuzzy extractors. A growing body of work, e.g. [12, 5, 10, 11], addresses the use of biometrics in cryptography. Boyen et al. [7, 6, 10] consider its application to KE. They introduce the notion of *robust fuzzy extractor* (RFE), and give generic constructions of biometric-based KE from RFE. While their setting is similar to ours, the problems solved by [7, 6, 10] are different. They give KE protocols that accept "close enough" secrets, thus enabling security and privacy of biometric authentication. They do not aim to give a formal KE definition that handles biometric/password misreading. Moreover, as shown in Sect. 2, their notion of RFE is insufficiently strong to guarantee security of their generic KE protocol in many practical settings. (However, instantiating their KE protocol with their RFE construction is secure, since the latter satisfies stronger requirements than required by the definition.)

2 Mistyping-related limitations in previous work

On robust fuzzy extractor (RFE) definition and KE protocol [7, 6, 10]. We first clarify underlying biometric technology limitations and assumptions. Biometrics are "fuzzy", i.e. each scan is likely to be different from, but "close" to the "true" scan. Error-correction [12] is then used to extract non-fuzzy keys usable in cryptography. However, error-correction cannot correct many misreading errors (up to 10%), since this would imply high false acceptance rate[3]. Thus misreading beyond error-correction ball occurs often, and must be considered.

We note a limitation of RFE definition [7, 6, 10], prohibiting its use with the generic KE construction (Sect. 3.3 of [7]) in many scenarios. Roughly, definition's domains of correctness and security guarantees coincide. That is, extracted randomness is only guaranteed to be good if the scan is within the *error-correction* distance t from the original. There are no guarantees on the randomness if this condition does not hold. This is, perhaps, due to the papers' implicit assumption that "natural" misreadings are almost always "close" and are corrected (i.e. FRR is negligible). However, as discussed above, this assumption often does not hold. Strengthening the randomness guarantees of RFE would increase its usability.

More specifically, a RFE (Gen, Rep) may exhibit the following vulnerability. Given the public helper string P, if the biometric w_0 is misread in a special way w' outside the error-correction ball, the extracted randomness $Rep(w', P)$ is predictable. Even more subtly, $Rep(w', P)$ and $Rep(w_0, P)$ could be related, but unequal. Clearly, KE protocols, including one of Sect. 3.3 of [7, 6], constructed from such RFE would not be secure. One solution is to require, for w' outside the

[3] In balanced optimized real-life systems, which compare scans directly, False Reject Rate (FRR) is usually 1..10%. Notably, NIST reports FRR of fingerprints 0.1..2%, iris 0.2..1% and face 10%. See [1] for comprehensive overview and references.

error-correction ball, that either $Rep(w', P) = \perp$ (property of RFE construction of [7, 6]) or that $Rep(w', P)$ is either equal to or independent from $Rep(w_0, P)$.

Finally, although [7, 6, 10] consider adversarial substitution of P with P', they guarantee $Rep(w', P') = \perp$ only for w' in the error-correction ball. This vulnerability also can be resolved by separating the error-correction and security domains. We defer detailed definition, analysis and constructions as future work.

On the definition and construction of [16]. We present the following practical outside-of-the-model mistyping attack on the protocol (and thus also on the definition) of Kolesnikov and Rackoff [16]. Specifically, resistance to Denial of Access (DoA) attacks of the protocol of [16] is compromised if the honest client ever mistypes. Indeed, since their protocol is not challenge-response, client C's message can be replayed. This is not a problem if C always types the correct password (session keys of C and server S will be independent). However, if the password was mistyped, both the original and replayed message will cause S to register password failure, violating the intent of the DoA resistance. We stress that the KR protocol is otherwise secure against mistyping (and we prove it in Sect. 6). Our definitions and protocols address and correct the above insecurity.

Above limitations show subtleties of mistyping and the need to address them.

3 Pre-definition discussion

Our main contribution is a formal treatment of mistyping in the combined keys KE setting of Kolesnikov and Rackoff [16]. The KR setting is a generalization of the Halevi-Krawczyk setting [15], in which clients have a password and the public key of S. In KR setting, clients carry stealable cards capable of storing cryptographic keys – public key of S and long key ℓ shared by C and S. Addition of the cards allows better functionality and security than that of HK. KR definitions and protocol guarantee and achieve strong security when C's card is secure, and weaker, password-grade, security, when the card is compromised.

We stress that the definition of KR does not handle mistyping. That is, it is possible to construct KR-secure protocols that "break" if the client ever mistypes his password. Sect. 3.3 of [16] provides an example and a short informal discussion on mistyping, and leaves the problem open. In Sect. 3.2, we expand this discussion, present more subtle mistyping threats, and discuss approaches to handling them. This leads to the presentation of our definitions in Sect. 4.

Notation. We concentrate on the two-factor authentication setting, where a client (denoted C) exchanges keys with a server (S). Both long and short keys are used for KE. 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. Denote the adversary by Adv. Sometimes we distinguish the game and real-life adversary, and denote the latter Adv_{Real}. Denote C's password by pwd and long key by ℓ. S's public/ private keys are pk_S and sk_S. Password failure and the associated control symbol output by S is denoted by P$\perp$.

On the Style of Definition. We chose the game (Bellare-Pointcheval-Rogaway [3]) style, since this allowed using the intuitive definition of KR (only existing two-factor-authentication KE definition). Extending KR allowed reduction of security claims of our definition/setting to those of KR. Further, the stronger and arguably more intuitive UC model unfortunately is sometimes too strict, ruling out some efficient protocols which appear to be good enough in practice.

Proposing a simulation-based (especially, UC) definition, and exploring the relationship between it and our definition would add confidence in both our and the UC treatment of the problem. We thus leave as an important next step the design, detailed analysis and comparison of a corresponding UC definition. We expect that our discussions of ideas and obstacles would aid in this future work.

3.1 Review of the framework of [16]

Our definition is an extension of the KE definition of KR (Def. 2 of [16]).

Recall, KR (and thus our) definition follows the common game-based paradigm. The real world and real adversary Adv_{Real} are abstracted as a game, played by the game adversary *Adv*. Game includes clients and servers – Interactive Turing Machines (ITM) running the KE protocol Π, communicating via channels controlled by *Adv*. Game rules mimic reality, and are designed so that *Adv*'s wins correspond to real-life breaks. Π is defined secure if no polytime *Adv* is able to win above certain "allowed" probability. Definition is thus reduced to the design of the game. KR break down the real world into five intuitive games (KE_1, KE_2, KE_3, DOA and SID), which mimic possible real-life attack scenarios.

Game KE_1 is the core of the definition; it addresses password security when the long key is compromised. The difficulty of KE_1 design is in balancing the power given to *Adv*, since Adv_{Real}'s non-negligible advantage must be accounted exactly. It is achieved by "charging" *Adv* for each active attack (i.e. P$\perp$ output by S). The allowed *Adv* win probability is a function of the number of charges.

KE_2 models Adv_{Real} posing as S to C. KE_3 models KE with uncompromised card. In both cases, *Adv* is allowed only negligible success, which is easy to model. DOA models a "denial of access" attack formalized by KR, which requires that *Adv* is not able to cut C's access to S by exhausting allowed password failures. Finally, SID is a game preventing technicality-based insecure protocols.

We stress that a good model need not mimic the world *exactly*. E.g., *Adv*'s ability to mistype or to know whether S failed may be different from Adv_{Real}'s, as long as *Adv* can win in *some* way (only) against bad protocols.

Mistyping in KR definitions. In KR games, client ITMs are always instantiated with correct password, which limits *Adv*'s ability to emulate mistyping. Many real-life attacks that exploit mistyping cannot be carried in the game, allowing vulnerable protocol to withstand *Adv*'s attacks and be defined secure. In Sect. 3.2, we discuss vulnerabilities, some natural "fixes" and their limitations.

3.2 Natural definitional approaches to mistyping (that don't work)

To better expose subtle definitional issues and the limitations of some natural approaches, we build presentation incrementally. We propose several mistyping-vulnerable protocols, each progressively more "tricky", and show that they are KR-secure. We then discuss corresponding natural "fixes" of the KR definition – ways of allowing *Adv* to modify or substitute client's password, so as to mimic real-life mistyping and allow *Adv* to carry the real-world attacks. We show that ultimately they are insufficient and conclude that, for technical reasons, direct mimicking of mistyping in the games does not result in a good model. For readability, we keep discussion brief and informal (but readily formalizeable).

Mistyping vulnerabilities by example. Let Π be a KR-secure KE protocol. Π_1, Π_2, Π_3 below are KR-secure, but fail in progressively more subtle ways.

$\mathbf{\Pi_1}$ *(S leaks long key upon mistyping).* Let Π_1 be a protocol as Π, except that in Π_1 S reveals the long key ℓ in a message, once password failure P$\perp$ occurred.

Clearly, Π_1 is "bad". But, it is easy to see that Π_1 is secure by KR definition. Since instances of C never mistype in the game, KR *Adv* cannot cause P$\perp$ without possession of ℓ. Thus, *Adv* cannot gain from S revealing ℓ, and Π_1 is KR-secure.

$\mathbf{\Pi_2}$ *(S leaks password upon repeated mistyping).* Let *pwd* be C's password.Let Π_2 be a protocol as Π, except that in Π_2, S reveals *pwd* once $pwd+1$ was tried twice. (Limited global state can be communicated among instances of S with the help of *Adv*, thus allowing Π_2 [16]; see Appendix A for detailed discussion.)

At the first glance, it may appear that Π_2 is "good". Indeed, the advantage *Adv* gets from causing the leak is canceled by the effort to obtain it – a redundant password attempt for each attempt of causing the leak (this is the reason why Π_2 is KR-secure). However, this leak can be caused by real-life honest C mistakenly entering $pwd + 1$ twice. This is not an unusual situation, and the resulting password compromise is clearly unacceptable.

$\mathbf{\Pi_3}$ *(S leaks a small hint about a password upon repeated mistyping).*Let *pwd* be C's password. Let Π_3 be a protocol as Π, except that in Π_3, S reveals whether $pwd = 0$ once $pwd + 1$ was tried 4 times. Π_3 is bad for the same reason as Π_2.

Definitional approaches. We consider strengthening *Adv* of KR by mimicking powers of real-life adversary. Our goal is to disallow above "bad" protocols.

Allowing *Adv* to specify the password of C's instances disqualifies Π_1.Indeed, *Adv* wins the game where he is not given ℓ, as follows. He instantiates C with a wrong password, causing P$\perp$ and leak of ℓ, which *Adv* uses to win.

To disqualify Π_2, *Adv* needs more than simple substitution of C's password. *Adv* needs the power to specify a "mistyping function" applied to the password given to C (idea also considered in [18]). That is, *Adv* specifies a map $F : D \mapsto D$, and C is instantiated with password $F(p)$. (Not every map F is allowed [16].)

While Π_3 is bad for the same reason as Π_2 (real-life C's mistyping leaks a password hint), it is harder to disqualify Π_3 due to the small size of the leak. It turns out that Π_3 is an important example, showing that allowing *Adv* to influence C's input is insufficient. We continue this discussion below in Sect. 4.

4 Mistyping-secure KE definition

Π_3, the last example of Sect. 3.2 is a (otherwise secure) protocol where S leaks a small password hint after four certain *repeated* mistypings. A repeated mistyping does not help *Adv* (he is checking already checked password). Since in KR definition, *Adv* is charged for each (even repeated) mistyping, the cost of mistyping outweighs the benefit of the leak, and *Adv* is not able to exploit the vulnerability.

This leads to our main idea – to allow *Adv* to run mistyped KE executions "for free". This way, *Adv* will be able to win whenever a non-negligible amount of information is leaked due to mistyping. It turns out that this additional power, applied properly, results in a good (i.e. sufficiently, but not too strong, and easy to use) definition, presented in this section.

Our extension of KR definition. We would like to give *Adv* the ability to observe and actively participate "for free" in mistyped KE sessions. This is not possible with the approaches we previously discussed, including that of [16]. This is because there *Adv* always learns whether S accepted the password, allowing *Adv* to verify a password guess, for which *Adv* must be charged. Our idea is to withhold failure information from *Adv* (and not charge him in case of P⊥) by default, thus allowing "free" mistypings. If *Adv* wants to obtain failure information, it is given to him upon special "check" request. Since this gives him information about the password, he is charged one attempt, if the check reveals P⊥. Note, this cost structure is a simple generalization of the one used in [16]. This amendment of KR is sufficient to handle mistyping.

Another advantage of this approach is allowing to mimic mistyping without *Adv* creating instances with substituted password. Indeed, *Adv* can make a password guess, and, based on it, emulate any mistyping sequence of C. As shown in Sect. 4.1, this guarantees security, since a "free" mistyping-dependent leak would confirm *Adv*'s guess, allowing him to win. On the other hand, C's input substitution, especially using a mistyping map, is technically complex, and makes the definition less usable, since proofs would have to consider all such maps.

We now present our definition. Let n be a security parameter, and $D = \{0,1\}^m$ is the password domain. (In general, m can be a function of n; interesting cases are when m is constant or logarithmic in n.) All players (*Adv*, C, S) are p.p.t. machines. As does [16], we use session IDs (SID) to partner instances of players, and impose the following correctness requirement. In the absence of adversary, all sessions terminate and intended parties output same *sid* and key.

Definition 1. *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.*

We start by presenting KE games, which model attacks of a real-life adversary Adv_{Real}. The first game models the setting where Adv_{Real} obtained C's long key, is attacking a server, and is allowed a limited number of password tries.

Game KE$_1$. *Adv deterministically chooses active attack threshold $q \in 1..|D|$ (based on security parameter n) and creates an (honest) server S. Adv chooses S's name; then S's public/private keys are set up, and the public key revealed to Adv. Adv then runs players by executing steps 1-7 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, his partner'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. The instance immediately responds with a reply (by giving it to Adv) and/or, terminates and outputs the result (a sid and either the session key, the* failure symbol $\perp$*, or, in case of the server instance, the* password failure symbol $P\perp$*) according to the protocol. Adv learns only the sid part of the output.*
6. *Adv "checks" any completed honest instance – then he is notified whether the instance output $P\perp$, $\perp$, or a session key. Adv gets charged one attempt, if he checked S^C and it output $P\perp$.*

   When Adv accumulates q charges, he becomes restricted – he can neither deliver messages to any instances S_j^C nor check any instances.
7. *Adv "opens" any successfully completed and checked honest instance – then he is given the session key output of 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, been checked by Adv, and output a session key. 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 2-7). 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 that we handle *sid* differently from [16]. Here we insist that parties always output *sid*, while previously *sid* was only output if a party did not fail. We need this change, since KE_1's interface needs to be the same for cases when an instance failed and did not fail. Outputting a *sid* only if KE succeeded (and letting it known to *Adv* for free) helps *Adv* determine whether $\mathrm{P}\perp$ occurred.

In all other KE games ($\mathbf{KE}_2$, $\mathbf{KE}_3$, SID and DOA) below, password mistyping and even the knowledge of *pwd* should not help *Adv*. We thus choose to reveal the password to *Adv* and remove restrictions on the number of $\mathrm{P}\perp$'s (thus removing the definition of q). We also allow *Adv* to specify C_i's password at its instantiations. These games are presented by modifying the above KE_1. All of the above four modifications are included in all games below.

KE_2 models the setting where *Adv* stole C's *pwd* and ℓ, but is attacking C.

Game $\mathbf{KE}_2$ *is derived from KE_1 as noted above; further, Adv is given ℓ and must challenge an honest client instance C_i^S.*

KE_3 models the setting where *Adv* only stole C's *pwd*, and is attacking S.

Game $\mathbf{KE}_3$ *derived from KE_1 as noted above, but Adv is not given ℓ.*

SID enforces non-triviality, preventing improper partnering (e.g. players unnecessarily outputting same *sid*). Recall, *Adv* is not allowed to challenge parties whose partner has been opened; SID ensures that *Adv* is not unfairly restricted.

Game SID *is derived from* KE_1 *as noted above; further, Adv does not ask for (nor answers) the challenge. Adv wins if any two honest partners output different session keys.*

Note, SID allows for one (or both) of the partners to output a failure symbol. *Adv* only wins if two successfully completed parties output different session keys.

Finally, game DOA models resistance to the Denial of Access (DoA) attacks. This game prevents vulnerabilities due to mistyping (see Sect. 4.1).

Game DOA *is derived from* KE_1 *as noted above; further, Adv does not ask for (nor answers) the challenge. Adv wins if the number of* $P\bot$*'s is greater than the number of client instances where he substituted the password.*

Definition 2. *We say that a key exchange protocol* Π *is* secure in the Combined Keys model with mistyping, *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.*

The definition for the HK setting (where C does not have ℓ) is extracted from Def. 2 by removing all uses of ℓ and the games where *Adv* doesn't know ℓ.

4.1 Why this is a good definition

First, since *Adv* is not weaker than *Adv* of [16], Def. 2 enforces basic security properties of the protocols. We additionally need to argue that the definition is not too strict and that it prevents mistyping-caused leaks in protocols. The former property is intuitive, and we support it by proposing an efficient protocol and proving its security w.r.t. Def. 2 (Sect. 6). The latter property, on the other hand, requires significantly more careful consideration, presented in this section.

Note, KE_1 is the only game where we need to be careful with not giving *Adv* too much power w.r.t. mistyping. In other games, unlimited ability of *Adv* to substitute C's input should not help him win against a secure protocol. At the same time, such *Adv* directly models real-life adversary. Therefore, this simple allowance resolves mistyping problems w.r.t. other games we consider.

KE_1 is the core of the definition, and most of the definitional subtleties appear in KE_1. We start with the discussion of the details and ideas about this game.

Why $\mathbf{KE_1}$ is a good model. Often, when a definition is proposed, a proof is provided, demonstrating the relationship between the new and previous definitions. This adds confidence in the proposed definition. We introduce the first definition in our setting; thus there is no previous definition to relate it to.

Our approach. Instead, we prove that if a protocol Π is secure by Def. 2, *Adv* of the game KE_1 cannot tell the difference between the following two executions, if he is not allowed to see the outputs of S. In one execution, selected (by *Adv*) client instances are instantiated with a mistyping sequence *Adv* chooses, and in the other they are instantiated with the password *pwd* of C. We stress that *Adv* is active during these executions; he can perform (almost) all the actions *Adv* of KE_1 can. This provides an informal "reduction" to the definition of [16], in the following sense. Assume the definition of [16] is "good", i.e. accurately identifies insecure protocols in its "no-mistyping" model. Then Def. 2 is "good" in the general setting, where clients are allowed to mistype.

Indeed, suppose Π is "bad". Due to the indistinguishability of the above executions, anything that Π leaks due to mistyping can also be seen and exploited without mistyping by *Adv* of KE_1 of [16]. Then Π will be insecure by definition of [16], since, by assumption, it is a good definition. Since KE_1 *Adv* of Def. 2 is at least as strong as that of [16], Π will also be insecure by Def. 2. From another angle, if active *Adv* cannot distinguish the above executions, then he is not learning anything from the mistypings, other than what may be inferred from the corresponding sequence of P$\perp$'s, but the latter is unavoidable anyway.

This reduction is informal, and serves only as evidence that our definition is good. By the nature of definitional work, it is not possible to "prove" definitions.

Formal theorem statement and proof of indistinguishability of the above executions is in Appendix B. Proof idea is that some passwords used in the mistyped execution must be unequal to C's *pwd*. Ability to distinguish executions gives a free hint of what *pwd* is not, allowing corresponding KE_1 *Adv* to win.

On DoA protection. As mentioned in Sect. 2, the definition of [16] does not model (and fails to guarantee) DoA resistance when honest users mistype. We need that a replayed client's flow must not cause S output P$\perp$. Therefore, C must send at least one message that is dependent on S's message. Thus, the one-round, two-independent-flow protocols are not possible if DoA is desired.

We change the DOA game accordingly. *Adv* knows *pwd*, and is now allowed to instantiate clients with passwords of his choice. *Adv* wins DOA, if the number of P$\perp$ is greater than the number of client instances with substituted password.

5 Application to biometric authentication

We note that our definitions and protocols are directly applicable to biometric-based authentication. For example, fuzzy extractors [11] can be naturally used in our two-factor authentication setting, as follows. The storage card now additionally contains the public data pub_C of C's biometric b_C. The (potentially short) randomness extracted from b_C plays the role of the password. To authenticate, C first reconstructs the password using extractor's recovery procedure $Rec(pub_C, b'_C)$, and then uses it as prescribed by a KE protocol. Misreading b'_C of b_C can cause variety in the output of *Rec* and thus effect mistypings in the protocol. Still, our definitions (in-particular, mistyping-security property) and properties of fuzzy extractors guarantee security of this construction, even if

Adv captured the card with the long key and pub_C. (In the HK setting, where C only has pk_S, we also can use our definition and above protocol – but pub_C is now sent by S to C authenticated by S's signature, as part of the protocol.)

However, we note that our definitions do not handle the general case, where b_C is used directly as input to C. That is, S knows "acceptance set" of C (AS_C), and accepts if C's submitted password/biometric $b_C \in \mathrm{AS}_C$. We anticipate that a natural extension of our definition would handle this case. In particular, the correctness requirement should be amended w.r.t. AS_C, and *Adv*'s allowed success rate may be dependent on AS as well. We leave this definition as future work, to be performed either as extension of our definition, or in the UC framework.

6 Mistyping-Secure KE Protocols

WLOG, assume protocol messages are formed properly (i.e. values drawn from appropriate domains, etc.). Let n be a security parameter, $E = (Gen, Enc, Dec)$ 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 $N_C \in \{0,1\}^n$ be the name of client C. (Shorter names may be used.)

Although KR definitions do not handle mistyping, their protocol resists all mistyping-related attacks, except for (perhaps, unimportant in some settings) DoA resistance. We first prove this fact. Constr. 1 is the protocol of [16], only with updated handling of *sid*, to satisfy the syntactic requirements of Def. 2.

Construction 1 *(KE with mistyping, no DoA resistance [16])*

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

Theorem 1. *Constr. 1 satisfies Def. 2, except for the success rate in game DoA.*

We now present a fully secure protocol in our model, derived from Constr. 1.

Construction 2 *is a challenge-response version of Constr. 1, where* C^S *replies with* $(\alpha, MAC_\ell(r, \alpha))$ *to message* r.

Theorem 2. *Constr. 2 is secure by Def. 2.*

We note that Constr. 2 can be modified to allow S to send confirmation to C whether he accepted, failed or password-failed. See Appendix D for details.

Proofs of security of Theorems 1 and 2 are presented in Appendix C.

Achnowledgements: We thank Shai Halevi, Hugo Krawczyk, and anonymous referees for valuable comments.

References

1. http://en.wikipedia.org/wiki/Biometrics#Performance,Retrieved 02/10/08.
2. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 491–506, 2003.
3. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.
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. Xavier Boyen. Reusable cryptographic fuzzy extractors. In *CCS*, pages 82–91. ACM Press, 2004.
6. Xavier Boyen, Yevgeniy Dodis, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Secure remote authentication using biometric data (revised version). Available at http://www.cs.stanford.edu/~xb/eurocrypt05b/.
7. Xavier Boyen, Yevgeniy Dodis, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Secure remote authentication using biometric data. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 147–163, 2005.
8. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-hellman. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.
9. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, 2005.
10. Yevgeniy Dodis, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *LNCS*, pages 147–163, 2006.
11. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. Cryptology ePrint Archive, Report 2003/235, 2003. http://eprint.iacr.org/.
12. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540, 2004.
13. Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *LNCS*, pages 408–432, London, UK, 2001. Springer.
14. Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
15. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.*, 2(3):230–268, 1999.
16. Vladimir Kolesnikov and Charles Rackoff. Key exchange using passwords and long keys. In *Theory of Cryptography, TCC 2006*, volume 3876 of *LNCS*, pages 100–119. Springer, 2006.
17. Vladimir Kolesnikov and Charles Rackoff. Password mistyping in two-factor-authenticated key exchange. In *ICALP (2)*, pages 702–714, 2008.
18. Philip MacKenzie and Michael Reiter. Delegation of cryptographic servers for capture-resilient devices. *Distributed Computing*, 16(4):307–327, 2003.

A On global state communication via *Adv*

We note that the KR definition does not allow communication between instances of S (or of C) directly. Such abstraction allows for a significant simplification in many KE settings. However, as pointed out in [16], in the setting with long secret keys, such as ours, instances of S can have limited private communication via the following side channel provided by active *Adv*. To send a message m, an instance of S encrypts and signs it with its public key pair and gives to *Adv*, who can later pass it to another instance of S in a special protocol field. Thus, a weak private authenticated channel (with *Adv* being able to only drop, replay and re-route messages) can be established.

In particular, such channel can implement a global counter of password failures P⊥. This can be done by each instance checking the special protocol field for current encrypted P⊥ counter, and sending out a signed encryption of the new value upon termination. Similarly, a global history of events, such as a sequence of passwords that S-instances saw, can be maintained by them.

We stress that this channel is the consequence of the setting, and is *not* a desired "feature" of the KR or our definitions. Indeed, *Adv* can always cut this channel (or just drop some messages), so it cannot be reliably used. Moreover, as discussed in Sect. 3.2, it introduces certain subtleties preventing the obvious ways of definitional handling of mistyping. Jumping ahead, we note that the subtleties caused by this channel are specific to the combined keys setting, where both long and short keys are used (long keys are necessary to facilitate semi-reliable communication between instances of S, and short keys require precise handling of *Adv*'s advantage, and his very limited attack powers in the games). We would like to prohibit this channel to simplify the model, but we don't see an elegant way to do it in a definition.

Indeed, while it is relatively easy to verify that a concrete protocol prohibits such channel, it is not at all clear how to enforce it in a definition. In particular, we would need to prevent an S instance from processing a message related to one generated by another S instance. It is unclear how to even define such relations. Another approach could be to "look inside" the protocol source code, and limit the uses of the sk_S. For example, if sk_S is not used to verify signatures, this may disrupt certain inter-instance communication. However, such white-box analysis is complex and may result in unnecessary restrictions on the protocols. We thus did not take this approach.

B Proof of Indistinguishability of Transcripts with and without Mistyping

We now formally prove the indistinguishability of the two executions, discussed in Sect. 4.1. Consider the following game KE_M derived from KE_1 of Def. 2.

Game $\mathbf{KE}_M$. *KE_M is the same as KE_1, with the following differences.*

- *Adv does not specify or use q. He is not allowed to check, open or challenge instances of S.*

– *Adv deterministically (based on the security parameter* n*) chooses the length of the distinguishing sequence* l_d*.*
– *A bit* b *is chosen uniformly at random by the game.*
– *Among client instances* C_i*, throughout the execution, Adv creates* l_d *challenge instances, as follows. When creating* C_i *which is a challenge instance, Adv chooses* $p_i \in D$ *and gives it to the game. If* $b = 1$ *then* C_i *is instantiated with* p_i*, otherwise, it is instantiated with its password pwd.*
– *Adv outputs a bit* b'*. He wins if* $b' = b$ *and loses otherwise.*

Adversary Adv_M of KE_M is active and strong – he has access to almost all the information *Adv* may see during the execution of KE_1. Adv_M can even see session keys output by the client (and thus session keys of partner server instances). Of course, we must prevent him from verifying if S output P$\perp$, since otherwise he would always win. We thus disallow Adv_M to check S. We prove the following.

Theorem 3. *Let* Π *be a protocol secure by Def 2. Then, no polytime adversary* Adv_M *wins* KE_M *with probability non-negligibly more than* $1/2$*.*

Note, Theorem 3 implies that *Adv* cannot tell the difference between any two sequences he can pick (otherwise at least one of them would differ from the sequence of all correct passwords). The theorem does not (and cannot) state there does not exist a sequence which would cause a distinguishable transcript. In fact, distinguishable sequences may exist, e.g. those depending on the private key of the server, but Adv_M cannot find them.

B.1 Reduction $\mathbf{KE_1}$ to $\mathbf{KE}_M$

For the proof, first consider an adversary who is only allowed to provide a single challenge (i.e. $l_d = 1$). We will later show (via a hybrid argument) that this restriction is easily removed. Consider the following game KE_{M1}.
Game $\mathbf{KE}_{M1}$**.** *KE_{M1} is the same as KE_M, with the exception that* $l_d = 1$*.*

Theorem 4. *Let* Π *be a protocol secure by Def. 2. Then, no adversary* Adv_{M1} *wins* KE_{M1} *with probability non-negligibly greater than* $1/2$*.*

Proof (sketch):
Suppose there exists an adversary Adv_{M1} who wins KE_{M1} with probability $Pr_w = 1/2 + \delta$, where δ is non-negligible. Consider success rate of Adv_{M1} in two cases – when he chooses $p_i = pwd$ and when he chooses $p_i \neq pwd$. Clearly, since the view of Adv_{M1} is independent of b in the first case, his success rate is equal to $1/2$. Thus, the success probability Pr_{w1} in case when $p_i \neq pwd$ must satisfy $Pr_{w1} = Pr(Adv_{M1} \text{ succeeds}|p_i \neq pwd) \geq Pr_w$, and $Pr(p_i \neq pwd)$ is non-negligible.

We show how to use Adv_{M1} to construct *Adv* who wins KE_1 with probability non-negligibly greater than allowed. *Adv* proceeds as follows. He choses $q = |D| - 1$. Note that a secure protocol would allow probability of *Adv*'s success of $\frac{1}{2} + \frac{q}{2|D|} = \frac{1}{2}\frac{2|D|-1}{|D|} = 1 - \frac{1}{2|D|}$. Adv then runs Adv_{M1}, satisfying all his requests in a natural manner, except when Adv_{M1} requests to create the challenge instance and provides a password p_i. In this case, Adv flips a uniform bit b_A and instantiates C_i with the password according to b_A. That is, if $b_A = 0$, C_i is instantiated, and if $b_A = 1$, Adv simulates execution of C_i with a password p_i. Note that Adv will never check S while running Adv_{M1}, so this execution is "free" for Adv. Note that what Adv_{M1} sees is distributed identically to what he would have seen in the game KE_{M1}. Eventually Adv_{M1} outputs a value, which indicates its guess at b_A. Recall, Adv_{M1} has an advantage by our assumption. Adv will use this advantage in a natural way, as follows.

Adv now makes $q - 1 = |D| - 2$ random attempts of the password pwd of the honest client C, created during running Adv_{M1}. If Adv succeeds in the guess, he wins KE_1. Let E_{ng} be the event that Adv does not succeed in guessing pwd, occurring with probability $\frac{2}{|D|}$. Conditioned on occurrence of E_{ng}, allowed probability Pr_{ng} of success of Adv is $\frac{3}{4}$. (This is obtained by solving $1 - \frac{1}{2|D|} = (1 - \frac{2}{|D|}) \cdot 1 + \frac{2}{|D|} \cdot Pr_{ng}$.) For the rest of the proof, we consider the case where E_{ng} occurred. We show how Adv wins with probability greater than $\frac{3}{4}$, thus proving the theorem.

If p_i has been tried by Adv, Adv proceeds with his last allowed password guess, and wins with the allowed expected probability $\frac{3}{4}$. However, if p_i has not been tried (call this event E_{np}), Adv makes a guess at whether $p_i = pwd$, based on the output of Adv_{M1}, as follows. If Adv_{M1} succeeded in guessing b_A, Adv guesses that $p_i \neq pwd$, otherwise he guesses that $p_i = pwd$. Compute the probability of Adv's guess being correct, conditioned on events E_{ng}, E_{np} having occurred.

$$\begin{aligned} Pr(Adv \text{ correct}|E_{ng}, E_{np}) = {} & Pr(Adv \text{ correct}|p_i = pwd, E_{ng}, E_{np}) \cdot Pr(p_i = pwd|E_{ng}, E_{np}) \\ & + Pr(Adv \text{ correct}|p_i \neq pwd, E_{ng}, E_{np}) \cdot (1 - Pr(p_i = pwd|E_{ng}, E_{np})) \end{aligned} \tag{1}$$

In the case $p_i = pwd$, Adv_{M1}'s view is independent of b_A, and his probability of winning is 1/2. Therefore, $Pr(Adv \text{ correct}|p_i = pwd, E_{ng}, E_{np}) = \frac{1}{2}$. It is easy to verify that in the case $p_i \neq pwd$, Adv guesses correctly whenever Adv_{M1} guesses correctly. Further, it is not hard to see that in case $p_i \neq pwd$, Adv_{M1}'s success does not depend on events E_{ng} or E_{np}. Therefore,

$$\begin{aligned} Pr(Adv \text{ correct}|p_i \neq pwd, E_{ng}, E_{np}) &= Pr(Adv_{M1} \text{ wins}|p_i \neq pwd, E_{ng}, E_{np}) \\ &= Pr(Adv_{M1} \text{ wins}|p_i \neq pwd) = Pr_{w1} \end{aligned}$$

Substituting into 1 and denoting $pr_{eq} = Pr(p_i = pwd|E_{ng}, E_{np})$, obtain

$$Pr(Adv \text{ correct}|E_{ng}, E_{np}) = \frac{1}{2}pr_{eq} + Pr_{w1}(1 - pr_{eq})$$

Since $Pr(p_i \neq pwd)$ must be non-negligible for Adv_{M1} to have a non-negligible advantage, $Pr(p_i \neq pwd|E_{ng}, E_{np}) = 1 - pr_{eq}$ is also non-negligible. Since Pr_{w1} is non-negligibly greater than $\frac{1}{2}$ by assumption and $(1 - pr_{eq})$ is non-negligible, $Pr(Adv \text{ correct}|E_{ng}, E_{np})$ is non-negligibly greater than $\frac{1}{2}$. Thus,

$$Pr(Adv \text{ wins } \mathrm{KE}_1|E_{ng}, E_{np}) = Pr(Adv \text{ correct}|E_{ng}, E_{np}) + (1 - Pr(Adv \text{ correct}|E_{ng}, E_{np})) \cdot \frac{1}{2}$$

is non-negligibly greater than $\frac{3}{4}$. Therefore $Pr(Adv \text{ wins } \mathrm{KE}_1|E_{ng})$ is non-negligibly greater than $\frac{3}{4}$. This completes the proof of the theorem. □

Theorem 5. *Let Π be a protocol. If there exist an adversary Adv_M winning KE_M with probability non-negligibly greater than $1/2$, then there exist an adversary Adv_{M1} winning KE_{M1} with probability non-negligibly greater than $1/2$.*

Proof (sketch):

Let Adv_M be an adversary satisfying the theorem's condition. Let pr_0 be the probability Adv_M wins (i.e. outputs 0) if $b = 0$ in KE_M, and pr_1 be the probability Adv_M wins (i.e. outputs 1) if $b = 1$ in KE_M. We will construct Adv_{M1} guaranteed by the theorem statement, by a simple hybrid construction. Intuitively, since there is a noticeable difference in the effects of the two mistyping sequences Adv_M generates (string of p_i's and string of passwords of C_i), there must be a noticeable difference in the effect of at least one consecutive pair of the sequence of hybrid strings. Moreover, the expected difference is noticeable in a randomly chosen such pair. Details below.

Adv_{M1} starts and runs Adv_M, passing Adv_M's requests to KE_{M1}. All requests of Adv_M are directly satisfied, with the exception of the challenge requests, since Adv_{M1} is only allowed to make one such request. Adv_{M1} randomly chooses $j \in_R [1..l_d]$. Adv_M's k-th request ($k \in [1..l_d]$) for mistyped clients creation is satisfied as follows. If $k < j$, the corresponding C_i is instantiated with its password *pwd*. If $k > j$, the corresponding C_i is instantiated with password p_i, supplied by Adv_M. If $k = j$, Adv_{M1} supplies p_i to KE_{M1}, and the game chooses its challenge by instantiating the corresponding C_i either with *pwd* or p_i. When Adv_M terminates, Adv_{M1} outputs what Adv_M output and terminates. It is not hard to verify that Adv_{M1} wins KE_{M1} non-negligibly more often than 1/2. The exact probability computation follows.

Consider the possible hybrid mistyping sequences caused by Adv_{M1} during the execution of KE_{M1} while running Adv_M. The actual sequence instantiated depends on the chosen j and the bit b chosen by KE_{M1}. There are $l_d + 1$ such sequences:

$$\begin{array}{rl}
s_{l_d} & = pwd\ pwd\ pwd\ ...\ \ pwd\ \ pwd \\
s_{l_d-1} & = pwd\ pwd\ pwd\ ...\ \ pwd\ \ p_{l_d} \\
s_{l_d-2} & = pwd\ pwd\ pwd\ ...\ p_{l_d-1}\ \ p_{l_d} \\
 & \qquad\qquad\qquad ... \\
s_1 & = pwd\ \ p_2\ \ \ p_3\ \ ...\ p_{l_d-1}\ \ p_{l_d} \\
s_0 & = p_1\ \ \ p_2\ \ \ p_3\ \ ...\ p_{l_d-1}\ \ p_{l_d}
\end{array}$$

Let prh_i be the probability that Adv_M outputs 1 when sequence s_i was instantiated. These probabilities are fixed, although not known. Observe that $prh_0 = pr_1$ and $prh_{l_d} = 1 - pr_0$.

Let Pr_w be the probability Adv_{M1} wins the game. Then

$$Pr_w = Pr(b=0) \cdot Pr(Adv_{M1} \text{ outputs } 0|b=0) + Pr(b=1) \cdot Pr(Adv_{M1} \text{ outputs } 1|b=1)$$

Note that

$$Pr(Adv_{M1} \text{ outputs } 0|b=0) = \frac{1}{l_d}\sum_{i=1}^{l_d}(1 - prh_i)$$

and

$$Pr(Adv_{M1} \text{ outputs } 1|b=1) = \frac{1}{l_d}\sum_{i=1}^{l_d}(prh_{i-1}).$$

Since $Pr(b=0) = Pr(b=1) = 1/2$,

$$\begin{aligned} Pr_w &= \frac{1}{2}\frac{1}{l_d}\sum_{i=1}^{l_d}(prh_{i-1} + 1 - prh_i) = \frac{1}{2} + \frac{1}{2}\frac{1}{l_d}\sum_{i=1}^{l_d}(prh_{i-1} - prh_i) \\ &= \frac{1}{2} + \frac{1}{2}\frac{1}{l_d}(prh_0 - prh_1 + prh_1 - prh_2 + \ldots + prh_{ld-1} - prh_{l_d}) \\ &= \frac{1}{2} + \frac{1}{2}\frac{1}{l_d}(prh_0 - prh_{l_d}) = \frac{1}{2} + \frac{1}{2}\frac{1}{l_d}(pr_1 - (1 - pr_0)) = \frac{1}{2} + \frac{1}{2}\frac{1}{l_d}(pr_0 + pr_1 - 1) \end{aligned}$$

By assumption, Adv_M wins KE_M with probability $Pr_M = 1/2 + \delta$. Since $Pr_M = Pr(b=0) \cdot pr_0 + Pr(b=1) \cdot pr_1 = \frac{1}{2}(pr_0 + pr_1)$, conclude that $pr_0 + pr_1 = 1 + 2\delta$ and, substituting in the above, obtain

$$Pr_w = \frac{1}{2} + \frac{\delta}{l_d}$$

Noting that δ is non-negligible and positive completes the proof. □

C Proofs of Security

In this section we present formal proofs of security of protocols of Sect. 6.

C.1 Proof of security of the protocol of Constr. 1 (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 Sect. C.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 Sect. C.1.

Theorem 1 follows from Prop. 1 and 2.

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 and a constant u. Then, for $i = 1, ..., u$, $Dist_1$ queries the* encryption oracle $O_E(p')$ *that produces random encryptions e_i, as follows. Here $p' \in D \cup \perp$. If $p' = \perp$, then $e_i = Enc_{pk_S}(N_C, pwd, k_i)$, otherwise $e_i = Enc_{pk_S}(N_C, p', k_i)$, where $k_i \in_R \{0,1\}^n$ are chosen randomly and unknown to $Dist_1$. 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 wins KE_1. 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 KE_1.*

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:*
   $Dist_1$ outputs 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, and r_j is the message previously sent by S_j^C):*
   $Dist_1$ outputs the session id $sid_j = (r_j, e')$ and stops.
7. *Adv checks S_j^C*:
   Recall, S_j^C sent r_j, received $m_{S_j} = (e', m')$, and terminated.
   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, $Dist_1$ outputs OK and stops.

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 decryption fails or $N'_C \neq N_C$, $Dist_1$ 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 check 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.)

8. *Adv delivers a message $m_{S_j} = (e', m')$ to $S_j^{B^i}$ (allegedly) from client B^i (recall, B^i is corrupt):*
   $Dist_1$ outputs the session id $sid_j = (r_j, e')$.
9. *Adv checks $S_j^{B_i}$:*
   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).
   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, $Dist_1$ outputs OK.
10. *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 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.
11. *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 10, 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 8 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.
12. *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 10 and 11, 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 10–11, and to O_C in 12 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, $p = prob(E_1) + (1 - prob(E_1))prob(E_2|\neg E_1) < \frac{q}{|D|} + \epsilon_1 + (1 - \frac{q}{|D|})(1/2 + \epsilon_2) = 1/2 + \frac{q}{|D|} + \epsilon$.
□

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 7 (Adv checks S_j^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⊥, otherwise $Dist_2$ outputs OK. 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 12 (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 7 (Adv checks S_j^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 12 (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.

□

C.2 Proof of Theorem 2

The proof is a simple natural modification of the above proof of Thm. 1.

D KE Definition in the Setting with Confirmation Flows

It is easy to observe that Def. 2 does not allow for the "connection established" or failure notification flows from the server to the client. This is because Adv would otherwise always win KE_1 by trying a password, and learning whether he succeeded without checking the server.

Requiring such a flow increases the minimal round complexity of the protocol. Indeed, secure two-flow protocols – S starts, C responds – are possible without this requirement. (These two flows must be ordered, due to replay issues in DoA, as discussed in Sect. 4.1.) The confirmation flow from S must come after the client's message, and thus a secure protocol would have a minimum of three ordered flows.

The inability of S to confirm its status is not necessarily an insecurity, and is often acceptable. The one-round savings that comes with it justifies the interest in the definition and protocols. In Sect. D.1 we discuss attacks that exploit the absence of the confirmation flow, and their relevance and importance. We amend our definition to allow for the confirmation flow and to ensure protection against these attacks.

D.1 Failure accounting on the client side

Consider the following attack by real-life Adv. He pretends to be S to the client C. While Adv cannot establish a session with C, he can simulate behaviour of S when the password was mistyped by C. C will reasonably believe that he indeed mistyped the password. Then Adv will try a password with S. Then he will let C login to S. During this authenticated session, S will verify with C that C indeed mistyped the password, and will reset his P$\perp$ counter. Ad now received a "free" password try. Def. 2 is vulnerable to this attack since S is not allowed to let C know its status (see Sect. D for discussion). Further, to our knowledge, protection against this attack has not been formally considered in the literature.

Biometric-based authentication is even more vulnerable to this attack. This is because biometric reading is not in control of the the user, and the state of the art of the technology still admits high false rejection rate. Therefore, clients might readily believe their biometric was misread frequently.

The need for client failure outputs. A natural way to handle this situation is to allow C output special failure symbols $\perp^C$ and P$\perp^C$. Symbol P$\perp^C$ is output whenever the client indeed has mistyped the password (as confirmed by a real S), and $\perp^C$ may be output whenever a non-P$\perp^C$ error occurs.

The intent is to let S certify to C that he's seen the P$\perp$ in this session. This will prevent the attack described above, since C will only be convinced of his mistyping if he indeed mistyped in a session with S. Additionally, C could detect behaviour of S inconsistent with the protocol, and $\perp^C$ might be used for reporting this. The use of $\perp^C$ is allowed, but not required by a secure protocol.

Note, Adv can prevent P$\perp^C$ by not delivering messages, but it is not in his interest to do so.

Definitional approach to P$\perp^C$. Note that the fact whether the KE session was successful can be later determined by a confirmation flow. Could one then use Def. 2 and consider the failure confirmation flow and P$\perp^C$ externally to KE? That is, could one define a "confirmation flow" protocol, that would take as input the output of the KE protocol, and guarantee the correctness and integrity of the confirmation flow? If so, one might use the sid output by KE as something to bind the confirmation flow and P$\perp^C$ to. However, in case of P $\perp$ the confirmation flow cannot be transmitted securely to the instance of C, since there is no common secret to encrypt with. The ephemeral information is lost after KE is completed, and the use of non-ephemeral information tying the instances of S and C – the server's secret key – in a relatively independent protocol is not a good idea. Thus approaching the confirmation flow externally to Def. 2 is not very elegant. Instead, we incorporate the output of P$\perp^C$ in the definition.

Correctness requirement. Justified by the above discussion, we require that in the absence of an active adversary, if a client mistypes his password, with overwhelming probability he will output P$\perp^C$.

D.2 Definition of KE with the confirmation flow

Recall that Def. 2 disallows a confirmation flow, since it would let *Adv* learn "for free" whether S accepted the password. If we are to allow this flow, we must "charge" *Adv* for the information he learns. A natural way to do so is to charge for every P$\perp$, even if *Adv* doesn't check that instance. However, we still want to give *Adv* the ability to observe mistyped executions freely. Thus, we allow *Adv* to specify the password given to instance of C, and we credit *Adv* for each P$\perp^C$. Thus, the charged P$\perp$ will be reversed if and only if it was caused by a client instance, in which case *Adv* did not learn whether the password was accepted. Of course, *Adv* might want to see the output of instances of S and C. We employ the same "checking" mechanism, and charge *Adv* for each P$\perp$ and P$\perp^C$ resulting from checking. The following definition is based on Def. 2, and incorporates the above changes. As in Sect. 4, the definition for the HK setting with mistyping is extracted from Def. 3 simply by removing all uses of ℓ and the games where *Adv* does not know ℓ. Due to space constraints, the formal definition is postponed to Appendix D.3.

Definition 3. *We say that a KE protocol Π is* secure in the Combined Keys Model with Mistyping and P$\perp^C$*, if it satisfies requirements of Def. 2, but with respect to the games modified as described above.*

It is easy to see that this definition affords *Adv* similar powers, as Def. 2. The only difference is that now *Adv* is charged immediately for each P$\perp$ of S, which is reversed if a client instance outputs P$\perp^C$. This simply requires *Adv* to mistype by substituting C's password, if he wishes to observe a mistyped execution for free. Therefore, *Adv* of Def. 3 can perform the same attacks as *Adv* of Def. 2.

Notes. In Def. 3, we require that the protocol outputs P$\perp^C$ when appropriate. We note an additional, technical, justification of this requirement. By not outputting P$\perp^C$, the protocol causes charges to *Adv*, preventing him from observing mistypings "for free". This is inconsistent with the design of the definition.

Note, C_i must not output P$\perp^C$ before a partner S_j outputs P$\perp$. If that were the case, *Adv* could simply not deliver further messages to S_j, thus gaining a "free" P$\perp^C$, and a password attempt with it.

Consider a protocol Π that does not send the P$\perp^C$ confirmation if *Adv* was active (e.g. set a certain bit) during the execution. Since *Adv* cannot get a credit for P$\perp^C$, Π can leak information of his choice and still be secure by Def. 3. We do not view as a problem that *Adv* learns information in form other than "is p the password of C?". What matters is the probability of success of *Adv* in guessing the session key, reflected in the games. We believe our definition serves as a good tie-breaker in subjective cases.

D.3 Formal Definition of Security in the Setting with the Confirmation Flow

In this section, we give a formal definition of security informally described in Appendix D.2

Let n, m, D be as in Def. 2.

Game 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 1-7 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. Adv may choose to specify the password for 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 (a sid and either session key, the* failure symbol $\perp$*, or the* password failure symbol $P\perp$ *for the server and* $P\perp^C$ *for the client) according to the protocol. Adv is given only sid.*
   Adv gets charged one attempt for each $P\perp$ and credited same for each $P\perp^C$ output by honest parties.
6. *Adv "checks" any completed honest instance – then he is notified whether the instance output $P\perp$, $P\perp^C$, $\perp$, or a session key. Adv gets charged one attempt, if he checked S^C or C^S and it output $P\perp$ or $P\perp^C$ correspondingly.*
   When Adv accumulates q charges, he becomes restricted (and is notified of that) – he can never again deliver messages to any instances S^C_j or check any instances.
7. *Adv "opens" any successfully completed and checked honest instance – then he is given the session key output of that instance.*

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

Then Adv continues to run the game as before (execute steps 2-7). 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 subtleties relevant specifically to this game. Here it may so happen that *Adv* does not know how many charges he had accumulated.

However, as long as Adv delivers the confirmation flow to C, charges do not accumulate, even though C may have mistyped. Further, consider the case when Adv accumulated $q-1$ charges. P⊥ caused by C's mistyping would make Adv restricted, even though Adv might have in hand the confirmation flow which will cause $\mathrm{P}\bot^C$. This situation can be avoided for every Adv by simply choosing a greater q and modifying his strategy accordingly. Technical problems only arise when $|D| = 2$: $q = 1$ disallows free mistyping, and $q = |D|$ does not enforce security. However, this setting is of limited interest; we choose to give a simpler, rather than marginally more general definition.

Games KE_2, KE_3, DOA and SID are derived from KE_1 in the same manner as the corresponding games in Def. 2. The only difference between these games and the ones of Def. 2 is syntactic – here C may output $\mathrm{P}\bot^C$. This is because the other differences between the two KE_1 games is only in the password accounting and the client's password substitution abilities. Both of these differences are removed by the modifications.

D.4 Efficient Protocol in the Setting with the Confirmation Flow

Construction 3 *(KE with mistyping and $\mathrm{P}\bot^C$)*

S^C		C^S
choose $r \in_R \{0,1\}^n$		*choose* $k \in_R \{0,1\}^n$,
	$r \rightarrow$	
		set $\alpha = Enc_{pk_S}(N_C, pwd, k)$
	$\leftarrow \alpha, MAC_\ell(r,\alpha)$	
set $sid = (r,\alpha), out = OK$;		*set* $sid = (r,\alpha)$
decrypt α		
if fail, set $N_C = \bot, k \in_R \{0,1\}^n$;		
verify $MAC_\ell(r,\alpha)$ *and* N_C		
if fail, set $out = \bot$		
else verify pwd		
if fail, set $out = \mathrm{P}\bot$;		
set $K = F_{F_k(r)}(0), K' = F_{F_k(r)}(1)$		*set* $K = F_{F_k(r)}(0), K' = F_{F_k(r)}(1)$
	$F_{K'}(out) \rightarrow$	
		decode $out \in \{\mathrm{P}\bot, \bot, OK\}$
		if fail, output $(sid, \bot)$
		if $out = \mathrm{P}\bot$, *output* $(sid, \mathrm{P}\bot^C)$
If $out = OK$, *output* (sid, K)		*if* $out = \bot$, *output* $(sid, \bot)$
else output (sid, out)		*if* $out = OK$, *output* (sid, K)

Theorem 6. *The protocol Π of Constr. 3 is secure by Def. 3.*

Proof. Proof of Theorem 6 is similar to that of Theorem 1 presented in Appendix C and is omitted for conciseness.