Three-Party, Password-Authenticated Key Exchange with Server Private Keys: Security Models and Protocols[*]

Jean Lancrenon[†]
Interdisciplinary Centre for Security, Reliability and Trust,
Université du Luxembourg
jean.lancrenon@uni.lu

June 9, 2016

Abstract

In this paper, we conduct a study of three-party, password-authenticated key exchange in the three party setting (3-PAKE), focusing on protocols that require the trusted third party to have a high-entropy private key. We show how in this scenario we can maintain the user-friendly functionality of password-based authentication while provably achieving security properties that ordinary password-authenticated key exchange protocols cannot, namely key-compromise impersonation and a special form of resistance against internal state revealing. We define new simulation-based security models specifically tailored to 3-PAKEs with server private keys and illustrate our work with several protocols.

1 Introduction

1.1 Password-Authenticated Key Exchange

2-PAKEs and 3-PAKEs This article's main focus is on the design and analysis of Password-Authenticated Key Exchange (PAKE) protocols. The goal of a PAKE is for two parties to perform a cryptographic key exchange that is authenticated using each party's knowledge of a *password*. In the two-party case (2-PAKE), both parties share the same password, and in the three-party case (3-PAKE), each party shares its own password with a trusted server which aids in the exchange. We will be studying 3-PAKEs in which the server also holds strong secret keying information to which corresponds a public key. We call such protocols *3-PAKEs with server private keys*, or 3-PAKE[spk]s.

Motivation and Challenges The study of PAKEs has become quite important simply because of the ubiquity of password-based methods in everyday applications. In contrast with cryptographically strong, long-term keys – be they symmetric and private or asymmetric and managed in a public-key infrastructure – passwords are easy to handle and much more user-friendly. These features are of course due to passwords' most prominent property: they are typically *low-entropy*. However, this also makes brute force password guessing a feasible task. Thus, PAKEs must be carefully designed in order to resist so-called *dictionary attacks*: a well-constructed PAKE should allow at most one password to be tested per online impersonation attempt (*online* dictionary attack resistance), and its message exchange should not leak any information whatsoever on the underlying password (*offline* dictionary attack resistance).

[*] This is the full version of the research paper *What Public Keys Can Do for Three-Party, Password-Authenticated Key Exchange* [30], which was published in the proceedings of the 10th European Workshop on Public-Key Infrastructures, Services and Applications (EuroPKI 2013).

[†] The present project is supported by the National Research Fund, Luxembourg. Specifically, it is funded by the Postdoc AFR project PAKAJ (FNR reference number 3975577) running from July 2012 to June 2014.

Our Contributions, in a Nutshell Most of the research done in the last decade has been centered on providing provably secure 2-PAKEs; 3-PAKEs have been much less studied. Most notably, although such protocols have been considered before [33, 39], there appears to only be one that has been proven secure [16]. We propose to broaden the study by describing several security models capturing adversarial capabilities and protocols proven secure in these models. We also show that the models we consider are strictly separate.

More specifically, our investigations show that adding a private key at the server first makes protocol design much simpler, as it becomes possible to rely on simple, well-understood cryptographic building blocks. Secondly, we are able to exhibit a protocol that enjoys security properties that are desirable in key exchange in general, but impossible to obtain in PAKEs that only use passwords. These properties are resistance against key compromise impersonation and resistance against (a special form of) internal state revealing. Third, all of this can be done without sacrificing usability.

1.2 Related Work

2-PAKEs A large amount of research has been done on 2-PAKEs. Bellovin and Merrit [6] were the first to consider the problem of dictionary attacks in 2-PAKEs, and the first to propose a solution, without defining a formal security model. Jablon [25] devised a protocol as well. Lucks [31] later gave formal definitions of security capturing offline dictionary attack resistance. Halevi and Krawczyk [22] and Boyarsky [7] provided security definitions and protocols also in the two-party setting where one of the parties is a server with strong keying information, and the other is simply a user with a password. Since, many other protocols have been proposed, e.g. [1, 4, 8, 10, 12, 27, 17, 18, 21, 23, 28, 29, 26].

Most of these solutions are provably secure, starting with [4] who adapt to 2-PAKEs the now widespread indistinguishability-based security model of [5] for ordinary key exchange, and [8], who build their model on the simulation-based ones in [35] and [3]. In [12, 27, 17, 18, 21, 28, 29, 26] one can find various practical 2-PAKEs secure without random oracles using Cramer and Shoup's smooth projective hashing tool [15], and assuming a Common Reference String (CRS) is hardwired into the protocol specification.

There are also theoretical results showing that efficient, but not practical, 2-PAKEs can be constructed and proven secure with neither random oracles nor CRSs, see [19, 20]. Designing a practical 2-PAKE secure in the standard model with no CRS remains an open problem.

3-PAKEs The first to consider 3-PAKEs seem to be Steiner et al. [36], building directly on [6]. Their solution requires the trusted server to only know the passwords. Other protocols operating like this can be found in [2, 34, 9, 37, 38]. Notably, in [2] the first rigorous security model for 3-PAKEs is defined, following the indistinguishability approach of [5, 4]. They also strengthen the most commonly used security model for 2-PAKEs and provide a generic construction to get 3-PAKEs from 2-PAKEs. Besides [2] and [38], none of the cited works contain formal security considerations.

Other proposed solutions require the trusted server to hold a secret key of its own, e.g. [33, 39, 16, 40]. The protocols in [33] and [39] lack security proofs. The results in [16] and [40] are much more closely related to this work, and as such require special mention.[1] We highlight how our work differs from theirs.

The work in [16] In [16], the protocol design and analysis is approached in a modular way, using password-based authenticators (devised in [24]) to derive a protocol secure in an unauthenticated model of communication from a protocol that is secure in an authenticated model of communication. Security is proven without idealized assumptions. The authors work in a security model based on Canetti and Krawczyk's framework in [13], an indistinguishability-based framework which does not allow detection of key compromise impersonation (or KCI, see section 2 further below). They do however consider internal state revealing that does not trivially open the protocol to offline dictionary attacks. The models we consider here are simulation-based models directly derived from those of Shoup [35] and Boyko, MacKenzie, and Patel [8]. First, we propose a hierarchy of models, and depending on which of our models one uses, one can choose to capture or ignore different scenarios: adaptive (long-term key) corruptions, adaptive internal state revealing, KCI.

[1] Both of these works were discovered by the author of this paper only after the present work was submitted and published at EuroPKI 2013.

We also prove that these are separate models. Secondly, simulation-based models naturally incorporate a notion of key indistinguishability that is very close to that of the real-or-random framework from [2]. This is a significant difference, as the results in [2] show that working in a find-then-guess-style model (in which only the key output by the *Test* query may be either real or random) like in [16] may lead to a considerable security degradation specifically when studying password-authenticated protocols. Third, our models may accommodate the analysis of protocols that are not necessarily designed in a modular way. We believe this gives us more flexibility. Ultimately however, the protocols we exhibit here can be viewed to some extent as modularly designed. In fact, protocol **Prot2** is very similar to the protocol in [16]. It differs essentially only in the method employed to authenticate the server to the users: in [16], the authors use an encryption-based authenticator, while we use signatures. Another difference is our dual use of the hash key from a family of universal hash function as a server nonce. Finally, we find it strange that the security definition in [16] ultimately yields an adversary advantage that is negligible, rather than negligible "plus an online guessing attack success rate".

The work in [40] The work in [40] claims to have a notion of resistance against both KCI and internal state revealing, as we do for the strongest of our models. However, upon closer examination, we have found some flaws in the work. First, the protocol proposed does actually yield an offline dictionary attack in case important internal state is revealed. In the notations of [40] (from, say, figure 1 of the paper in question) if client A hopes to complete its protocol run with client B, it needs to hold on to both the ephemeral random value $x \in \mathbb{Z}_q$ and the group element $X^* := g^x H_1(pw_A, A, B)$, as both of these are needed for the final key computation. Revealing internal state will thus reveal both x and X^*, and an adversary can exhaustively search for pw_A by testing if $X^* = g^x H_1(pw, A, B)$ for successive values of pw. Destroying the encryption randomness fails to prevent this. The same can be said of client B in the protocol. Secondly, even without taking into account internal state, theorem 1 and 2 in [40] are unlikely to be valid (at least against active adversaries), as the encryption scheme is only required to be IND-CPA secure. Any proof of security of a PAKE that relies on encryption to directly hide the password is most likely going to rely at least on IND-CCA-2 encryption (see section B.1), as technically the simulator (in any model one uses) needs to be able to decrypt adversary messages in order to detect password guesses. Accordingly, IND-CCA-2 encryption plays an important role in both our work (and in [16]). Third, the *TestPassword* query is unnecessary. Finally, the protocol relies on idealized assumptions (specifically, the random oracle model).

1.3 Organization of the Paper

In section 2 we explain why we think 3-PAKE[spk]s are worth considering. Next, section 3 states the security properties we can expect of a 3-PAKE[spk], fixes some of the notation to be used throughout the paper, and gives an overview of the ideal-world simulation paradigm for provable security. Sections 4, 5, and 6 then respectively exhibit the static, password-adaptive, and password-and-state-adaptive network adversary models, each section being punctuated with a protocol and some comments. Section 7 is used to ease the reader into sections 8 and 9, which provide complete proofs of security against network adversaries for two of the three protocols we propose. (These proofs are admittedly very repetitive, but they have the advantage of being completely self-contained. Only once does one need to refer to a portion of the proof of one theorem to understand that of another, because in that case the proofs are absolutely identical: lemma 2 is proven exactly the same way as lemma 6.) Section 10 completes all three of our models with a single model capturing security against an Honest-but-Curious server. Finally, section 11 concludes the paper with some prospective future work. Computational assumptions and security definitions for some cryptographic primitives can be found in the appendix.

2 Why 3-PAKEs with Server Private Keys?

2.1 Retaining User-Friendliness

In a 3-PAKE with server private keys, in addition to knowing every user's password, the trusted server $\mathcal{T}$ has cryptographically strong, secret keying material $sk_{\mathcal{T}}$ to which corresponds public keying material $pk_{\mathcal{T}}$ that is available to each user.

The reader may be legitimately wondering at this point why a string like $pk_{\mathcal{T}}$ is suddenly making an appearance at the users' end: after all, is not the whole point of using passwords to get rid of such cumbersome data? The answer is that the users will not have to know it because since it is the same for every user, it can be hardwired into the protocol specification. The application thus retains its user-friendliness despite this added feature.

This concept is not new in PAKE research. All *practical* 2-PAKEs that are proven secure *without random oracles* use this hardwiring technique because they rely on a *Common Reference String* (CRS) known to all users. A CRS is basically a long public string that is generated in some secret manner and that all parties know. In PAKEs, they tend to appear as public keys to public-key encryption schemes; the "secret" part is then the corresponding secret decryption key which must be immediately destroyed: any entity that gets access to it can essentially undermine the whole system undetected.

The reason a CRS-based construction is preferable to one where all users jointly compute the public string is purely practical: deploying the same CRS for everybody allows new users to join the system without having to update the common data each time.

2.2 Proving Security Without Idealized Assumptions

The previous observation is useful to make our case from a theoretical point of view. Aside from the work of Abdalla et al. [2] and Cliff et al. [16], all 3-PAKEs in the literature either lack a security proof, or rely on idealized assumptions. In [2] the authors devise a way to generically construct a 3-PAKE from a 2-PAKE and prove its security in the following sense: if the underlying 2-PAKE is secure in the standard model of computation, then so is the resulting 3-PAKE. Thus [2] shows a generic way to obtain a standard-model-secure 3-PAKE from a standard-model-secure 2-PAKE.

We claim that given the current state-of-the-art in 2-PAKE research, our method for obtaining 3-PAKEs secure in the standard model is a good, and possibly more efficient, alternative. We reason as follows.

We already mentioned that all practical 2-PAKEs secure without idealized assumptions use a CRS. In particular, some entity has to generate this CRS and be trusted to destroy – or at the very least, to never disclose – the corresponding secret. If we construct a 3-PAKE generically from a CRS-based 2-PAKE, the entity most naturally placed to perform this operation is the server itself, since it is already trusted with all of the passwords. But then this server is also a very natural candidate to trust with correctly *using* cryptographically strong secrets, rather than ignoring or discarding them. It is, after all, a pity to not use the decryption key of a public-key encryption scheme. Thus, it makes sense to return to considering 3-PAKE[spk]s.

The upshot is that if our *starting point* is having strong secret keys at the server, the protocol design complexity drops significantly. We emphasize however that we are **not** attempting to make the case that in general protocol design, a CRS-based construction can systematically be replaced by a server with secret keys. The reasoning as presented here is very specific to the cryptographic primitive at hand.

2.3 Simpler Practical Protocol Designs

We explained above that often the CRS in 2-PAKEs is a public encryption key. The reason is quite simple: no information on a given password can be efficiently plucked from a *semantically secure encryption* of it, thus guaranteeing offline dictionary attack resistance. The challenge then becomes finding a way for a legitimate partner to exploit the resulting ciphertext's contents *without the decryption key.* Subtle mechanisms, such as *smooth projective hashing* [15], have been used to do this, but they yield complex designs.

In contrast, our scenario authorizes the server to use the secret keys, so we can rely in a much more straightforward way on well-understood, classic cryptographic primitives. As an example, in the protocols we propose users encrypt their passwords which can then be verified through decryption by the server, and the server digitally signs its messages to the users, who can verify the signatures. The only requirements we make on these primitives involve their level of security. No other properties are necessary.

We note that the (provably secure) protocols we present require at most six messages to be sent. Factoring in the additional security properties we can achieve, this is certainly not excessive: four of the messages are used to authenticate users to the server and vice-versa. The precise role of the other messages depend on the protocol, but can be thought of as the key confirmation process. We can compare this to an instantiation of Abdalla et al.'s scheme [2] with a two-message 2-PAKE, with no key confirmation between the parties and the server and none either between the two parties at the end: this yields seven messages sent.

2.4 Capturing More Security Properties

Finally, a good reason to consider 3-PAKE[spk]s is that with them we can provably defend against more attacks than with ordinary 3-PAKEs. Specifically, we can prevent *key compromise impersonation* and we can give a meaningful security definition of resistance against a limited form of *internal state* revealing.

Key Compromise Impersonation In any key exchange scheme, *Key Compromise Impersonation* (KCI) is said to occur if an attacker that gets its hands on a user's long-term keying information can impersonate some other user to that user. It is easy to see that 2-PAKEs cannot satisfy this property: if an attacker compromises a user's password, that attacker can always impersonate the *other holder* of that password to that user. At least, it cannot impersonate other users to that user, because that would require compromising other passwords, since these are distributed pairwise.

In the 3-PAKE case, this last point is no longer true. A user shares its password only with the trusted server. If its own password is the only mechanism that proves to a user that it is indeed speaking to the server, then compromise of this password by an adversary will allow this adversary to impersonate to that user *the server and any other user in the network at any time of its choice*, which really is the worst KCI scenario possible.

We are able to thwart this however if the server authenticates itself to users via strong secret keying information. This is an interesting property because it is quite reasonable to assume that strong secret keys at the server will be better protected than any password ever is (or ever will be) by a user.

Concretely, the protocols we propose either heavily mitigate the risk of KCI (see 5.3), or eliminate it entirely (see 6.3). Interestingly, protocol **Prot1** (see 5.3) shows that simply adding high-entropy private keys at the server is not sufficient to fully defend against KCI. One still needs to be careful with the protocol design.

Unerased Internal State Revealing Authenticated key exchange research also nowadays frequently considers security notions involving the revealing of certain forms of internal state. What this means precisely varies from author to author, but the general idea is to let the adversary obtain the values of certain intermediate computations during a protocol execution. Several flavors of this can be found in the literature, e.g. the *ephemeral randomness* revealing of [32] or the *unerased internal state* revealing of [35].

This kind of query is rarely seen in PAKE research for a simple reason: authorizing any form of internal state revealing opens PAKEs to immediate dictionary attacks. Thus, while some works do consider revealing state (e.g. [1, 10]), it includes password revealing by default.

We show that a meaningful definition of resistance against *Unerased Internal State* (UIS) revealing can be considered for 3-PAKE[spk]s by exhibiting protocols that are not trivially subjected to dictionary attacks when targeted by this query. In fact, all of the protocols we consider do this, even though only one of them actually fulfills the corresponding security requirement (see 6.3), while the two others clearly do not (see 6.5).

3 3-PAKE[spk]s and Ideal-World Simulation

This section and the four that follow contain the main technical contributions of the paper. We essentially adapt the simulation-style security models of Shoup [35] (the version of April 1999) and Boyko et al. [8] to the case of 3-PAKE[spk]s. Throughout the rest of the paper, we adopt their notations.

3.1 General System Description

The Users The system consists first of a set of *users* which we index by strictly positive integers i and to whom are attributed identities ID_i of some agreed upon length. We assume that each user i has a password pw_i that was selected uniformly at random from some *dictionary* D, and that this password is kept secret. D is a (possibly small) publicly known set of strings of some agreed upon length.

The Server There is also a *server* $\mathcal{T}$ indexed by 0. A *server key generation algorithm* $\mathcal{K}_\mathcal{T}$ is run once on input 1^η - where $\eta \in \mathbb{N}$ is a security parameter - at the time the system is initialized to produce a pair of strings $(pk_\mathcal{T}, sk_\mathcal{T})$. $\mathcal{T}$ keeps $sk_\mathcal{T}$ secret and $pk_\mathcal{T}$, which is publicly available data, is copied into the users' protocol specification. The list $\{ID_i, pw_i\}_i$ is handed over to $\mathcal{T}$, who also keeps the passwords secret.

Statement of Goals and Desirable Properties

Protocol Goal At the end of a protocol run performed by registered users i and i' and server $\mathcal{T}$, either i and i' used as input their correct respective passwords pw_i and $pw_{i'}$ and the server public key $pk_\mathcal{T}$ to authenticate themselves to $\mathcal{T}$, and $\mathcal{T}$ used as input its secret key $sk_\mathcal{T}$ to authenticate itself to both parties, in which case i and i' should have computed a secret, random-looking, shared session key SK, or the protocol is aborted by one of the parties.

It is important to note that we do not consider valid those protocols which require the server to authenticate itself to the users using the shared password. Given the keying information the server has, it should not need this.

Desired Security Properties We have already mentioned online and offline dictionary attack resistance as necessary properties for all PAKEs. In [2], the authors identify a very specific kind of dictionary attack that is proper to 3-PAKEs: *insider attacks* in which malicious users registered into the system with their own passwords try to determine honest users' passwords through manipulated protocol runs. This is a very important property that we take into account as well.

Another security feature isolated in [2] that we capture here is that of *session key privacy with respect to the server*. As in [2], we must assume that $\mathcal{T}$ is at worst *Honest-but-Curious* (HbC), i.e. that the only adversarial behavior $\mathcal{T}$ might engage in involves eavesdropping on conversations. It is otherwise trusted to carry out the protocol faithfully. However, in an effort to minimize the trust placed in the server with respect to session key use, we require that $\mathcal{T}$ not be able to compute the session key established during a protocol run. Abdalla et al. [2] propose with this notion a clear formal separation between three-party key *exchange* and three-party key *distribution*.

We end this paragraph with a security property that remains to be integrated into our model, but that we believe the protocols we present below satisfy, that is *server forward secrecy*: disclosure of the server's secret keys should not reveal past session keys. We stress this property does not automatically hold even given the security against the HbC server. We leave this as future work.

3.2 Ideal-World Simulation Methodology

To define security, we use the so-called *ideal-world simulation paradigm* which finds its origins in multi-party computation. Two computational environments are described. Within the *ideal world* the protocol's goals are ideally achieved between all users in the presence of an adversary. Whatever mischief he may cause in this world represents inevitable attacks on the very service we are trying to provide. In the *real world*, the target protocol is executed between users in the presence of an adversary who may disturb it however he wishes according to the powers he is afforded. The goal resides in comparing these two execution environments: we say that security is achieved if for any real-world adversary running against the protocol we can construct an ideal-world adversary that behaves the same way. Given the definition of the ideal world, this implies that any winning strategy the real-world adversary finds is necessarily one of the inevitable ones. Thus, we prove that the protocol can really do no better.

Adversaries and Ring Masters In both worlds, the *adversary* ($\mathcal{M}^*$ in the ideal world, $\mathcal{M}$ in the real world) plays against an entity we call the *ring master* ($\mathcal{RM}^*$ in the ideal world, $\mathcal{RM}$ in the real world) whose task is to run the protocol generating all of the necessary random variables, and to answer and react to various *operations* the adversary asks to have performed. **All adversaries are assumed to be probabilistic, polynomial-time algorithms (PPTA for short), where the time measure is a function of the security parameter $\eta \in \mathbb{N}$.**

We will be considering two types of adversaries: *network adversaries*, which completely control all communications between honest users and the server, and *server adversaries*, which simply try to infer session key information from honest protocol runs. As in [2], we model these adversaries separately: trying to merge them into one entity does not make sense, for we would effectively be giving the malicious network adversary all of the secret keys.

Transcripts In either world, as the execution of the interaction between the ring master and the adversary progresses, a *transcript* – $\mathcal{IW}(\mathcal{M}^*)$ in the ideal world and $\mathcal{RW}(\mathcal{M})$ in the real world – logging the adversary's actions is constructed. These random variables are the data used to compare both computational environments.

Definition of Security We are now ready to define security:

We say that a 3-PAKE[spk] is secure against network adversaries (respectively, against HbC servers) if for every real-world, network adversary $\mathcal{M}$ (resp., HbC server $\mathcal{T}$) there exists an ideal-world network adversary $\mathcal{M}^*$(resp., HbC server $\mathcal{T}^*$) such that $\mathcal{IW}(\mathcal{M}^*)$ and $\mathcal{RW}(\mathcal{M})$ (resp., $\mathcal{IW}(\mathcal{T}^*)$ and $\mathcal{RW}(\mathcal{T})$) are computationally indistinguishable.

Jumping ahead, the adversary will in both worlds be allowed to have past-established session keys placed in the transcript (see the "reveal session key" operation). In the real world, these are the real keys. In the ideal world, they will be random, pairwise-independent bitstrings. The computational indistinguishability requirement above then naturally captures (among other things) the idea that correctly exchanged session keys cannot be efficiently told apart from random strings. This is why when following the ideal-world simulation paradigm - as is done in [35] or [8] - the test query found in most models in the style of [5] is unnecessary.

4 Static Network Adversaries

We first describe a basic security model that captures *statically corrupted users*, i.e. parties that are controlled by the network adversary upon initialization. The adversary registers users of its own with passwords of its choice, but cannot corrupt honest users that are already in play. Of course, the adversary may still try to guess passwords by initiating, and interfering in, key exchange communications.

The bulk of our models' description is in this section; those in sections 5 and 6 below are built on this one by adding adversarial capabilities. Concretely, describing our models involves stating what operations the adversary can ask of the ring master, explaining what their effects are, and by which string(s) they are accounted for in the transcript. In the model description, the operation names are in bold, followed directly by the arguments they take. Paragraphs in italics contain additional comments or explanations.

We expected our first protocol, dubbed **Prot0** (see 4.4), to be provably secure in this model, but upon closer inspection it appears not to be so. This phenomenon is interesting as it seems to be directly related to the use of passwords and the Diffie-Hellman key exchange. We comment on this at the end of the section.

4.1 The Static Ideal World

Here we describe the operations the static network adversary $\mathcal{M}^*$ may ask of the ideal-world ring master $\mathcal{RM}^*$.

Let η be the security parameter. In the ideal world, the only parameters η determines are the length ℓ_{SK} of the session keys, and the length of the password ℓ_{pw}. We require that ℓ_{SK} grows fast enough as a function of η, in the sense that $2^{\ell_{SK}}$ is a negligible as a function of η. No such requirement need be made for ℓ_{pw} however.

Initialize server

$\mathcal{M}^*$ starts the game on input 1^η by formally invoking the server that is ultimately controlled by $\mathcal{RM}^*$. It carries the index 0. It also specifies a non-empty dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$, an algorithm to efficiently select passwords uniformly at random from D, and an algorithm to efficiently tell if an element of $\{0,1\}^{\ell_{pw}}$ is in D or not.

String logged in the transcript: $(\text{"}\textit{initialize server}\text{"}, 0, D)$

No identity or even integer index really needs to be assigned to this party in this formal setting, where there can be no confusion as to which higher authority honest parties are to communicate with. In practice, adding the identity of $\mathcal{T}$ to the messages is likely to be necessary.

Having the adversary choose the dictionary is convenient for two reasons. First, it seemingly gives our adversary more power, although whether $\mathcal{M}^*$ (or $\mathcal{M}$ in the real world) chooses D or gets it as input really does not make a big difference. Even the restriction that the passwords be sampled uniformly at random is not necessary, but we keep it for simplicity. Second, by doing this we avoid having to deal with defining an additional distribution ensemble of dictionaries indexed by the security parameter η. Instead, it is implicitly defined by $\mathcal{M}^*$ (or $\mathcal{M}$).

Initialize user, i, ID_i

$\mathcal{M}^*$ chooses an *identity* bitstring ID_i with which to initialize i. ID_i should not have been used before for another user, or for a "set password" operation (see below). Also, a *password* pw_i is chosen uniformly at random from D and assigned to i outside of the adversary's view.

Transcript: $(\text{"}\textit{initialize user}\text{"}, i, ID_i)$

This query allows $\mathcal{M}^*$ to bring into existence new honest players, with their own passwords.

Set password, ID, pw

$\mathcal{M}^*$ specifies an identity string ID that has not previously been assigned to a user and associates to this string a password $pw \in D$ of its choice. $\mathcal{RM}^*$ knows the pair (ID, pw).

Transcript: $(\text{"}\textit{set password}\text{"}, ID, pw)$

This query allows $\mathcal{M}^*$ to put into play statically corrupted users with their own passwords registered at the server. Server 0, controlled by $\mathcal{RM}^*$, should be thought of as having access to these passwords. It is through this operation that insider attacks as considered by Abdalla et al. in [2] are modeled.

Initialize user instance, (i,j), $role_{ij}$, PID_{ij}

$\mathcal{M}^*$ asks to have a previously initialized user i initialize an *instance* j of that user. $\mathcal{M}^*$ assigns to (i,j) a *role* $role_{ij}$, which is either *open* or *connect*, and a *partner identity* bitstring PID_{ij}. This bitstring may or may not have been previously used to initialize a user. If not, we require that a "set password" operation on PID_{ij} has been previously performed.

Transcript: $\big(\text{"}\textit{initialize user instance}\text{"}, (i,j), role_{ij}, PID_{ij}\big)$

This operation allows $\mathcal{M}^*$ to activate many different key exchanges for a given user. During such an exchange, a user may either be waiting for its partner to connect to it, or may be expected to connect to its partner. In either case, the user knows its local session counter, and does not know the local session counter of its partner. $\mathcal{M}^*$ knows both, since it controls the network communications. The activated instance knows which identity string it is targeting, although it is not aware of whether that string belongs to an honest user.

Initialize server instance, $(0, k)$**,** $PIDS_{0k}$

$\mathcal{M}^*$ asks to have an instance k of the server initialized. $\mathcal{M}^*$ specifies also the instance's *partner identities* $PIDS_{0k} = (OID_{0k}, CID_{0k})$ where OID_{0k} is the *opening partner identity* and CID_{0k} is the *connecting partner identity.* We require these identity strings to be distinct, that at least one of them has been assigned to a user, and that both have been assigned passwords. Note that $PIDS_{0k}$ is *ordered*: the server assigns to OID_{0k} the role *open* and to CID_{0k} the role *connect.*

Transcript: $\big(\text{"}\textit{initialize server instance}\text{"}, (0, k), PIDS_{0k}\big)$

This operation allows $\mathcal{M}^*$ to invoke instances of the server. Naturally, this server expects to be relaying messages between two specific users during an exchange, hence the pair of partner identities. The server instance need not be assigned a role because it will always be communicating with a pair of identities having the same form: one will be asked to open, and the other to connect. On the other hand, it is reasonable to expect the server to know each of its partners' assigned roles in the exchange at hand. This is data local to the server; if one of the two identities in play is actually a statically corrupted user, this role attribution at the server still occurs.

Terminate user instance, (i, j)

$\mathcal{M}^*$ specifies a previously initialized user instance to *terminate.* This instance will no longer participate in a key exchange.

Transcript: $\big(\text{"}\textit{terminate user instance}\text{"}, (i, j)\big)$

Terminate server instance, $(0, k)$

$\mathcal{M}^*$ specifies a previously initialized server instance to terminate. It can no longer participate in a key exchange.

Transcript: $\big(\text{"}\textit{terminate server instance}\text{"}, (0, k)\big)$

Test instance password, i**,** $(0, k)$**,** pw

$\mathcal{M}^*$ specifies a previously initialized user i, a previously initialized and not yet terminated server instance $(0, k)$ with ID_i as one of the components of $PIDS_{0k}$, and a password string $pw \in D$. It receives in return the knowledge of whether or not $pw = pw_i$. This is allowed under the condition that $(0, k)$ has not completed an exchange (see below). We also make the restriction that a "test instance password" operation can only be done **once on input** i**,** $(0, k)$**,** pw**.**

We will say that a password was *successfully guessed* if it has been the target of a successful "test instance password" operation. Once a password has been successfully guessed, $\mathcal{M}^*$ will no longer try to guess it. If a "test instance password" operation results in an incorrect guess, we will say that the operation *failed.*

This operation leaves no record in the transcript.

This operation allows $\mathcal{M}^*$ to make password guesses at the server. However, no such password guesses are allowed to target users because that would require the adversary to impersonate the server, which we forbid explicitly; this reflects the server having a private key that network adversaries do not have access to. It also formally excludes protocols that call for passwords to be used to authenticate the server to users.

*Notice that if $\mathcal{M}^*$ tries a guess on one end of the server instance, **it still has the possibility to attempt a guess at the other end of this same instance.** Thus, there are scenarios where $\mathcal{M}^*$ could run **two** "test instance password" operations on **one** initialized instance. This cannot happen in [8]. However, this is not a violation of the usual desired restriction on online guessing, since this ability to perform two tests involves two distinct passwords.*

The reason this operation does not log anything in the transcript will be more apparent in the proofs of security, but the idea is that if we were to include a password testing log in the ideal world, we would need one in the real world as well. Thus, we would need some kind of predefined real-world event indicating that a password is being tested. Yet, no such event can be reasonable defined in general; in fact our objective with this model is to precisely single out what these events are for every given real-world protocol.

Exchange completed, $(0, k)$

$\mathcal{M}^*$ specifies a previously initialized and not yet terminated server instance $(0,k)$ and indicates that it has completed its role in the current exchange. This requires that no **failed** "test instance password" operation was conducted on $(0,k)$ targeting a component of $PIDS_{0k}$ that is assigned to an initialized user. (Recall that there is always at least one such component.)

Transcript: $\big($"*exchange completed*", $(0,k)\big)$

This operation allows $\mathcal{M}^*$ to stipulate when a server instance has served its purpose in an exchange. Accordingly, it can only occur if both password checks have passed.

Start session, (i,j)

$\mathcal{M}^*$ specifies a previously initialized and not terminated user instance (i,j) and indicates that a session key should be constructed for it. One of several *connection assignments* is given to (i,j), depending on how it was initialized:

- **Open for connection from** (i',j') **through** $(0,k)$

This requires $PID_{ij} = ID_{i'}$ for some initialized user i', $role_{ij} = open$, (i',j') to have been initialized, $PID_{i'j'} = ID_i$, $role_{i'j'} = connect$, $(0,k)$ to have been initialized, and $PIDS_{0k} = (ID_i, ID_{i'})$. $\mathcal{RM}^*$ selects a session key SK^*_{ij} uniformly at random, and we now say that (i,j) *is open for connection from* (i',j') *through* $(0,k)$.

*There are not too many prerequisites for an instance to be open for connection from another instance. This is because we authorize implicit authentication. In this case, the user that is the last to send a message can very well have only received replayed material and generated a key with this material. However, the protocol should guarantee that this key will remain uncomputable by the adversary. (Note that in this paper we do not have an example of a provably secure implicitly authenticated protocol. We were hoping that **Prot0** might do, but were mistaken, see 4.4. Protocols **Prot1** and **Prot2** enjoy mutual authentication.)*

The reason a server instance is required to have been initialized mirrors the fact that before constructing a session key and beginning to use it, the opening instance must be sure that its purported partner has authenticated itself, and this can only be done through the server.

- **Connect to** (i',j') **through** $(0,k)$

This requires $PID_{ij} = ID_{i'}$ for some initialized user i', $role_{ij} = connect$, (i',j') to have been opened for connection from (i,j) through $(0,k)$, and $(0,k)$'s exchange to have been completed after (i',j') has been initialized. In this case, $\mathcal{RM}^*$ sets $SK^*_{ij} \leftarrow SK^*_{i'j'}$.

For an instance to connect to another instance, there are more requirements. This is due to the fact that both users must have had their passwords checked at the server before they can share a key, implying that the server's exchange was necessarily completed. Also, note that the connecting instance must have been initialized before the partner instance was opened for connection and the server instance has completed its exchange.

- **Exposed through** $(0,k)$

If $role_{ij} = open$ (resp., $connect$), this is allowed if $(0,k)$ has been initialized with $PIDS_{0k} = (ID_i, PID_{ij})$ (resp., $(0,k)$'s exchange has been completed with $PIDS_{0k} = (PID_{ij}, ID_i)$), and either PID_{ij} is not the identity of an initialized user, or PID_{ij} is the identity of an initialized user whose password was successfully guessed, or pw_i was successfully guessed. In this case, the adversary $\mathcal{M}^*$ specifies the session key SK^*_{ij}.

This connection assignment basically stipulates the conditions that need to be met for an honest instance to be in fact sharing a key with $\mathcal{M}^*$. Either its partner is a statically corrupted user that was legally registered by the server at $\mathcal{M}^*$'s request, or it is a user whose password was previously guessed, or its own password was previously guessed. In any of these cases, the connection still needs to have gone through an honest server instance with the appropriate partner identities. (The condition is a bit more stringent in the case of a connecting instance, since the server's exchange actually needs to be completed.) This is because $\mathcal{M}^*$ is not allowed to impersonate the server. Note however that these rules do not forbid opening or connecting instances even if a password has been guessed: two users can have engaged in a perfectly honest exchange even if the adversary knows a password.

The conservative vs. liberal exposure rule: In [35], two different exposure rules are exhibited. Adapted to our case, the **liberal** rule is the one described above while the more constraining **conservative** rule stipulates that exposing (i, j) **is not allowed if only** pw_i **was successfully guessed.** Just as in [35], simulatability using one or another model provides different security guarantees. This distinction is absent from [8] because it only makes sense when the communicating parties have **different long-term keys to begin with**, which is obviously not the case for 2-PAKEs. We shall return to this issue in sections 5 and 6.

We finally require that connection assignments be efficiently computable from the transcript up to the current "start session" operation. In other words, if the last current log in the transcript is a "start session" operation, the connection assignment attributed to the user in question is uniquely determined by the rest of the transcript. This requirement reflects the fact that some criterion is in place to determine how instances match their conversations. How this is done is not relevant to the ideal service, but it is crucial for concrete protocols. This constraint also forces connection assignments to be unique.

The reader may be wondering by what means connection assignments are uniquely determined from transcript prefixes, i.e. with which transcript logs. See the "implementation" operation below.

Transcript: $\big(\text{"}start\ \ session\text{"}, (i, j)\big)$

Reveal session key, (i, j)

$\mathcal{M}^*$ specifies a previously initialized user instance (i, j) that holds a session key. It receives in return the session key SK^*_{ij} that instance holds.

Transcript: $\big(\text{"}reveal\ \ session\ \ key\text{"}, (i, j), SK^*_{ij}\big)$

This operation is added to model session key leakage due to higher-level use of the established keys. The idea is to ensure that revealing past session keys does not affect the security of new session keys.

Implementation, *string*

$\mathcal{M}^*$ simply adds a string to its transcript. This operation is needed for several reasons. First, it is used to make sure that the transcripts are comparable in a meaningful sense, i.e. such that they cannot be distinguished because of a simple difference in syntax. Secondly, the information that goes into the transcript via these operations is used to determine which connection assignments are attributed to the instances that have started sessions. How this is done exactly will be more obvious when we describe the static real world.

Transcript: $(\text{"}implementation\text{"}, string)$

This completes our description of the ideal world adversary's actions.

4.2 Some Further Explanations

Dictionary Attacks We briefly explain how the model thwarts dictionary attacks. The "test instance password" operation can only be tried out once per password held by a given server instance, and in the ideal world this is the only mechanism allowed to verify guesses. Therefore, online trials are indeed limited to one per one of the two passwords at a given instance. Furthermore, since testing is no longer allowed after a server instance has completed an exchange, offline verification is avoided as well. Finally, a server instance can be the intermediary between an honest user and a statically corrupted one, thus taking insider attacks into account.

Server Instances Our model is designed in such a way that a correct exchange can only occur between two user instances and a single unique server instance. This seems like a fairly natural requirement. We have to explicitly model the server since that it where all of the password testing happens. Thus, as there are user instances, it is reasonable to have server instances. From that point forward, even if the server's role is limited to checking passwords, it sounds like good security practice to uniquely bind a server instance to a given session key exchange.

Key Compromise Impersonation Observe the difference between the liberal and conservative exposure rules: some thought reveals that the first rule allows KCI, while the second does not. Below we shall see that

Prot1 is secure under the liberal rule and insecure under the conservative rule (see 5.3), while and **Prot2** is secure under the conservative rule (see 6.3).

Remarks: There are some notable differences between our definitions and those found in [8]. First, rather than using an "application" operation to model key leakage due to higher-level applications, we simply reveal session keys. This is more direct, and likely an equivalent definition.

Another more interesting difference is that despite being in the implicit authentication model, we do not need so-called "dangling" connection assignments. The reason for these in [8] is that in 2-PAKEs mutual authentication is equivalent to key confirmation since there is no way for one party to verify the other's password message by message. (Such a mechanism would immediately open the protocol to dictionary attacks.) 3-PAKE[spk]s do not suffer from this; in fact, our protocols do actually perform message-by-message authentication.

We followed the work of [8] in that we explicitly integrated the passwords into the ideal world. As it is already explained in [8], not doing so would require explicitly incorporating into the model a formula measuring the non-negligible advantage the adversary has in guessing passwords through online impersonation attempts. Roughly, rather than having $\mathcal{IW}(\mathcal{M}^*)$ be computationally indistinguishable from $\mathcal{RW}(\mathcal{M})$, security would be defined by asking that $\mathcal{IW}(\mathcal{M}^*)$ and $\mathcal{RW}(\mathcal{M})$ be "only negligibly more distinguishable than $\frac{N_{online}}{\#D}$", where N_{online} is the number of online guessing attempts made by $\mathcal{M}$ in the real world. A security model (for 2-PAKEs) in this vein can be found in [18]. In a certain sense, such a definition is arguably more elegant because it is conceptually meaningful that in the ideal world of *any* key exchange protocol, the authentication mechanism is ultimately irrelevant: entities that authenticate correctly by whatever means end up sharing a key, and otherwise they do not.

4.3 The Static Real World

Now we describe the operations a static network adversary $\mathcal{M}$ can demand to have done by $\mathcal{RM}$ if it controls the network in the real world.

Again, let η be the security parameter. In the real world, an actual protocol is being run, so η is used to determine much more than just the session key and password lengths ℓ_{SK} and ℓ_{pw}.

Initialize server

On input 1^η, $\mathcal{M}$ starts the game by formally invoking the server $\mathcal{T}$, as before controlled by $\mathcal{RM}$. Like in the ideal model, we identify it with index 0. $(pk_\mathcal{T}, sk_\mathcal{T})$ is generated by $\mathcal{RM}$ running $\mathcal{K}_\mathcal{T}(1^\eta)$, and $pk_\mathcal{T}$ is given to $\mathcal{M}$, whose first output is the (non-empty) dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$ (and the algorithms that go with it). $\mathcal{RM}$ will run $\mathcal{T}$.

Transcript: $(\text{"}initialize\ server\text{"}, 0, D)$, followed by
$(\text{"}implementation\text{"}, \text{"}server\ public\ key\text{"}, pk_\mathcal{T})$

Initialize user, i, ID_i

$\mathcal{M}$ chooses an identity ID_i with which to initialize i. ID_i should not have been used before for another user, or for a "set password" operation. A password pw_i is chosen uniformly at random from D and assigned to i outside of $\mathcal{M}$'s view. The user gains access to $pk_\mathcal{T}$.

Transcript: $(\text{"}initialize\ user\text{"}, i, ID_i)$

Set password, ID, pw

$\mathcal{M}$ specifies an identity ID that has not already been attributed to a user, and a password $pw \in D$.

Transcript: $(\text{"}set\ password\text{"}, ID, pw)$

Initialize user instance, (i, j), $role_{ij}$, PID_{ij}

$\mathcal{M}$ asks to have a previously initialized user i initialize an instance (i, j) of that user. $\mathcal{M}$ assigns to (i, j) a role $role_{ij}$ of *open* or *connect* and a partner identity PID_{ij}. If PID_{ij} has not been attributed to a user, we require that a "set password" operation on PID_{ij} should have been previously performed. Obviously, the instance has access to $pk_\mathcal{T}$.

Transcript: $\big(\text{"}\mathit{initialize\ \ user\ \ instance}\text{"}, (i,j), role_{ij}, PID_{ij}\big)$

Initialize server instance, $(0,k)$**,** $PIDS_{0k}$

$\mathcal{M}$ asks to have an instance k of the server initialized, and specifies the partner identities $PIDS_{0k} = (OID_{0k}, CID_{0k})$, where OID_{0k} is the opening partner identity and CID_{0k} is the connecting partner identity. These strings should be distinct, at least one of them has been assigned to a user, and both must have been assigned passwords. The server assigns to OID_{0k} the role *open* and to CID_{0k} the role *connect*. Server instances, which are controlled by $\mathcal{RM}$, have access to $sk_{\mathcal{T}}$.

Transcript: $\big(\text{"}\mathit{initialize\ \ server\ \ instance}\text{"}, (0,k), PIDS_{0k}\big)$

Deliver user message, (i,j)**,** $InMsg$

For this operation to take place, user instance (i,j) must be initialized. $\mathcal{M}$ specifies an *incoming message* $InMsg$ which the instance processes according to the protocol specification, and using the keying information it was handed at its initialization. The instance eventually produces an *outgoing message* $OutMsg$ and reports its *status* $status_{ij}$ to $\mathcal{M}$. The status can be one of three values: *accept*, *continue*, or *reject*. If the instance accepts, it generates a session key SK_{ij} and halts. If the instance asks to continue, it has not generated a key yet and is ready to process another expected message. If the instance rejects, it terminates without generating a session key, and can no longer be used in the computation.

Transcript: $\big(\text{"}\mathit{implementation}\text{"}, (i,j), InMsg, OutMsg, status_{ij}\big)$

If the instance accepts, add to the transcript:
$\big(\text{"}\mathit{start\ \ session}\text{"}, (i,j)\big)$

If the instance rejects, add:
$\big(\text{"}\mathit{terminate\ \ user\ \ instance}\text{"}, (i,j)\big)$

This query is a means for $\mathcal{M}$ to actively manipulate messages to and from various instances. It then gets reports on how honest instances behave in reaction to what they receive. $\mathcal{M}$ can, for instance, interleave runs, inject messages of its own, or simply forward messages as they should be from one instance to another.

Deliver server message $(0,k)$**,** $InMsg$

For this operation to take place, server instance $(0,k)$ must be initialized. $\mathcal{M}$ specifies an *incoming message* $InMsg$ which the instance processes according to the protocol specification, and using the keying information it has access to. The instance eventually produces an *outgoing message* $OutMsg$ and reports its *status* $status_{0k}$ to $\mathcal{M}$. The status can be one of three values: *accept*, *continue*, or *reject*. If the instance accepts, it has delivered all of the messages it expects to and halts. If the instance asks to continue, it is ready to process another expected message. If the instance rejects, it terminates and can no longer be used to process messages in the computation.

Transcript: $\big(\text{"}\mathit{implementation}\text{"}, (0,k), InMsg, OutMsg, status_{0k}\big)$

If the instance accepts, add:
$\big(\text{"}\mathit{exchange\ \ completed}\text{"}, (0,k)\big)$

If the instance rejects, add:
$\big(\text{"}\mathit{terminate\ \ server\ \ instance}\text{"}, (0,k)\big)$

Reveal session key, (i,j)

$\mathcal{M}$ specifies a previously initialized user instance (i,j) that has accepted. It receives in return the session key SK_{ij} that instance holds.

Transcript: $\big(\text{"}\mathit{reveal\ \ session\ \ key}\text{"}, (i,j), SK_{ij}\big)$

Adversary coins

When $\mathcal{M}$ ceases interacting with $\mathcal{RM}$, the last entry in the transcript is

$(\text{"}\mathit{implementation}\text{"}, \text{"}\mathit{adversary\ \ coins}\text{"}, coins)$

where *coins* is the string of all random values chosen by $\mathcal{M}$ during the course of the interaction.

4.4 A Tentative Protocol: the (Odd) Case of Prot0

We now present the first protocol we designed to fit our models, **Prot0**. Contrarily to what we expected, it seems **not provably secure using the above described formalism**. Fortunately, this is not the case of the two other protocols in this work. We choose to expose **Prot0** anyway because it is interesting to see the obstruction the simulation runs into: the combination of the passwords' (possibly) low entropy and use of the Diffie-Hellman construct. Note that **Prot0** can only hope to achieve implicit authentication, as would any four-pass protocol following this message pattern.

Setup Let $\eta \in \mathbb{N}$ be the security parameter. Let G be a group of prime order q, with eta being q's bitlength, let g be a generator of G, let $Enc := (\mathcal{K}_E, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme, and let $Sig := (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$ be a public-key signature scheme. We run $\mathcal{K}_E(1^\eta)$ to get (pk_E, sk_E) and $\mathcal{K}_S(1^\eta)$ to get (pk_S, sk_S). The server's strong secret keying material $sk_\mathcal{T}$ is set to (sk_E, sk_S) and the corresponding public keying material $pk_\mathcal{T}$ - which we assume is hardwired into the protocol specification - is set to (pk_E, pk_S). Finally, let $\{H_n\}_n$ be a family of universal hash functions, mapping into $\{0,1\}^{\ell_{SK}}$, where ℓ_{SK} is long enough in η (i.e., $\frac{1}{2^{\ell_{SK}}}$ is a negligible function of η). We also assume that (q, G, g) and $\{H_n\}_n$ are a part of the protocol specification. Let D be the dictionary.

Running Prot0 between $\mathcal{A}$lice and $\mathcal{B}$ob $\mathcal{A}$ and $\mathcal{B}$ are assigned identity bitstrings $ID_\mathcal{A}$ and $ID_\mathcal{B}$ respectively, and hold passwords $pw_\mathcal{A}$ and $pw_\mathcal{B}$ respectively.

1). $\mathcal{A}$ chooses $x \in \mathbb{Z}_q^*$ uniformly at random, computes $X \leftarrow g^x$, computes $c_\mathcal{A} \leftarrow \mathcal{E}_{pk_E}(X, pw_\mathcal{A}, ID_\mathcal{A}, ID_\mathcal{B})$, and sends $c_\mathcal{A}$ to $\mathcal{T}$;

2). $\mathcal{T}$ decrypts $c_\mathcal{A}$ and checks $\mathcal{A}$'s password. If the test fails, the protocol is aborted. Otherwise, $\mathcal{T}$ chooses a hash index n uniformly at random, computes $\sigma_{\mathcal{T}1} \leftarrow \mathcal{S}_{sk_S}(X, n, ID_\mathcal{A}, ID_\mathcal{B})$, and sends $(X, n, ID_\mathcal{A}, ID_\mathcal{B}, \sigma_{\mathcal{T}1})$ to $\mathcal{B}$;

3). $\mathcal{B}$ verifies the signature. If the test fails, the protocol is aborted. Otherwise, $\mathcal{B}$ chooses $y \in \mathbb{Z}_q^*$ uniformly at random, computes $Y \leftarrow g^y$, computes $c_\mathcal{B} \leftarrow \mathcal{E}_{pk_E}(X, Y, n, pw_\mathcal{B}, ID_\mathcal{A}, ID_\mathcal{B})$, computes session key $SK_\mathcal{B} \leftarrow H_n(X^y)$, and sends $c_\mathcal{B}$ to $\mathcal{T}$;

4). $\mathcal{T}$ decrypts $c_\mathcal{B}$ and checks $\mathcal{B}$'s password, the identities, and the values of X and n. If any test fails, the protocol is aborted. Otherwise, $\mathcal{T}$ computes $\sigma_{\mathcal{T}2} \leftarrow \mathcal{S}_{sk_S}(X, Y, n, ID_\mathcal{A}, ID_\mathcal{B})$, and sends $(X, Y, n, ID_\mathcal{A}, ID_\mathcal{B}, \sigma_{\mathcal{T}2})$ to $\mathcal{A}$;

5). $\mathcal{A}$ verifies the signature against the data received and the value of the group element it first sent. If any test fails, it refuses to generate a session key. Otherwise, $\mathcal{A}$ computes session key $SK_\mathcal{A} \leftarrow H_n(Y^x)$.

Why Proving the Security of Prot0 Apparently Fails Rather unsurprisingly, one of the assumptions we were hoping to rely on here is the *Decisional Diffie-Hellman assumption* (DDH, see the appendix for precise definitions). Combined with the *entropy smoothing* (see the appendix) property of the hash family $\{H_n\}_n$, we can roughly assert that for randomly chosen exponents x and y, a randomly chosen hash key n, and a randomly chosen ℓ-bit string R, the tuples

$$\big(g, g^x, g^y, n, H_n(g^{xy})\big) \ \textit{and} \ (g, g^x, g^y, n, R)$$

cannot be told apart. How this should be used in the security proof is straightforward: whenever the real-world adversary $\mathcal{M}$ has messages correctly delivered between instances, the session keys that are assigned are replaced with random strings. To show indistinguishability of transcripts, a candidate distinguishing algorithm $\mathcal{D}$ will ask first for challenge group elements $X \leftarrow g^x$ and $Y \leftarrow g^y$ and a hash key n to compute the message flows, and then ask for a challenge string either equal to $H_n(g^{xy})$ or random R depending on the value of some hidden bit.

Consider the following situation. The adversary $\mathcal{M}$ has a connecting instance (i', j') send the first message to a server instance $(0, k)$, and has the server instance's response forwarded to an opening instance (i, j). At this point, according to the protocol description, (i, j) accepts the second message, is opened for connection from (i', j') through $(0, k)$, and generates a session key SK_{ij}. This session key is requested as a challenge string by $\mathcal{D}$, who has also already requested X, Y, and n to compute the sent messages. Suppose now that $\mathcal{M}$ has guessed i's password pw_i, and uses this knowledge to replace (i, j)'s message response with an encryption of some weird group element U. If this is delivered to $(0, k)$, it will have to be accepted because the password check passes. If the fourth protocol message is then further delivered to (i', j'), it will be accepted as well. The problem now is assigning (i, j) its session key. Since the adversary chose U, we cannot rule out that it may be able to compute $H_n(X^u)$ (where $u := log_h(U)$), so this is the exact value that $\mathcal{D}$ needs SK_{ij} to be. $\mathcal{D}$'s only hope to compute this is to ask its challenger for x and compute $H_n(U^x)$; unfortunately, this trivially breaks the DDH challenge because $SK_{i'j'}$ - which is equal to $H_n(g^{xy})$ or a randomly chosen R - was already previously queried.

We could ignore this scenario if we could somehow argue that its probability of occurrence during an execution is negligible. Unfortunately, this is not the case: what allows $\mathcal{M}$ to inject U into the message flows is that it *guessed i's password.* Since D is presumed to be a small set of values, guessing pw_i can most certainly not be considered a negligible-probability event.

This further illustrates the subtlety in handling password-based authentication. Protocols **Prot1** and **Prot2** (see 5.3 and 6.3) do not have this problem; it is fixed at the cost of adding key confirmation, which incidentally also provides mutual authentication.

5 Password-Adaptive Network Adversaries

Here, we extend our model by adding a *"reveal password" operation.* This of course models password leakage outside of the key exchange protocol itself, through e.g. password mismanagement. The adversary can now dynamically corrupt users, allowing us to capture *user forward secrecy.* We then show that by including a key confirmation flow to **Prot0**, we obtain a provably secure protocol – **Prot1** – in the enriched model, under the liberal exposure rule.

Let us mention that intuitively, there should be some sort of connection between the model obtained here and the static network adversary model because in practice, password leakage does occur in the static model as well. We comment on this observation in paragraph 5.4.

5.1 The Password-Adaptive Ideal World

Here, and in section 6.1, we only show how the static network adversary model is modified to accommodate additional adversarial capabilities. We basically need to add a query that allows the adversary to reveal passwords for users of its choice. We also need to slightly modify the conditions under which a user instance can be exposed.

Reveal password, i

$\mathcal{M}^*$ specifies a previously initialized user i, and receives pw_i from $\mathcal{RM}^*$. i must not have already been the target of a successful password guess.

We shall say that *user i's password has been revealed* if i has been the target of a "reveal password" operation. We slightly change our previous terminology: from now on, we shall say that the password of an initialized user i is *known* if it either has been successfully guessed, or if i's password has been revealed.

In Shoup's paper [35], the analogous query is not accompanied by any secret user information in the ideal world, because there is none. Our situation is different since we explicitly placed passwords in the ideal world, following [8]. Accordingly, this extra information has a role to play in the security proof.

Transcript: $(\text{"}reveal\ password\text{"}, i, pw_i)$

A modification of the rules for testing passwords

The "text instance password" operation is no longer used on input i if pw_i was already the target of a "reveal password" query.

A modification of the rules for exposing

The exposure rule must also be modified to accommodate the adversary's new capability. When a "start session" operation is performed on user instance (i, j) with $role_{ij} = open$ (resp., $connect$), this user may be exposed through $(0, k)$ if $(0, k)$ has been initialized with $PIDS_{0k} = (ID_i, PID_{ij})$ (resp., $(0, k)$'s exchange has been completed with $PIDS_{0k} = (PID_{ij}, ID_i)$), and either PID_{ij} is not the identity of an initialized user, or PID_{ij} is the identity of an initialized user whose password is known, or pw_i is known.

Distinguishing the **liberal** and **conservative** exposure rules here is of course still relevant. As was the case in section 4, to get the conservative rule from the liberal one stated above, one simply needs to remove the condition "or pw_i is known". Protocol **Prot1** described in 5.3 is actually provably secure under the liberal rule, and insecure under the conservative rule, whereas **Prot2**, described in 6.3, is secure under the conservative rule. The difference that this makes is significant: **Prot2 is secure against KCI while Prot1 is apparently not.**

5.2 The Password-Adaptive Real World

Reveal password, i

$\mathcal{M}$ specifies a previously initialized user i, and receives pw_i from $\mathcal{RM}$. i must not have already been the target of a successful password guess.

The terminology set in terms of passwords being known or revealed in our description of the ideal world carries over to the real world.

Transcript: $(''reveal\ password'', i, pw_i)$

Note that it is unnecessary to add an implementation operation to specify the password, since it is already specified in both the ideal and real worlds.

5.3 Prot1: Adding a Confirmation Code to Prot0

While **Prot0** is not provably secure, it is possible to slightly modify it to get a provably secure protocol in the password-adaptive network adversary model. We simply add a key confirmation mechanism. The same fix is used by Shoup to get from **DHKE** to **DHKE-1** in [35]. The obstruction we described in trying to prove **Prot0** secure is actually exactly the same. The only difference is that since we are dealing with passwords, the problem arises for us already in the static adversary case, while in [35] the protocol **DHKE** is secure in the static-adversary model and not provably secure in the stronger model allowing dynamic user corruptions.

Setup η again being our security parameter, let G be a group of prime order q (of bitlength η), g be a generator of G, $Enc := (\mathcal{K}_E, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme, and $Sig := (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$ be a public-key signature scheme. Run $\mathcal{K}_E(1^\eta)$ to get (pk_E, sk_E) and $\mathcal{K}_S(1^\eta)$ to get (pk_S, sk_S). Set $sk_{\mathcal{T}} := (sk_E, sk_S)$ and $pk_{\mathcal{T}} := (pk_E, pk_S)$. Finally, let $\{H_n\}_n$ be a family of universal hash functions, mapping into $\{0,1\}^{\ell_{SK}+\ell_{CC}}$, where both ℓ_{SK} and ℓ_{CC} are long enough in η ($\frac{1}{2^{\ell_{SK}}}$ and $\frac{1}{2^{\ell_{CC}}}$ are negligible in η). Session keys will be of length ℓ_{SK} and confirmation codes will be of length ℓ_{CC}. In what follows, "randomly" means "uniformly at random".

Running Prot1 between $\mathcal{A}$lice and $\mathcal{B}$ob

1). $\mathcal{A}$ chooses exponent $x \in \mathbb{Z}_q^*$ randomly, computes $X \leftarrow g^x$, computes $c_{\mathcal{A}} \leftarrow \mathcal{E}_{pk_E}(X, pw_{\mathcal{A}}, ID_{\mathcal{A}}, ID_{\mathcal{B}})$, and sends $c_{\mathcal{A}}$ to $\mathcal{T}$;

2). $\mathcal{T}$ decrypts $c_{\mathcal{A}}$ and checks $\mathcal{A}$'s password. If the check passes, it chooses a hash index n randomly, computes $\sigma_{\mathcal{T}1} \leftarrow \mathcal{S}_{sk_S}(n, X, ID_{\mathcal{A}}, ID_{\mathcal{B}})$, and sends $(X, n, ID_{\mathcal{A}}, ID_{\mathcal{B}}, \sigma_{\mathcal{T}1})$ to $\mathcal{B}$;

3). $\mathcal{B}$ verifies the signature. If this check passes, it chooses $y \in \mathbb{Z}_q^*$ randomly, computes $Y \leftarrow g^y$, computes ciphertext $c_{\mathcal{B}} \leftarrow \mathcal{E}_{pk_E}(X, Y, n, pw_{\mathcal{B}}, ID_{\mathcal{A}}, ID_{\mathcal{B}})$, and sends $c_{\mathcal{B}}$ to $\mathcal{T}$;

4). $\mathcal{T}$ decrypts $c_{\mathcal{B}}$ and checks $\mathcal{B}$'s password, the identities, and the values X and n. $\mathcal{T}$ computes $\sigma_{\mathcal{T}2} \leftarrow \mathcal{S}_{sk_S}(X, Y, n, ID_{\mathcal{A}}, ID_{\mathcal{B}})$, and sends $(X, Y, n, ID_{\mathcal{A}}, ID_{\mathcal{B}}, \sigma_{\mathcal{T}2})$ to $\mathcal{A}$;

5). $\mathcal{A}$ verifies the value of the group element it sent in the first protocol message and verifies the signature against the data received. If both checks pass, it computes master key $MK_{\mathcal{A}} \leftarrow H_n(Y^x)$, and then parses it into one bitstring $SK_{\mathcal{A}} \in \{1,0\}^{\ell_{SK}}$ and $\kappa_{\mathcal{A}} \in \{0,1\}^{\ell_{CC}}$. $SK_{\mathcal{A}}$ is the session key, and $\kappa_{\mathcal{A}}$ is the confirmation code. $\kappa_{\mathcal{A}}$ is sent to $\mathcal{B}$;

6). $\mathcal{B}$ computes master key $MK_{\mathcal{B}} \leftarrow H_n(X^y)$, parses it into two strings $SK \in \{0,1\}^{\ell_{SK}}$ and $\kappa_{\mathcal{B}} \in \{0,1\}^{\ell_{CC}}$. If $\kappa_{\mathcal{B}} \neq \kappa_{\mathcal{A}}$, $\mathcal{B}$ stops. Otherwise, $\mathcal{B}$ sets $SK_{\mathcal{B}} \leftarrow SK$.

Theorem 1 *If Enc is IND-CCA-2-secure, Sig is EU-ACMA-secure, and the DDH assumption holds in G,* ***Prot1*** *is secure under the liberal exposure rule against password-adaptive network adversaries.*

The complete proof of this theorem is in section 8. The definitions of IND-CCA-2 security, EU-ACMA-security, and the DDH assumption are in appendices B.1, B.2, and A.1. ■

We comment on the failure of **Prot1** to completely defend against KCI.

The proof of security zeros in on the type of KCI attack which is possible in principle. One example is as follows, in $\mathcal{A}$lice and $\mathcal{B}$ob notation. The attack will involve two different instances of the server, denoted $(\mathcal{T},1)$ and $(\mathcal{T},2)$, one instance of $\mathcal{A}$, and one instance of $\mathcal{B}$.

Steps **1).**, **2).**, **3).**, and **4).** are performed as normal between $\mathcal{A}$ and $\mathcal{B}$ through $(\mathcal{T},1)$. At this point, $\mathcal{A}$ has chosen $X = g^x$, $(\mathcal{T},1)$ has chosen n_1, and $\mathcal{B}$ has chosen $Y = g^y$. Since step 4) was carried out, a signed message containing n_1 and Y is "on its way" to $\mathcal{A}$.

Now suppose that our network adversary $\mathcal{M}$ intercepts this fourth message. While $\mathcal{A}$ is waiting, $\mathcal{M}$ replays to $(\mathcal{T},2)$ the first message that $\mathcal{A}$ sent. $(\mathcal{T},2)$ - a new server instance - has no reason to reject this replayed message, so it chooses a new n_2, and sends out a new second message which reveals n_2.

Next, assume that $\mathcal{M}$ knows $pw_{\mathcal{B}}$. Suppose that it uses this knowledge to compute a third protocol message c of the form $c \leftarrow \mathcal{E}_{pk_E}(X, V, n_2, pw_{\mathcal{B}}, ID_{\mathcal{A}}, ID_{\mathcal{B}})$, where V is some group element that may depend in some weird way on X and Y_1. $\mathcal{M}$ now has c delivered to $(\mathcal{T},2)$, which computes a fourth protocol message, that $\mathcal{M}$ finally has delivered to $\mathcal{A}$.

$\mathcal{A}$ has been waiting for this message; $\mathcal{A}$ accepts it, since all checks pass, and computes $SK_{\mathcal{A}}||\kappa_{\mathcal{A}} \leftarrow H_{n_2}(V^x)$.

Finally, suppose that $\mathcal{A}$ starts using $SK_{\mathcal{A}}$, and that $\mathcal{M}$ somehow gets the entire session key. $\mathcal{M}$ can now reconstruct the entire string $H_{n_2}(V^x)$. This is where $\mathcal{B}$ is at risk because the assumptions made on G and $\{H_n\}_n$ do not guarantee that $H_{n_1}(g^{xy})$ is incomputable in this case: $\mathcal{M}$ has at its disposal $\big(X, Y, V, n_1, n_2, H_{n_2}(V^x)\big)$ *where V is a mysterious function of X and Y*. In particular, $\mathcal{M}$ could very well compute the confirmation code $\mathcal{B}$ expects, thereby having $\mathcal{B}$ accept despite $\mathcal{A}$ having computed a different key.

In the end, formally $\mathcal{M}$ has indeed successfully impersonated $\mathcal{A}$ to $\mathcal{B}$ using knowledge of $pw_{\mathcal{B}}$. Notice however that in reality, $\mathcal{M}$ has to work a lot more to succeed in this extremely convoluted attack beyond somehow getting $pw_{\mathcal{B}}$. Remember, revealing $pw_{\mathcal{B}}$ is only a sufficient prerequisite for exposing (in the *liberal* sense) an instance of $\mathcal{B}$ *in the ideal model*; this does not mean that it is sufficient in reality *for a given protocol.*

It turns out that this attack and variations of it are the only KCI scenarios possible against **Prot1**. This is why it is reasonable to state that while this protocol does not fully eliminate KCI, it does heavily mitigate it.

5.4 A Comment on the Relation Between the Static and Password-Adaptive Models

Usually, in security models for classical key exchange using public-key infrastructures, the difference made by allowing long-term key corruptions is quite stark. Some protocols secure in the static sense are immediately and blatantly broken as soon as such a query is permitted (e.g. Shoup's **EKE** protocol in [35], **not to be confused with Bellovin and Merrit's "Encrypted Key Exchange" protocol [6]**). When passwords are used as long-term secrets, the difference is not so obvious because even in the static adversary case, password guessing can be successful, and the advantage the adversary is granted in the protocol execution is essentially the same. One might therefore postulate that on some level, both models are equivalent.

Our feeling is that this is probably not the case, for several reasons. First, it would be fundamentally unsound because password guesses and corruption queries model two completely different types of information leakage which in practice have nothing to do with one another. In the language developed in this paper, the corresponding theoretical discrepancy can be described as follows. If we were to try to simulate a model with password-revealing queries in a model without them, we would have no choice but to simulate password-revealing requests with an amount of password guesses sufficient to give the right answer. But if we hope to show indistinguishability of transcripts, all of these password guesses would have to be logged in **both** models, making the adversary of the adaptive case basically do as much work for his reveal query as he would by just performing successive authentication attempts.

The difference would be more visible if we replaced our *asymptotic* security definitions with *exact* security ones; we could then obtain formulas expressing the adversary's advantage as functions of the number of guessing attacks and the number of password reveal queries.

6 Password-and-State-Adaptive Network Adversaries

In this section, we further enhance the model by granting the adversary the power to corrupt user instances in the following sense: upon corruption, a user instance's *Unerased Internal State* (UIS) is revealed. This models storage of ephemeral data in insecure memory (see [35]). This is a particularly interesting query to study because revealing internal state is seldom considered for password protocols since it leads to immediate dictionary attacks. This appears to not systematically be the case here precisely because we have private keys at the server. We note however that it is crucial that certain random bits be erased as soon as they have served their purpose.

Our goal is this: if a user instance's UIS *alone* is revealed, that instance's session key, **and at most one other instance's session key**, is compromised.

This "weak" definition of internal state revealing and the corresponding security goal were first exhibited by Shoup [35].

Let us pause to state what this query does *not* reveal by definition. First of all, it does not ***formally*** reveal the session key, for this key does not even "exist" until the UIS has served its purpose in the protocol. Secondly, it obviously does not reveal the password[2]. This can be concretely interpreted as follows: in reality, a password should be human-memorable. In particular, one can assume that it becomes input to the protocol only when the user types it into whatever interface is implemented by the program. Thus, the password has no business being in intermediate memory anyway. Following this line of thought, if one actually cares about UIS revealing it seems a prudent design principle that either a). a password should only be used as input once in a protocol or b). if a password needs to be used in several protocol computations separated by message deliveries, the user should have to retype her password in for each of these computations. Clearly, the latter option raises usability concerns.

Section 6.3 describes **Prot2**, which is provably secure in this model. **Prot0** and **Prot1** are insecure in this sense, but still achieve something from a UIS point-of-view with respect to the passwords, see section 6.5 for further discussion.

6.1 The Password-and-State-Adaptive Ideal World

Corrupt instance, (i, j)

(i, j) should not be terminated and should not have started a session. If (i, j) has been the target of this operation, we shall simply say that (i, j) *has been corrupted.* This terminology applies in the real world (described below) as well. During the execution, an instance that has been corrupted is either *unbound* or

[2] We warn the reader however that these notions are not those from the latest version of [35]. Taking the definitions of the latest version - version 4, which is from November 1999 - would not be suitable for our purpose as instance corruptions in that case automatically reveal long-term secrets as well. If we were to do this, we could obviously not hope to protect the password. The definitions we use can be found in the April 1999 version of [35].

bound. Upon corruption, it starts out unbound. Also, if an instance is bound and is later corrupted again, it remains bound.

Transcript: $\big(\text{"}\textit{corrupt }\text{"}, (i,j)\big)$

A further modification of the rules for exposing

We first allow (i,j) to be exposed if it has been corrupted. This is the *relaxed exposure rule.* If (i,j) was already connected using the special connection rule defined below, it remains so.

Next, if there exist $(0,k)$ and (i',j') such that (i,j), (i',j'), and $(0,k)$ have matching roles and partners, (i,j) may be exposed if (i',j') is corrupted and unbound. (i',j') then becomes bound. This is the *special exposure rule.*

A special premature connecting rule

Let (i,j) have the role *connect*, and let (i',j') and $(0,k)$ be such that (i',j') is open for connection from (i,j) through $(0,k)$. If (i,j) is corrupted and unbound, we allow the adversary to prematurely connect (i,j) to (i',j') through $(0,k)$. In this case, $\mathcal{M}^*$ receives $SK_{i'j'}$, and (i,j) becomes bound. This is the *special connection rule.*

The purpose of binding instances The mechanism enforcing the fact that at most one additional instance is compromised solely due to a corruption is the attribution of the "bound" or "unbound" status to a corrupted instance. Intuitively, if in the course of a simulation a user instance must absolutely be exposed but its natural partner instance is already bound, the logic of the protocol should imply that a relevant password is known to the adversary, thereby allowing an ordinary exposure to take place.

6.2 The Password-and-State-Adaptive Real World

In the real world, instance corruptions are treated as follows.

A modification of user instances and their message deliveries

When a user instance (i,j) is initialized, in addition to the random variable $status_{ij}$, the random variable $InternalState_{ij}$ is initialized outside of $\mathcal{M}$'s view and at first set to ε (the empty string). $InternalState_{ij}$ is updated at each message delivery. If $status_{ij} = accept$, $InternalState_{ij}$ is set once again to ε.

Corrupt instance, (i,j)

$\mathcal{M}$ specifies an initialized and not yet terminated user instance, and recovers the value of $InternalState_{ij}$.

Transcript: $\big(\text{"}\textit{corrupt}\text{"}, (i,j)\big)$ followed by
$\big(\text{"}\textit{implementation}\text{"}, \text{"}\textit{internal state}\text{"}, (i,j), InternalState_{ij}\big)$

The $InternalState_{ij}$ **Random Variable** The values this variable takes will be specified in the real-world protocol description. According to the definition, the adversary recovers the value of $InternalState_{ij}$ at the time of corruption. This may very well be an empty value at times, but notice that in neither the ideal nor real worlds do we prohibit the adversary from corrupting an instance more than once. Recall that in the ideal world, if an instance that was corrupted becomes bound and is later corrupted again, it remains bound.

Revisiting Dictionary Attacks The comments made in section 4.2 still apply here. Ideally, corrupting user instances has no effect on an adversary's ability to test password guesses beyond the one or two online impersonations that can be performed on a server instance. This accurately captures the idea that dictionary attacks are not aided by UIS revealing. But this is just a definition; it does not show how a protocol concretely achieves this... (See the next paragraph.)

6.3 Prot2: Binding the Random Choices to Both Passwords

The setup is the same as for **Prot1**, except that the H_n map into $\{0,1\}^{\ell_{SK}}$. **Prot2** runs as follows. Also, we explicitly assign a value to the the internal state variable in the protocol description. Let ε denote the empty string.

1). $\mathcal{A}$ selects $x \in \mathbb{Z}_q^*$ randomly, computes $X \leftarrow g^x$, and sends $(X, ID_\mathcal{A}, ID_\mathcal{B})$ to $\mathcal{T}$. Here, $\mathcal{A}$'s state is set to $InternalState_\mathcal{A} \leftarrow (x, X)$;

2). $\mathcal{T}$ selects a hash index n randomly, and sends $(X, n, ID_\mathcal{A}, ID_\mathcal{B})$ to $\mathcal{B}$.

3). $\mathcal{B}$ selects $y \in \mathbb{Z}_q^*$, computes $Y \leftarrow g^y$ and $MK_\mathcal{B} \leftarrow H_n(X^y)$, ***erases*** y, and computes the ciphertext $c_\mathcal{B} \leftarrow \mathcal{E}_{pk_E}(1, X, Y, n, pw_\mathcal{B}, ID_\mathcal{A}, ID_\mathcal{B})$. ***It erases the randomness used to compute*** $c_\mathcal{B}$, and sends $c_\mathcal{B}$ to $\mathcal{T}$. Here, $\mathcal{B}$'s internal state is set to $InternalState_\mathcal{B} \leftarrow (X, Y, n, MK_\mathcal{B})$;

4). $\mathcal{T}$ decrypts $c_\mathcal{B}$ and checks the password $pw_\mathcal{B}$, the identities, and the random values n and X. If the checks pass, it computes $\sigma_{\mathcal{T}1} \leftarrow \mathcal{S}_{sk_S}(1, X, Y, n, ID_\mathcal{A}, ID_\mathcal{B})$, and sends $(Y, n, \sigma_{\mathcal{T}1})$ to $\mathcal{A}$;

5). $\mathcal{A}$ checks the signature $\sigma_{\mathcal{T}1}$ against the data received and the held group element X. If the test passes, it computes $SK_\mathcal{A} \leftarrow H_n(Y^x)$, ***erases*** exponent x, and computes $c_\mathcal{A} \leftarrow \mathcal{E}_{pk_E}(2, X, Y, n, pw_\mathcal{A}, ID_\mathcal{A}, ID_\mathcal{B})$. ***It erases the randomness used to compute*** $c_\mathcal{A}$ and sends $c_\mathcal{A}$ to $\mathcal{T}$. $\mathcal{A}$'s internal state is reset to $InternalState_\mathcal{A} \leftarrow \varepsilon$;

6). $\mathcal{T}$ decrypts $c_\mathcal{A}$, checks password $pw_\mathcal{A}$, the identities, and checks the random values X, Y, and n. It computes $\sigma_{\mathcal{T}2} \leftarrow \mathcal{S}_{sk_S}(2, X, Y, n, ID_\mathcal{A}, ID_\mathcal{B})$, and sends $(Y, \sigma_{\mathcal{T}2})$ to $\mathcal{B}$.

7). $\mathcal{B}$ checks the value of Y, and then checks the signature $\sigma_{\mathcal{T}2}$ against the data received and the held originator element X and hash index n. It sets $SK_\mathcal{B} \leftarrow MK_\mathcal{B}$. Finally, $\mathcal{B}$'s internal state is rest to $InternalState_\mathcal{B} \leftarrow \varepsilon$.

Theorem 2 *If Enc is IND-CCA-2-secure, Sig is EU-ACMA-secure, and the DDH assumption holds in G,* ***Prot2*** *is secure under the conservative exposure rule against password-and-state-adaptive network adversaries.*

The full proof of the theorem is in section 9, and the definitions of IND-CCA-2 security, EU-ACMA security, and the DDH assumption, are in the appendix (see B.1, B.2, and A.1). ■

Since the theorem holds under the conservative exposure rule, **Prot2** fully avoids KCI. What about UIS revealing?

Revisiting Dictionary Attacks *Again* UIS is basically "whatever information is needed for an instance to continue its computations". Everything else – as specified in the protocol description above – is *erased* as soon as it is no longer needed. In particular, what (say) $\mathcal{A}$ needs to hold on to after sending its first protocol message is its exponent x, otherwise it cannot compute the session key. What it does not need to hold on to is the randomness used in the public-key encryption performed to compute the fifth protocol message in step 5). Hence this randomness is erased and therefore off limits to the UIS revealing. But the fact that the encryption is randomized is exactly what protects the password, even if the other parts of the plaintext are known to the attacker. This is the *very definition* of semantic security.

It is worth contrasting this with the situation in CRS-based 2-PAKEs that are standard-model-secure and that use smooth projective hashing. Such protocols also protect the password with public-key encryption. Recall that decryption in this setting is impossible; nobody even has the decryption key. Now, this problem is circumvented using the smooth projective hashing mechanism. But this is where the catch lies in terms of internal state: for the mechanism to work, the parties *need to hold on to the randomness used for encryption, and it becomes UIS.* Thus, revealing UIS causes trivial dictionary attacks.

Comparing UIS According to Role It is interesting to note the inherent asymmetry that the initiator and responders have in this protocol as far as UIS is concerned. Arguably, the most striking observation is that the protocol initiator (i.e. $\mathcal{A}$) needs to hold on to its random exponent between two messages, while the responder (i.e. $\mathcal{B}$) does not. More interestingly, if we try to "force" symmetry to occur, e.g. by asking $\mathcal{B}$ to hold on to its exponent y and only compute the session key at the end, ***we lose simulatability***. See 6.4 for an explanation of this.

Replaying Stale Data As for the security goal that we aim to achieve with UIS revealing, it is indeed reached by this protocol intuitively because the random group elements of both instances are bound to both passwords. In particular, if an adversary who gets the exponent x of an originator wants to successfully replay

$X = g^x$ in a protocol run, it can only do so through knowledge of the originator's password because the fifth protocol message contains an encryption of the new responder's fresh group element. A stale message cannot therefore simply be replayed.

6.4 Variations of Unerased Internal State in Prot2

It is instructive to see the kind of obstruction that arises when using a simulatability definition for different definitions of the UIS held by the user instance with the role "connect". In "$\mathcal{A}$lice and $\mathcal{B}$ob" notation, $\mathcal{B}$ has the role "connect".

Suppose first that upon receipt of the third protocol message, $\mathcal{B}$ does not compute $H_n(X^y)$ directly before erasing y, but instead waits until it has checked the signature on the sixth message to do so. Here, $X = g^x$ is the group element (purportedly) chosen by $\mathcal{A}$, $Y = g^y$ is the group element and hash index chosen by $\mathcal{B}$, and n is the hash index (purportedly) chosen by $\mathcal{T}$. Consider the case where the first four messages are correctly computed and delivered. $\mathcal{A}$ having received the fourth message, she has computed the session key. Since all messages were faithfully delivered and neither $\mathcal{A}$ nor $\mathcal{B}$ has been corrupted, our simulation-based security definition has had $\mathcal{A}$ opened for connection from $\mathcal{B}$, and the session key $SK_{\mathcal{A}}$ she has been assigned is a random string. But if $\mathcal{B}$ is corrupted at this point, the adversary $\mathcal{M}$ learns the exponent y in addition to having X and n. Thus, $\mathcal{M}$ will have no trouble deciding whether $SK_{\mathcal{A}} = H_n(Y^x)$ or not.

6.5 Unerased Internal State in Prot0 and Prot1

We return briefly to **Prot0** and **Prot1** to see what happens to them under UIS revealing. For this, we need to specify the *InternalState* variable for the users at all stages of the descriptions of both "**Prot**s". It seems that we cannot get away with having the initiator's exponent being a part of *InternalState*, but as with **Prot2**, we can get rid of all of the random bits used for the encryption scheme.

Under these assumptions, clearly neither protocol fulfills the security goal, for if the exponent x chosen by $\mathcal{A}$ is revealed to adversary $\mathcal{M}$, $\mathcal{M}$ can replay the first protocol message to many new server instances communicating with many other $\mathcal{B}$ instances and compute the correct session key (and, in the case of **Prot1**, confirmation code) each time, without $\mathcal{M}$ ever needing to reveal $pw_{\mathcal{A}}$ or $pw_{\mathcal{B}}$.

However, even though the desired goal defined in case of UIS revealing is not achieved, UIS revealing *still fails to open either protocol to dictionary attacks*, for the exact same reason as in the case of **Prot2**: no reasonable definition of UIS for **Prot0** or **Prot1** will contain the encryption's random bits. Of course, it would be better to actually *prove* this in an adequate model, which we do not do here. Since we see no way to prove it secure even in a basic sense, we have no real idea on how this could be done for **Prot0**. We nevertheless give some indications on how it might be possible to proceed for **Prot1** (or similar protocols).

Recall that the ideal-world device that limits the number of compromised instances in the face of an instance corruption is the fact that a corrupted instance may become bound (see section 6.1). The corruption operation alone does not suffice. Therefore, to capture the idea that revealing UIS does not open a protocol to dictionary attacks, we simply a). strip the ideal world of the "bound" and "unbound" attributes and b). make the rules for exposing only a function of whether a given partnered instance is corrupted. These changes should suffice again by virtue of the fact that in the ideal world, corrupting a user instance does not aid in any way in testing passwords.

7 Preparing for the Proofs of Security

The purpose of this section is to lay out the framework necessary to understand the proofs of security. The full proofs of each theorem are in the two following sections, and each of these begins with a proof sketch. Here we fix the notations used in, and explain the structure of, both the full proofs and the sketches.

Technically speaking, the proof sketches (see section 8.1 for **Prot1** and 9.1 for **Prot2**) add nothing to the full proofs. We have decided to include them anyway because the high-level view they provide **a).** was extremely useful for us to finally pin down the correct arguments developed in the full proofs and **b).** really shows off the "simulation" mechanism. Because of this, we think that that from the reader's point of view these sketches may actually be more instructive than the proofs themselves, which are very long due to the moderately high complexity of the protocols. However, some of the features of the proof sketches can really only be properly explained with the full proofs.

7.1 Some Notations and Terminology

In either protocol, the user that generates the first message is called the *originator*. The other user is called the *responder*. In both **Prot1** and **Prot2**, the originator happens to be the user instance that computes the session key first; therefore, originators will always receive the role *open*, and responders, the role *connect*. We will always try to use (i, j) to designate an originator instance and (i', j') to designate a responder. We will sometimes refer to these as *i-originators* and *i'-responders*. $(0, k)$ will always be used for server instances.

$\mathcal{M}$ is the real-world network adversary. Recall that it is a PPTA in security parameter η. As stated in section 3.2, our objective is to build an ideal-world adversary $\mathcal{M}^*$ whose interaction with the ideal world ring master $\mathcal{RM}^*$ is computationally indistinguishable from an interaction between $\mathcal{M}$ and a real-world ring master $\mathcal{RM}$. To achieve this, we build $\mathcal{M}^*$ by having it simulate $\mathcal{RM}$'s actions while running $\mathcal{M}$ as subroutine. Security is achieved if $\mathcal{M}^*$ can unambiguously translate any real-world action $\mathcal{M}$ takes into a corresponding legal ideal-world action. In particular, any of $\mathcal{M}$'s successful real-world attacks should correspond to an unavoidable ideal-world attack.

7.2 Structure of the Proof Sketches

The proof sketches make only mention of $\mathcal{M}^*$, $\mathcal{M}$ as it is used by $\mathcal{M}^*$, and $\mathcal{RM}^*$. $\mathcal{RM}$ is "played by" $\mathcal{M}^*$ for $\mathcal{M}$. In these sketches, we directly show how important events in the real world correspond to events in the ideal world. The most significant real-world events that need to be translated into ideal-world ones are those message deliveries that

a). lead to session keys being computed by user instances: these should be interpreted as "start session" operations with corresponding, unique connection assignments, and

b). constitute real-world, online password guesses: these should be interpreted in the ideal world as "test instance password" operations.

Of course, the security of the primitives used as protocol ingredients is invoked to show that the ideal-world requirements for operations to take place are indeed satisfied. The fact that these security properties are informally exploited is what makes these demonstrations only sketches.

7.3 More Notations, Structure of the Full Proofs, and Simplifying Assumptions on the Encryption Scheme

The full security proofs carry out the above simulation in much more detail, making explicit the precise role each primitive has in securing the protocol. These proofs proceed as sequences of games, a very common technique in provable security.

The first game in the sequence is played between $\mathcal{M}$ and $\mathcal{RM}$ according to the precise real-world rules. The intermediate games introduce small changes to the way $\mathcal{RM}$ runs the execution environment, changes that the adversary should not be able to computationally notice based on the security properties of the various building blocks in the protocol. Each game mainly exploits the exact security definition of one of the protocol's features. With each change, the adversary's wiggle-room becomes more restricted up until the second-to-last game, where $\mathcal{M}$ really has no more freedom to attack in a meaningful way. The very last game is used to classify $\mathcal{M}$'s remaining available actions into ideal-world operations, thereby completing the simulation. It is in this last game that the fully modified ring master is converted into a simulator $\mathcal{M}^*$.

Since the intermediate games exhibit properties that fall somewhere in between those of the real and ideal worlds, we call them *hybrid* worlds. Each hybrid world has a transcript of its own that follows the same logging rules as in the real world. These hybrid-world transcripts are of course used to measure the adversary's ability to discern consecutive worlds.

For $a \in \mathbb{N}$, the hybrid-world ring master in the a-th world will be denoted $\mathcal{RM}_a^h$. In the real world and in all of the hybrid worlds, the adversary will always be denoted $\mathcal{M}$. The transcript random variable of the a-th hybrid world will be denoted $\mathcal{HW}_a(\mathcal{M})$.

We adopt the following (somewhat abusive) terminology. Let $s := (s_1, ..., s_t)$ and $s' := (s'_1, ..., s'_{t'})$ be tuples of strings and let w be either s itself, an encryption $\mathcal{E}_{pk}(s)$ of s, or a signature $\mathcal{S}_{sk}(s)$ of s. If all of the s'_i appear as components of s, we shall say that *w contains s'* or that *s' is contained in w.* We assume that a signature on a message is not automatically accompanied by the message itself.

Group elements that are computed - or purportedly computed - by originator (resp., responder) instances will be referred to as *originator* (resp., *responder*) *group elements.* We shall try to systematically reserve X and U to designate originator group elements and Y and V to designate responder group elements.

We shall say that an integer (function) ℓ of η is *long enough in η* if $\frac{1}{2^\ell}$ is negligible in η. (For instance, "$\ell := \eta$" is long enough in η.)

About the Encryption Scheme... We end this paragraph by making a few benign assumptions on the encryption scheme $Enc = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

First, we shall assume that for any fixed message m and for any public encryption key pk_E, the function $r \mapsto \mathcal{E}_{pk_E}(m; r)$ is injective, where r is the encryption randomness. An example of an IND-CCA-2 secure scheme that satisfies this is the Cramer-Shoup [14] scheme. The assumption is not really limiting because even if it does not strictly hold for *Enc*, the semantic security of the scheme implies that it will hold with overwhelming probability.

Secondly, we shall assume that the function which to a secret decryption key associates the corresponding public key is injective as well. This is also true for Cramer-Shoup [14]. Also, in the event that it is not strictly true, it should be true with overwhelming probability.

Finally, we assume that decryption is perfect. This is rather common; we do not elaborate further.

It will be indicated in the proofs (even sometimes in the sketches) where these assumptions are used. We could avoid making them, but this would only further complicate the proofs below, and add little value for our purpose.

Let the game-hops begin.

8 Prot1 is Secure Against Password-Adaptive Network Adversaries

Paragraph 8.1 below contains the sketch of the proof of security for **Prot1**. All other paragraphs in this section - from 8.2 on - together form the full proof.

8.1 The Proof Sketch for Prot1

$\mathcal{RM}^*$ is the ideal-world ring master, $\mathcal{M}^*$ is the ideal-world adversary under construction, and $\mathcal{M}$ is the real-world adversary $\mathcal{M}^*$ is using as a subroutine. $\mathcal{M}^*$ has to use its queries to $\mathcal{RM}^*$ to answer the queries from $\mathcal{M}$. In what follows, (i, j) will always be an originator instance, $(0, k)$ will always be a server instance, and (i', j') will always be a responder instance.

In this sketch, we only show how message deliveries are handled, since this is the crucial point in the simulation. The other queries $\mathcal{M}$ may make have direct counterparts in the ideal world. Hence, these are simply forwarded to $\mathcal{RM}^*$ by $\mathcal{M}^*$ on the same input. We also omit the setup phase.

Text in italics is used every now and then to explain informally how some of the properties of the protocol's constructs are exploited.

The case of an originator instance

- **Generating the first protocol message**

Suppose $\mathcal{M}$ asks to have (i,j) generate the first protocol message. $\mathcal{M}^*$ selects exponent $x \leftarrow \mathbb{Z}_q^*$ uniformly at random, computes $X \leftarrow g^x$, and computes $c \leftarrow \mathcal{E}_{pk_E}(1_G, 1^{\ell_{pw}}, ID_i, PID_{ij})$. It then outputs $OutMsg_1 \leftarrow c$ to $\mathcal{M}$.

By the semantic security of the encryption scheme, $\mathcal{M}$'s behavior will not change even though $(1_G, 1^{\ell_{pw}})$ is encrypted rather than (X, pw_i). It is important that this substitution be made, for a priori pw_i is not available to $\mathcal{M}^*$; indeed $\mathcal{RM}^*$ selects passwords outside of $\mathcal{M}^*$'s view. In contrast with the reason $1^{\ell_{pw}}$ takes the password's place, the technical reason $\mathcal{M}^*$ substitutes 1_G for X has nothing to do with the specification of the ideal-world, and is made clear in section 8.6. For the purpose of the proof sketch, it is not really necessary to do this, but we choose to anyway to stay consistent with the ideal-world adversary built in the full proof.

- **Receiving the fourth protocol message and computing the fifth**

Suppose (i,j) receives the fourth protocol message $InMsg_4$. Let $OutMsg_1$ be the first protocol message output by (i,j) and let X be the group element chosen by (i,j).

•• Suppose first that no server instance $(0, k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, PID_{ij})$ received $OutMsg_1$. In this case, (i,j) simply rejects $InMsg_4$ and terminates.

This is justified because given that the first protocol message is non-malleably encrypted (remember, Enc is IND-CCA-2-secure), $\mathcal{M}$'s only hope to get any information on X is by delivering exactly $OutMsg_1$ to such a server instance.

•• Suppose now that there exists a server instance $(0, k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, PID_{ij})$ that received $OutMsg_1$, thereby revealing X to $\mathcal{M}$. If $InMsg_4$ is not of the correct format - i.e. of the form (X, V, n, σ) for a non-trivial group element V, a hash index n, and a string σ - (i,j) rejects and terminates. Otherwise (i,j) computes the signature verification equation on input $(X, V, n, ID_i, PID_{ij}, \sigma)$.

••• If the verification fails, (i,j) rejects $InMsg_4$ and terminates.

••• If the verification succeeds, the unforgeability of the signature scheme and the size of the hash key space imply that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that chose n and computed a signature on input $(X, V, n, ID_i, PID_{ij})$ to output a second protocol message $OutMsg_2$. Also, $(0,k)$ must have received as a first protocol message either $OutMsg_1$, or some $InMsg_1$ containing (X, pw_i). (i,j)'s session is started, and we now need to determine what its connection assignment will be.

•••• Suppose that $PID_{ij} = ID$ where ID was not assigned to a user. In this case, (i,j) is exposed through $(0,k)$. $\mathcal{M}^*$ extracts master key $MK_{ij} \leftarrow H_n(V^x)$, where $g^x = X$, specifies $SK_{ij}^* \leftarrow pf_{\ell_{SK}}(MK_{ij})$ to $\mathcal{RM}^*$, and outputs $OutMsg_5 \leftarrow sf_{\ell_{CC}}(MK_{ij})$ to $\mathcal{M}$.

•••• Suppose now that $PID_{ij} = ID_{i'}$ for some initialized user i'. Let $InMsg_3$ be the third protocol message that $(0,k)$ received.

••••• Suppose that no user instance computed $InMsg_3$. By the rules governing the way $\mathcal{M}^*$ runs server instances (see below), $InMsg_3$'s acceptance by $(0,k)$ implies that $\mathcal{M}^*$ either revealed or successfully guessed $pw_{i'}$. (i,j) is exposed through $(0,k)$, $\mathcal{M}^*$ computes $MK_{ij} \leftarrow H_n(V^x)$, specifies $SK_{ij}^* \leftarrow pf_{\ell_{SK}}(MK_{ij})$ to $\mathcal{RM}^*$, and outputs $OutMsg_5 \leftarrow sf_{\ell_{CC}}(MK_{ij})$ to $\mathcal{M}$.

••••• Suppose a user instance did compute $InMsg_3$. The way server instances process the third message (below) implies that it is necessarily a responder (i', j') with $PID_{i'j'} = ID_i$ that received as a second protocol message some $InMsg_2$ containing the same data as $OutMsg_2$. Furthermore, by the size of Enc's randomness space, (i', j') is unique, and we also know that (i', j') chose the group element V. Thus, at this point, (i,j) has participated in a correct conversation with (i', j') through $(0,k)$. (i,j) is therefore opened for connection from (i', j') through $(0,k)$. $\mathcal{RM}^*$ selects SK_{ij}^* uniformly at random from $\{0,1\}^{\ell_{SK}}$, $\mathcal{M}^*$ selects $OutMsg_5$ uniformly at random from $\{0,1\}^{\ell_{CC}}$, and outputs $OutMsg_5$ to $\mathcal{M}$.

This substitution goes unnoticed by $\mathcal{M}$ under the DDH assumption coupled with the entropy smoothing property of the hash family.

The case of a responder instance

- **Receiving the second protocol message and computing the third**

Suppose (i',j') receives the second protocol message $InMsg_2$.

•• If $InMsg_2$ is not of correct format, (i',j') rejects and terminates.

•• If $InMsg_2$ is of correct format, let U and n be the non-trivial group element and hash index present in $InMsg_2$. (i',j') computes the signature verification equation on input $(U, n, PID_{i'j'}, ID_{i'})$.

••• If verification fails, the message is rejected and (i',j') terminates.

••• If verification succeeds, the unforgeability of the signature scheme and the size of the hash index space ensure that there exists a unique server instance $(0,k)$ that computed a second protocol message $OutMsg_2$ on input $(U, n, PID_{i'j'}, ID_{i'})$. In this case, $\mathcal{M}^*$ selects $y \leftarrow \mathbb{Z}_q^*$ uniformly at random, computes $Y \leftarrow g^y$, computes $c' \leftarrow \mathcal{E}_{pk_E}(U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'})$, and outputs $OutMsg_3 \leftarrow c'$ to $\mathcal{M}$.

Again, the security of encryption means that replacing $pw_{i'}$ by $1^{\ell_{pw}}$ and Y by 1_G does not change $\mathcal{M}$'s view. Also again, why Y is also replaced is clarified in section 8.6.

- **Receiving the fifth protocol message**

Suppose (i',j') receives the fifth protocol message $InMsg_5$. Let $InMsg_2$ be the second protocol message (i',j') has received, containing originator group element U and hash index n, let $OutMsg_3$ be the third protocol message computed by (i',j'), and let $Y = g^y$ be the group element chosen by (i',j'). Let $(0,k)$ be the unique server instance that computed a second protocol message $OutMsg_2$ on input $(U, n, PID_{i'j'}, ID_{i'})$. Let $InMsg_1$ be the first protocol message received by $(0,k)$.

•• Suppose that $OutMsg_3$ was not delivered to $(0,k)$. Then (i',j') simply rejects $InMsg_5$.

This is justified because by the non-malleability of the encryption, only through delivery of exactly $OutMsg_3$ to $(0,k)$ can $\mathcal{M}$ get sufficient information to be able to compute the ℓ_{CC}-bit suffix of $H_n(U^y)$. Notice that this is true even if (i',j') is conversing directly with the adversary through $(0,k)$.

•• Suppose now that $OutMsg_3$ was delivered to $(0,k)$, thereby revealing Y to $\mathcal{M}$. $(0,k)$'s exchange is now completed, according to the rules governing the treatment of server instances (below). Let $OutMsg_4$ be the fourth message computed by $(0,k)$.

••• Suppose that $PID_{i'j'} = ID$, where ID was never issued to a user. In this case, $\mathcal{M}^*$ extracts the master key $MK_{i'j'} \leftarrow H_n(U^y)$, computes $\kappa_{i'j'} \leftarrow sf_{\ell_{CC}}(MK_{i'j'})$, and compares $InMsg_5$ to $\kappa_{i'j'}$. If they are equal, (i',j')'s session is started, and (i',j') is exposed through $(0,k)$. Note that this is legal because $OutMsg_3$ was delivered to $(0,k)$, whose exchange is therefore completed. $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow pf_{\ell_{SK}}(MK_{i'j'})$ as (i',j')'s session key. If $InMsg_5 \neq \kappa_{i'j'}$, (i',j') rejects and terminates.

••• Suppose that $PID_{i'j'} = ID_i$ for some initialized user i.

•••• Suppose that the following condition **is not** met: there exists an originator (i,j) with $PID_{ij} = ID_{i'}$ that output a first protocol message $OutMsg_1$, and either $InMsg_1 = OutMsg_1$ or $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ and $InMsg_1$ contains the same originator group element as the second protocol message output by $(0,k^{(1)})$. In this case, $InMsg_1$ was not generated by a user instance, yet was accepted by $(0,k)$. This implies - see the case of server instances below - that $\mathcal{M}^*$ has either revealed or successfully guessed pw_i. Then $\mathcal{M}^*$ extracts the master key $MK_{i'j'} \leftarrow H_n(U^y)$, computes $\kappa_{i'j'} \leftarrow sf_{\ell_{CC}}(MK_{i'j'})$, and compares $InMsg_5$ to $\kappa_{i'j'}$. If they are equal, (i',j')'s session is started, (i',j') is exposed through $(0,k)$, and $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow pf_{\ell_{SK}}(MK_{i'j'})$ as (i',j')'s session key. If they are not equal, (i',j') rejects and terminates.

•••• Suppose the aforementioned condition **is** met, i.e. there exists an originator (i,j) with $PID_{ij} = ID_{i'}$ that output a first protocol message $OutMsg_1$, and either $InMsg_1 = OutMsg_1$ or $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ and $InMsg_1$ contains the same originator group element as the second protocol message output by $(0,k^{(1)})$. Then by the size of *Enc*'s randomness space, (i,j) is unique. Also, U was chosen by (i,j).

••••• Suppose no fourth message was delivered to (i, j). In this case, (i', j') simply rejects $InMsg_5$. This is justified because $\mathcal{M}$ only has at its disposal (U, Y, n) to try to compute $sf_{\ell_{CC}}\big(H_n(g^{uy})\big)$, which cannot be done under the DDH.

••••• Suppose some fourth message $InMsg_4$ was delivered to (i, j).

•••••• Suppose $InMsg_4$ was rejected. In this case also, (i', j') simply rejects $InMsg_5$.

•••••• Suppose now that $InMsg_4$ was accepted by (i, j). Then (i, j)'s session was started. Let V and $n^{(2)}$ be the responder group element and hash index within $InMsg_4$. By the rules describing the actions $\mathcal{M}^*$ takes when dealing with fourth message deliveries to originator instances, there exists a unique server instance $(0, k^{(2)})$ with $PIDS_{0k^{(2)}} = (ID_i, ID_{i'})$ that computed a fourth protocol message $OutMsg_4^{(2)}$ on input $(U, V, n^{(2)})$ with $n^{(2)}$ chosen by $(0, k^{(2)})$. Also, $(0, k^{(2)})$ was delivered as a first protocol message either $OutMsg_1$ or some $InMsg_1^{(2)}$ containing U. Let $InMsg_3^{(2)}$ be the third protocol message that $(0, k^{(2)})$ received.

••••••• Suppose that $InMsg_3^{(2)}$ was not generated by any responder instance. Since it was accepted by $(0, k^{(2)})$, according to the way the fourth protocol message delivery is handled (see above), the password $pw_{i'}$ has either been revealed or guessed by $\mathcal{M}^*$, and (i, j) has been exposed. $\mathcal{M}^*$ has already extracted master key $MK_{ij} \leftarrow H_{n^{(2)}}(V^u)$, and computed SK^*_{ij} and $OutMsg_5$ with it. In this case, $\mathcal{M}^*$ further extracts $MK_{i'j'} \leftarrow H_n(U^y)$, computes $\kappa_{i'j'} \leftarrow sf_{\ell_{CC}}(MK_{i'j'})$, and compares $InMsg_5$ to $\kappa_{i'j'}$. If they are equal, (i', j')'s session is started, (i', j') is exposed through $(0, k)$, and $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow pf_{\ell_{SK}}(MK_{i'j'})$ as (i', j')'s session key. *(Exposing is legal here because $pw_{i'}$ is known to $\mathcal{M}^*$. This is where key compromise impersonation is possible in practice for **Prot1**. The fundamental reason for this is that V could very well be a function of U and Y.)* If they are not equal, (i', j') rejects.

••••••• Suppose that $InMsg_3^{(2)}$ was generated by a responder instance. Then this instance is unique and is of the form $(i', j'^{(2)})$ with $PID_{i'j'^{(2)}} = ID_i$. The way fourth message deliveries are handled indicates that (i, j) has in this case participated in a correct exchange with $(i', j'^{(2)})$ through $(0, k^{(2)})$, and so was opened for connection from $(i', j'^{(2)})$ through $(0, k^{(2)})$. $\mathcal{RM}^*$ has selected SK^*_{ij} uniformly at random from $\{0, 1\}^{\ell_{SK}}$, $\mathcal{M}^*$ has selected $OutMsg_5$ uniformly at random from $\{0, 1\}^{\ell_{CC}}$, and $OutMsg_5$ was output to $\mathcal{M}$.

•••••••• Suppose $j'^{(2)} \neq j'$. Then one can see that we also have $k^{(2)} \neq k$. In this case, (i', j') simply rejects $InMsg_5$.

This works because assuming DDH, $\mathcal{M}$ still cannot compute $sf_{\ell_{CC}}\big(H_n(g^{uy})\big)$ from (U, Y, n).

•••••••• Suppose finally that $j'^{(2)} = j'$. Then we also have $k^{(2)} = k$, $Y^{(2)} = Y$, and $n^{(2)} = n$. Furthermore, $InMsg_4$ certifies the same data as $OutMsg_4$. In this case, $\mathcal{M}^*$ compares $InMsg_5$ and $OutMsg_5$. If they are equal, (i', j') has had a correct exchange with (i, j) through $(0, k)$, so it's session is started and it is connected to (i, j) through $(0, k)$ and $\mathcal{RM}^*$ sets $SK^*_{i'j'} \leftarrow SK^*_{ij}$. If $InMsg_5 \neq OutMsg_5$, (i', j') rejects.

The case of a server instance

• **Receiving the first protocol message and computing the second**

Let $InMsg_1$ be delivered to $(0, k)$ by $\mathcal{M}$.

•• Suppose that $InMsg_1$ was not generated by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_1$ to get plaintext w.

••• Suppose w is not of the form $(U, pw, OID_{0k}, CID_{0k})$ where U is a non-trivial group element and pw is a password. Then $InMsg_1$ is rejected.

••• Suppose w does have the correct format.

•••• Suppose $OID_{0k} = ID$ was not assigned to a user. Then ID has necessarily been input to a "set password" request, so $\mathcal{M}^*$ knows pw_{ID}. If $pw = pw_{ID}$, $(0, k)$ accepts the message and computes $OutMsg_2$ as specified by the protocol. Otherwise, $InMsg_1$ is rejected.

•••• Suppose $OID_{0k} = ID_i$ for some initialized i. If pw_i is already known to $\mathcal{M}^*$ *(i.e. if pw_i has been successfully tested as defined below or i was targeted by a "reveal password" operation)*, $\mathcal{M}^*$ compares pw and

pw_i and answers appropriately. If pw_i is not known, $\mathcal{M}^*$ makes a "test instance password" request on input $\big((0,k), i, pw\big)$. If $\mathcal{RM}^*$ answers positively, $\mathcal{M}^*$ accepts the message and computes $OutMsg_2$. It also learns pw_i If $\mathcal{RM}^*$ answers negatively, $InMsg_1$ is rejected.

This construction is justified by the fact that encryption is non-malleable, the consequence being that $\mathcal{M}$ cannot feasibly manipulate an encryption containing a correct password with the goal of just changing, say, a group element or an identity. $\mathcal{M}$'s only real option is to encrypt a password itself, making it reasonable to interpret a non-honest message as a password guess.

•• Suppose that $InMsg_1$ was generated by a user instance. Then this instance is unique by the size of Enc's randomness space.

••• If $OID_{0k} = ID_i$ for some initialized i and the unique instance that computed $InMsg_1$ is an originator instance (i,j) with $PID_{ij} = CID_{0k}$ then $(0,k)$ accepts the message without decrypting. Let X be the group element chosen for (i,j). $(0,k)$ computes $OutMsg_2$ using X.

Since $\mathcal{M}^*$ has a global view of the network it is running, it can check without decrypting whether or not the necessary and sufficient conditions for message acceptance are fulfilled. This is technically important to be able to work with IND-CCA-2 security, see section 8.5.

••• If any of the above-listed conditions are not met, $InMsg_1$ is rejected without decrypting.

• **Receiving the third protocol message and computing the fourth**

Let $InMsg_3$ be delivered to $(0,k)$. Let $InMsg_1$ be the first message received by $(0,k)$, let $OutMsg_2$ be the second message computed by $(0,k)$, and let U and n be the originator group element and hash index held by $(0,k)$.

•• Suppose that $InMsg_3$ was not generated by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_3$ to obtain plaintext w.

••• Suppose that w is not of the form $(U, V, n, pw, OID_{0k}, CID_{0k})$ where V is a non-trivial group element and pw is a password. Then $(0,k)$ rejects.

••• Suppose $InMsg_3$ is of correct format.

•••• If $CID_{0k} = ID$ was not assigned to a user, then $\mathcal{M}^*$ knows pw_{ID}. pw is compared to pw_{ID} to determine the correct response.

•••• Suppose $CID_{0k} = ID_{i'}$ for some initialized i'. If $\mathcal{M}^*$ knows $pw_{i'}$, it is simply compared to pw. If $\mathcal{M}^*$ does not yet know i', $\mathcal{M}^*$ makes a "test instance password" query on input $\big((0,k), i', pw\big)$ to find out from $\mathcal{RM}^*$ if $pw = pw_{i'}$, and computes the response accordingly.

Once again, it is because Enc is non-malleable that a message concocted by the adversary can be reasonably interpreted as a password guess.

•• Suppose now that $InMsg_3$ was computed by a user instance. Then by the size of Enc's randomness space, this instance is unique.

••• Suppose that $CID_{0k} = ID_{i'}$ for some initialized user i', that the unique user instance that computed $InMsg_3$ is a responder (i',j') with $PID_{i'j'} = OID_{0k}$, and that (i',j') received and accepted as a second protocol message some $InMsg_2$ containing (U,n). Then $(0,k)$ accepts $InMsg_3$ without decrypting it, and $\mathcal{M}^*$ computes $OutMsg_4$ using U, n, and Y, where Y is the group element chosen for (i',j'). $(0,k)$'s exchange is completed.

Similarly to the case of the first message delivery, $\mathcal{M}^*$ can check these conditions directly without needing to decrypt.

••• If any of the above conditions does not hold, $InMsg_3$ is rejected without decrypting.

This completes the proof sketch for theorem 1.

8.2 The Real World

Let $\mathcal{RM}$ be the real-world ring master, and $\mathcal{M}$ be a real-world adversary (a PPTA). The interaction between $\mathcal{M}$ and $\mathcal{RM}$ follows the rules defined in section 5.2. For completeness, we show how the game is initialized specifically for **Prot1**.

Let $\eta \in \mathbb{N}$ be the security parameter. The game begins with $\mathcal{RM}$ getting 1^η as input and passing it to $\mathcal{M}$ whose first action is to start the server.

- **Initialize server** $\mathcal{RM}$ sets up the parameters of **Prot1**, as a function of η. This involves constructing the group parameters (q, G, g), specifying the hash family $\{H_n\}_n$ where each H_n maps into $\{0,1\}^{\ell_{SK}+\ell_{CC}}$ for a long enough session key length ℓ_{SK} and long enough confirmation code length ℓ_{CC}, running $\mathcal{K}_E(1^\eta)$ and $\mathcal{K}_s(1^\eta)$ to respectively get (pk_E, sk_E) and (pk_S, sk_S), and finally specifying a password length ℓ_{pw}. The tuple of public parameters $\big(1^\eta, (q, G, g), \{H_n\}_n, pk_E, pk_S, \ell_{pw}\big)$ is then handed to $\mathcal{M}$.

$\mathcal{M}$ responds by generating a non-empty dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$, which is given back to $\mathcal{RM}$. $\mathcal{M}$ also provides a description of algorithms to efficiently sample elements of D uniformly at random and to efficiently tell if an element of $\{0,1\}^{\ell_{pw}}$ is in D or not.

Of course, from this point on $\mathcal{RM}$ runs user and server instances using the passwords sampled from D and the keying information $(pk_\mathcal{T}, sk_\mathcal{T}) := \big((pk_E, pk_S), (sk_E, sk_S)\big)$.

8.3 The First Hybrid World: Unique Random Choices

Let $\mathcal{RM}_1^h$ be the first hybrid world's ring master and let $\mathcal{M}$ be a PPTA. In the first hybrid world, $\mathcal{RM}_1^h$'s actions differ from those of $\mathcal{RM}$ as follows.

When $\mathcal{M}$ performs the "initialize server" operation, in addition to setting up the protocol parameters, $\mathcal{RM}_1^h$ initializes the sets $\mathcal{CE} \subset \mathbb{Z}_q^*$ of previously chosen exponents, $\mathcal{CHI} \subset \mathcal{K}_H$ of previously chosen hash indexes (where $\mathcal{K}_H$ is the index space for the hash family), and $\mathcal{CER} \subset \mathcal{R}_E$ of previously chosen encryption randomness, where $\mathcal{R}_E$ is the randomness space of *Enc*. All three sets start out empty.

Whenever $\mathcal{RM}_1^h$ has to select an exponent x from $\mathbb{Z}_q^*$ for a user instance, it samples x uniformly at random from $\mathbb{Z}_q^* - \mathcal{CE}$ and adds x to $\mathcal{CE}$. Whenever $\mathcal{RM}_1^h$ has to select a hash index n for a server instance, n is sampled uniformly at random from $\mathcal{K}_H - \mathcal{CHI}$ and then added to $\mathcal{CHI}$. Whenever $\mathcal{RM}_1^h$ has to select a randomness r from $\mathcal{R}_E$, it does so from $\mathcal{R}_E - \mathcal{CER}$ and adds r to $\mathcal{CER}$.

These changes, which we call the *Nonce Uniqueness Rules* (NUR), basically force all of the instances in the system to choose pairwise-distinct random values. This guarantees that any given instance is uniquely determined by the random choices it makes. More specifically, it allows uniquely associating honestly generated messages to the instances that computed them. Why this is also done with the encryption scheme's randomness will be made clear later. Since G, $\mathcal{K}_H$, and $\mathcal{R}_E$ have cardinalities that grow faster than any polynomial in η, intuitively we have the following result:

Proposition 1 *The transcript random variables $\mathcal{HW}_1(\mathcal{M})$ and $\mathcal{RW}(\mathcal{M})$ are statistically close.*

Proof: To ease notation, set $\phi := \mathcal{HW}_1(\mathcal{M})$ and $\psi := \mathcal{RW}(\mathcal{M})$. Let $\mathcal{T}$ be the set of all possible transcripts that ϕ or ψ can yield. We are to show that the expression

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$$

is negligible in η. We first do some work on $\mathcal{T}$. Specifically, we partition $\mathcal{T}$ into good sets to work with.

1). Let $\mathcal{T}_U$ ("U" for "*Unlucky*") be the set of elements in $\mathcal{T}$ yielded by interactions in which the ring master has at some point chosen the same exponent, or hash index, or encryption randomness at least twice.

2). Let $\mathcal{T}_O$ (the "O" is for "*Ordinary*") be the set of elements in $\mathcal{T}$ yielded by interactions in which the ring master has always chosen previously unused values.

We clearly have $\mathcal{T} = \mathcal{T}_U \cup \mathcal{T}_O$. It is also clear by construction that ϕ always takes its values in $\mathcal{T}_O$, while ψ may take its values in both $\mathcal{T}_U$ and $\mathcal{T}_O$.

Lemma 1 *$\mathcal{T}_U$ and $\mathcal{T}_O$ are disjoint.*

Proof: Let $t \in \mathcal{T}$. Suppose that it is both in $\mathcal{T}_U$ and $\mathcal{T}_O$. This means that there exists a configuration ci_O of the interaction between $\mathcal{RM}$ respecting the NUR or $\mathcal{RM}_1^h$ and $\mathcal{M}$ that yields transcript t, and there exists a configuration ci_U of the interaction between $\mathcal{RM}$ failing to respect the NUR and $\mathcal{M}$ that also yields t. Since ci_O and ci_U both output t, the public setup parameters - which are placed in t - worked with in both configurations are the same. In particular, both configurations use the same public encryption key pk_E. This namely implies that the decryption key sk_E is the same in both configurations as well.

Suppose first that t is in $\mathcal{T}_U$ as a result of $\mathcal{RM}$ having chosen the same hash key n for two distinct server instances in ci_U. Then n will have been immediately placed in t in the output messages of these two different server instances. But in ci_O this situation can only be a result of the ring master having chosen the same hash index n twice for two distinct server instances as well, which is a contradiction. We have proven that if $t \in \mathcal{T}_O$, t cannot be in $\mathcal{T}_U$ via a collision in hash index choices.

Suppose next that t is in $\mathcal{T}_U$ because in ci_U ring master $\mathcal{RM}$ chose the same exponent x for two different user instances. Then t contains, as the output messages of two different user instances, encryptions c and c' both of the form $\mathcal{E}_{pk_E}(..., g^x, ...)$. Since these strings are the result of computations in ci_O as well and in ci_O the decryption key being used is the same as that in ci_U, the ring master necessarily chose x twice as well in ci_O. This is again a contradiction. Thus, if $t \in \mathcal{T}_O$, t cannot be in $\mathcal{T}_U$ via a collision in exponent choices.

Suppose finally that t is in $\mathcal{T}_U$ because the ring master - in ci_U - chose the same encryption randomness r for two different user instances. In this case, t contains encryptions - produced by two different user instances - of the form $c = \mathcal{E}_{pk_E}(m; r)$ and $c' = \mathcal{E}_{pk_E}(m'; r)$. Then c and c' were produced also in ci_O, and since the decryption key is the same in ci_U and ci_O, c and c' decrypt in ci_O to the same messages m and m'. But in ci_O, the two randoms used are distinct, i.e. c is of the form $\mathcal{E}_{pk_E}(m; r_1)$ and c' is of the form $\mathcal{E}_{pk_E}(m'; r_2)$, where $r_1 \neq r_2$. By the injectivity of the encryption function in the randomness argument (see paragraph 7.3, this is still a contradiction: indeed, this implies $r = r_1 \neq r_2 = r$.

This concludes the (tedious) proof of the lemma. ■

We return to the proof of the proposition. The lemma has the immediate consequence that $\mathcal{T} = \mathcal{T}_O \sqcup \mathcal{T}_O$ and that ϕ only yields values in $\mathcal{T}_O$. We can write

$$\sum_{t \in \mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t \in \mathcal{T}_U} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$$

The expression $\sum_{t \in \mathcal{T}_U} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$ collapses to $\mathbb{P}[\psi \in \mathcal{T}_U]$ by the lemma and the fact that $\mathcal{RM}_1^h$ respects the NUR. Now, let $\mathcal{X}$ denote the random variable of exponent, hash index, and encryption randomness choices made by $\mathcal{RM}$ in the real-world interaction, let $\mathcal{Z}$ be the set of values $\mathcal{X}$ may reach, and let $\widetilde{\mathcal{Z}}$ be the subset of $\mathcal{Z}$ consisting of those choices which respect the NUR. By definition we have $\mathbb{P}[\psi \in \mathcal{T}_U] = \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$.

We now turn our attention to $\sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$. We have

$$\begin{aligned}\sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| &= \sum_{t \in \mathcal{T}_O} \Big|\mathbb{P}[\phi = t] - \big(\mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}] + \mathbb{P}[\psi = t | \mathcal{X} \notin \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]\big)\Big| \\ &= \sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}]\big|\end{aligned}$$

because conditioned on $\mathcal{X} \notin \widetilde{\mathcal{Z}}$, the lemma shows that ψ cannot be in $\mathcal{T}_O$. Next, we compare $\mathbb{P}[\phi = t]$ and $\mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]$. The adversary $\mathcal{M}$ is identical in both the real world and the hybrid world, and conditioned on the random instance choices respecting the NUR, these choices in the real world are identically distributed to those in the hybrid world. We conclude that $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]$. Combining this with our previous equation yields

$$\begin{aligned}\sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| &= \sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\phi = t]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}]\big| \\ &= \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}] \sum_{t \in \mathcal{T}_O} \mathbb{P}[\phi = t] \\ &= \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]\end{aligned}$$

so we basically end up with

$$\sum_{t \in \mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = 2\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$$

It remains to show that $\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$ is a negligible quantity.

Lemma 2 *$\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$ is negligible.*

Proof: We can first write

$$\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}] = \sum_{(q, \mathcal{K}_H, \mathcal{R}_E)} \mathbb{P}\big[\mathcal{X} \notin \widetilde{\mathcal{Z}} \big| (q, \mathcal{K}_H, \mathcal{R}_E)\big] \mathbb{P}\big[(q, \mathcal{K}_H, \mathcal{R}_E)\big]$$

where the event "$(q, \mathcal{K}_H, \mathcal{R}_E)$" is shorthand for the event that the group size is q, the hash key space is $\mathcal{K}_H$, and the encryption randomness space is $\mathcal{R}_E$. We fix a possible value for $(q, \mathcal{K}_H, \mathcal{R}_E)$ and focus our attention on $\mathbb{P}\big[\mathcal{X} \notin \widetilde{\mathcal{Z}} \big| (q, \mathcal{K}_H, \mathcal{R}_E)\big]$.

Let $CI_{(q, \mathcal{K}_H, \mathcal{R}_E)}$ be the set of configurations of the interaction between $\mathcal{RM}$ and $\mathcal{M}$ in which the setup parameters contain $(q, \mathcal{K}_H, \mathcal{R}_E)$, and let $\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)}$ denote the random variable of exponent, hash index, and encryption randomness choices made by $\mathcal{RM}$ in $CI_{(q, \mathcal{K}_H, \mathcal{R}_E)}$. We have by definition that

$$\mathbb{P}[\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)} \notin \widetilde{\mathcal{Z}}] = \mathbb{P}\big[\mathcal{X} \notin \widetilde{\mathcal{Z}} \big| (q, \mathcal{K}_H, \mathcal{R}_E)\big]$$

Let B be an upper bound on the running time of $\mathcal{M}$. The random variable $\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)}$ can be viewed as

$$\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)} = (\mathcal{X}_q, \mathcal{X}_{\mathcal{K}_H}, \mathcal{X}_{\mathcal{R}_E})$$

where $\mathcal{X}_q$ takes values in $\bigsqcup_{i=0}^{B} (\mathbb{Z}_q^*)^i$, $\mathcal{X}_{\mathcal{K}_H}$ takes values in $\bigsqcup_{i=0}^{B} \mathcal{K}_H^i$, and $\mathcal{X}_{\mathcal{R}_E}$ takes values in $\bigsqcup_{i=0}^{B} \mathcal{R}_E^i$. We see then that $\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)} \notin \widetilde{\mathcal{Z}}$ if and only if one of these three components has a collision in its coordinates. Let Col_q, $Col_{\mathcal{K}_H}$, and $Col_{\mathcal{R}_E}$ be the events that $\mathcal{X}_q$, $\mathcal{X}_{\mathcal{K}_H}$, and $\mathcal{X}_{\mathcal{R}_E}$ samples two identical coordinates, respectively. We can write

$$\mathbb{P}[\mathcal{X}_{(q, \mathcal{K}_H, \mathcal{R}_E)} \notin \widetilde{\mathcal{Z}}] \leq \mathbb{P}[Col_q] + \mathbb{P}[Col_{\mathcal{K}_H}] + \mathbb{P}[Col_{\mathcal{R}_E}]$$

Now, for any fixed value of η, the set of possible triples $(q, \mathcal{K}_H, \mathcal{R}_E)$ is finite. Let $\overline{q}$, $\overline{\mathcal{K}_H}$, and $\overline{\mathcal{R}_E}$ be such that $\overline{q} - 1$, $\#\overline{\mathcal{K}_H}$, and $\#\overline{\mathcal{R}_E}$ be minimal among all possible values. For large enough η, we have $B^2 < \overline{q} - 1$, $B^2 < \#\overline{\mathcal{K}_H}$, and $B^2 < \#\overline{\mathcal{R}_E}$. Therefore these inequalities are true for all possible triples $(q, \mathcal{K}_H, \mathcal{R}_E)$. Since we have $\mathbb{P}[Col_q] = \sum_{i=0}^{B} \mathbb{P}\big[Col_q \big| \mathcal{X}_q \in (\mathbb{Z}_q^*)^i\big] \mathbb{P}\big[\mathcal{X}_q \in (\mathbb{Z}_q^*)^i\big]$, and since the ring master samples exponents uniformly at random and independently of all other events in the interaction, we can use the birthday bound to conclude that $\mathbb{P}[Col_q] \leq \frac{B^2}{q-1} \leq \frac{N^2}{\overline{q}-1}$. Similarly, $\mathbb{P}[Col_{\mathcal{K}_H}] \leq \frac{B^2}{\#\mathcal{K}_H} \leq \frac{B^2}{\#\overline{\mathcal{K}_H}}$ and $\mathbb{P}[Col_{\mathcal{R}_E}] \leq \frac{B^2}{\#\mathcal{R}_E} \leq \frac{B^2}{\#\overline{\mathcal{R}_E}}$.

Thus, we finally get that

$$\begin{aligned}\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}] &= \sum_{(q,\mathcal{K}_H,\mathcal{R}_E)} \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}} | (q,\mathcal{K}_H,\mathcal{R}_E)]\mathbb{P}\big[(q,\mathcal{K}_H,\mathcal{R}_E)\big] \\ &\leq \sum_{(q,\mathcal{K}_H,\mathcal{R}_E)} \big(\mathbb{P}[Col_q] + \mathbb{P}[Col_{\mathcal{K}_H}] + \mathbb{P}[Col_{\mathcal{R}_E}]\big)\mathbb{P}\big[(q,\mathcal{K}_H,\mathcal{R}_E)\big] \\ &\leq B^2(\frac{1}{\overline{q}-1} + \frac{1}{\#\overline{\mathcal{K}_H}} + \frac{1}{\#\overline{\mathcal{R}_E}}) \sum_{(q,\mathcal{K}_H,\mathcal{R}_E)} \mathbb{P}\big[(q,\mathcal{K}_H,\mathcal{R}_E)\big] \\ &= B^2(\frac{1}{\overline{q}-1} + \frac{1}{\#\overline{\mathcal{K}_H}} + \frac{1}{\#\overline{\mathcal{R}_E}})\end{aligned}$$

This last expression being indeed negligible in η, we have the result. ■

8.4 The Second Hybrid World: Secure Signatures

In this next world, we tend to the signatures. Let $\mathcal{RM}_2^h$ be the second hybrid world's ring master and let $\mathcal{M}$ be an adversary. $\mathcal{RM}_2^h$ answers $\mathcal{M}$'s operation requests exactly as $\mathcal{RM}_1^h$ would except in how to process messages that are supposed to be signed by the server. The only changes that are made involve the "deliver user message" operation.

• **Delivering the second protocol message and computing the third** Let (i',j') be a responder instance expecting the second protocol message and let $InMsg_2$ be delivered to (i',j') by $\mathcal{M}$.

$\mathcal{RM}_2^h$ first checks to see if $InMsg_2$ is of the form (U,n,σ) for a group element $U \neq 1_G$, hash index n, and string σ. If not, $\mathcal{RM}_2^h$ has (i',j') reject the message and terminate. Otherwise, it computes the signature verification equation on input $(U,n,PID_{i'j'},ID_{i'},\sigma)$. If verification fails, (i',j') rejects (and terminates). If verification succeeds, $\mathcal{RM}_2^h$ examines past computations. If there exists a server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'},ID_{i'})$ that computed a second protocol message $OutMsg_2$ on input (U,n), $\mathcal{RM}_2^h$ has (i',j') accept the message, and pursue the protocol according to the specification. Otherwise, we shall say that $InMsg_2$ was *forged*; (i',j') rejects $InMsg_2$, even though the verification equation passed.

• **Delivering the fourth protocol message and computing the fifth** Let (i,j) be an originator instance waiting for the fourth protocol message and let $InMsg_4$ be delivered to (i,j). Let X be the group element chosen for (i,j).

$\mathcal{RM}_2^h$ checks to see if $InMsg_4$ is of the form (X,V,n,σ) where $V \neq 1_G$ and n are a group element and hash index, and σ is a string. If not, $InMsg_4$ is rejected and (i,j) is terminated. Otherwise, $\mathcal{RM}_2^h$ computes the signature verification equation on input $(X,V,n,ID_i,PID_{ij},\sigma)$. If this fails, the message is rejected. If it succeeds, $\mathcal{RM}_2^h$ examines previous actions. If there exists a server instance $(0,k)$ with $PIDS_{0k} = (ID_i,PID_{ij})$ that computed a fourth protocol message $OutMsg_4$ on input (X,V,n), (i,j) accepts $InMsg_4$ and the protocol continues. Otherwise, we shall say that $InMsg_4$ was *forged*; (i,j) rejects despite the verification equation having passed.

The effect of these changes is to force user instances to reject signatures that have been forged. Notice that in accordance with the precise definition of EU-ACMA security (see appendix B.2), we do not consider a string (that passes the verification equation) output by the adversary as being a forged signature if the adversary has already seen a valid signature on the same message. This is sufficient to provide security for our purpose because in the real world, all that is necessary is to prove that the signed data was indeed authenticated by the server. The malleability of the "signature around the message" does not affect this inference.

Since providing a forged signature is an event that occurs with negligible probability, we have:

Proposition 2 *The transcript random variables $\mathcal{HW}_2(\mathcal{M})$ and $\mathcal{HW}_1(\mathcal{M})$ are statistically close.*

Proof: Set $\phi := \mathcal{HW}_2(\mathcal{M})$ and $\psi := \mathcal{HW}_1(\mathcal{M})$, and let $\mathcal{T}$ be the set of values that can be reached be ϕ or ψ. We must show that

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$$

is a negligible quantity in η.

As in the proof of proposition 1, we begin be adequately partitioning $\mathcal{T}$ into subsets to work with.

1). Let $\mathcal{T}_O$ ("O" for "Ordinary") be the set of transcripts in which no forged signatures have been delivered to user instances that should accept them following protocol specification.

2). Let $\mathcal{T}_1$ be the set of transcripts in which at least one forged signature was delivered to a user instance, and this instance has accepted it.

3). Let $\mathcal{T}_2$ be the set of transcripts in which at least one forged signature was delivered to a user instance, the format check and verification equation have passed, and the instance has rejected.

It should be clear by definition that $\mathcal{T}_O$ is indeed disjoint from both $\mathcal{T}_1$ and $\mathcal{T}_2$. Also, a transcript yielded by ϕ cannot record the successful delivery of a forged signature. Similarly, a transcript yielded by ψ cannot record the forced rejection of a forged signature. Thus, $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint as well. These facts imply that we have the following partition:

$$\mathcal{T} = \mathcal{T}_O \sqcup \mathcal{T}_1 \sqcup \mathcal{T}_2$$

As a result, we can write

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \sum_{t\in\mathcal{T}_1} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \sum_{t\in\mathcal{T}_2} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$$

which can even be rewritten as follows:

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \mathbb{P}[\phi \in \mathcal{T}_2] + \mathbb{P}[\psi \in \mathcal{T}_1]$$

We now study the expressions $\sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$, $\mathbb{P}[\phi \in \mathcal{T}_2]$, and $\mathbb{P}[\psi \in \mathcal{T}_1]$.

First of all, for all $t \in \mathcal{T}_O$ we have $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t]$ because by construction the interaction between the adversary and ring master is identically distributed in both worlds provided no forged messages are delivered to user instances susceptible to computing a correct verification equation. Thus, obtain

$$\sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = 0$$

For the other two formulas, we have

Lemma 3 $\mathbb{P}[\phi \in \mathcal{T}_2] = \mathbb{P}[\psi \in \mathcal{T}_1] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1\Big]$ *where $\mathcal{B}$ is a PPTA trying to break the EU-ACMA-security (see appendix B.2) of the signature scheme. In particular, both $\mathbb{P}[\phi \in \mathcal{T}_2]$ and $\mathbb{P}[\psi \in \mathcal{T}_1]$ are negligible.*

Proof: "$\phi \in \mathcal{T}_2$" and "$\psi \in \mathcal{T}_1$" are events that occur if and only if the adversary - which is the same in both worlds - has at least one forged signature delivered to a user instance that will verify the correct signature equation. Up until this happens, both worlds are identically distributed. Thus, we indeed have $\mathbb{P}[\phi \in \mathcal{T}_2] = \mathbb{P}[\psi \in \mathcal{T}_1]$.

We now have to construct algorithm $\mathcal{B}$. It will basically play the role of the ring master while running the adversary $\mathcal{M}$ as a subroutine.

At the beginning of the game, on input 1^η, the challenger $\mathcal{CH}$ runs $\mathcal{K}_S(1^\eta)$ once to get (pk_S, sk_S), and gives pk_S to $\mathcal{B}$. $\mathcal{B}$ generates the remaining public parameters himself and hands them to $\mathcal{M}$, which returns a dictionary D back to $\mathcal{B}$. From this point on, $\mathcal{B}$ can play the role of the ring master in an interaction with

$\mathcal{M}$, respecting the NUR the same way $\mathcal{RM}_1^h$ or $\mathcal{RM}_2^h$ would. What we need to clarify is how $\mathcal{B}$ treats the generation and reception of signed messages, since $\mathcal{B}$ does not have control over the secret signing key.

- **Receiving the first protocol message and computing the second** Let $(0, k)$ be a server instance expecting to receive the first protocol message and suppose $\mathcal{M}$ has $InMsg_1$ delivered to $(0, k)$. $\mathcal{B}$ processes this message as $\mathcal{RM}_1^h$ or $\mathcal{RM}_2^h$ would. If $InMsg_1$ is accepted, let U be the originator group element contained in $InMsg_1$. $\mathcal{B}$ selects hash index n respecting the NUR, and makes a "signature" query to $\mathcal{CH}$ on input message $(U, n, OID_{0k}, CID_{0k})$. Upon receiving the response σ from $\mathcal{CH}$, $\mathcal{B}$ outputs $OutMsg_2 \leftarrow (U, n, \sigma)$ as $(0, k)$'s response to $\mathcal{M}$.

- **Receiving the second protocol message and computing the third** Let (i', j') be a responder instance expecting to receive the second protocol message and suppose that $\mathcal{M}$ has $InMsg_2$ delivered to (i', j'). $\mathcal{B}$ first checks whether $InMsg_2$ is of the form (U, n, σ) for some non-trivial group element U, some hash index n, and some string σ. If not, the message is rejected. Otherwise, $\mathcal{B}$ computes the signature verification equation on input $(U, n, PID_{i'j'}, ID_{i'}, \sigma)$ with public key pk_S. If verification fails, the message is rejected. If it succeeds, $\mathcal{B}$ examines its past challenger queries. If $\mathcal{B}$ has previously made a "signature" request on input $(U, n, PID_{i'j'}, ID_{i'})$, the message is accepted and the protocol continues. If such a query has not previously been made, $\mathcal{B}$ stops its interaction with $\mathcal{M}$, outputs $(U, n, PID_{i'j'}, ID_{i'}, \sigma)$ in a "forgery" query, and the game between $\mathcal{B}$ and $\mathcal{CH}$ ends with total output 1.

- **Receiving the third protocol message and computing the fourth** Let $(0, k)$ be a server instance expecting to receive the third protocol message, and suppose that $\mathcal{M}$ has $InMsg_3$ delivered to $(0, k)$. $\mathcal{B}$ processes this message following the same rules as the ring master of the first hybrid world. If the message is accepted, let U, n, and V be the originator group element, hash index, and responder group element contained in $InMsg_3$. $\mathcal{B}$ makes a "signature" query to $\mathcal{CH}$ on input $(U, V, n, OID_{0k}, CID_{0k})$. Upon receiving the answer σ, $\mathcal{B}$ outputs $OutMsg_4 \leftarrow (U, V, n, \sigma)$ to $\mathcal{M}$.

- **Receiving the fourth protocol message and computing the fifth** Let (i, j) be an originator instance expecting the fourth protocol message, and suppose that it receives $InMsg_4$ from the adversary. Let X be the group element chosen for (i, j). $\mathcal{B}$ first checks if $InMsg_4$ is of the form (X, V, n, σ) for some non-trivial group element V, hash index n, and bitstring σ. If not, the message is rejected. Otherwise, $\mathcal{B}$ computes the signature verification equation on input $(X, V, n, ID_i, PID_{ij})$ using public key pk_S. If verification fails, the message is rejected. If it succeeds, $\mathcal{B}$ examines past signature requests. If $(X, V, n, ID_i, PID_{ij})$ was already the input to a "signature" query, the message is accepted and the protocol continues. If not, $\mathcal{B}$ ends the interaction with the adversary, outputs $(X, V, n, ID_i, PID_{ij}, \sigma)$ in a "forgery" query, and the game with $\mathcal{CH}$ ends with total output 1.

This completes our description of $\mathcal{B}$. We now analyze it.

It is clearly a PPTA, since $\mathcal{M}$ is, and it is built to abide by the rules of the security game $\mathbb{G}^{EU-ACMA}$. Therefore, on one hand the security of the signature scheme implies that $\mathcal{B}$ outputs a forged signature only with negligible probability in η. On the other hand, $\mathcal{B}$ is constructed in such a way that it successfully outputs a forgery if and only if the interaction between $\mathcal{B}$ and the subroutine $\mathcal{M}$ reaches a configuration in which $\mathcal{M}$ has had a signed message that was never output by a server instance accepted by a user instance. Since up until this happens $\mathcal{B}$ runs in an identically distributed manner to the ring master in the first and second hybrid worlds, the probability that such a configuration is reached is equal to both $\mathbb{P}[\phi \in \mathcal{T}_1]$ and $\mathbb{P}[\psi \in \mathcal{T}_2]$. These quantities are thus negligible as well. The proposition is proved. ■

8.5 The Third Hybrid World: Using the Encryption

In the third hybrid world, we make use of the encryption scheme's security for two things: a). hiding the password in honestly generated encrypted messages and b). hiding certain group elements from the adversary's view until the relevant messages are delivered to appropriate server instances. The first of these two properties is necessary to guarantee offline dictionary attack resistance. The second is more subtle, and has consequences on the deliveries of the fourth and fifth protocol messages. We shall return to these points in in paragraphs 8.6 and 8.7.

The changes that are made to the ring master in this world involve the way honest user and server instances process the first and third protocol messages. The operations that are affected are the user and server message deliveries. The way the game is setup remains the same. Let $\mathcal{M}$ be an adversary.

• **Computing the first protocol message** Let (i, j) be an originator instance and suppose that $\mathcal{M}$ has (i, j) compute the first protocol message. $\mathcal{RM}_3^h$ selects x uniformly at random respecting NUR, computes $X \leftarrow g^x$, computes ciphertext $c \leftarrow \mathcal{E}_{pk_E}(1_G, 1^{\ell_{pw}}, ID_i, PID_{ij})$ respecting NUR, and outputs $OutMsg_1 \leftarrow c$ to $\mathcal{M}$.

• **Receiving the first protocol message and computing the second** Let $(0, k)$ be a server instance expecting to receive the first protocol message, and suppose that $\mathcal{M}$ has $InMsg_1$ delivered to $(0, k)$.

•• Suppose that $InMsg_1$ was not computed by a user instance. In this case, $\mathcal{RM}_3^h$ computes $(0, k)$'s response as in the previous hybrid world.

•• Suppose that $InMsg_1$ was computed by a user instance. By the NUR (specifically, the uniqueness of the encryption randomness), this instance is unique. (Recall that the encryption scheme is injective in the randomness argument over fixed messages.)

••• Suppose that $OID_{0k} = ID_i$ for some initialized user i, that the instance that computed $InMsg_1$ is an originator (i, j) with $PID_{ij} = CID_{0k}$, and that $InMsg_1$ is the first protocol message output by (i, j). In this case, $\mathcal{RM}_3^h$ has $(0, k)$ accept $InMsg_1$ without decrypting. Let X be the group element chosen for (i, j) by $\mathcal{RM}_3^h$. $\mathcal{RM}_3^h$ computes $OutMsg_2$ using X, and outputs $OutMsg_2$ to $\mathcal{M}$.

••• Suppose that the above conditions are not met. Then $\mathcal{RM}_3^h$ has $(0, k)$ reject the message without decrypting.

• **Receiving the second protocol message and computing the third** Let (i', j') be a responder instance waiting for the second protocol message and suppose that $\mathcal{M}$ has $InMsg_2$ delivered to (i', j'). $\mathcal{RM}_3^h$ accepts or rejects the message as in the previous hybrid world. Suppose that $InMsg_2$ is accepted. By the NUR and the rules on the treatment of signatures, there exists a unique server instance $(0, k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second protocol message $OutMsg_2$ containing the same tuple as $InMsg_2$. Let U be the group element in $InMsg_2$ and n be the hash index chosen for $(0, k)$. $\mathcal{RM}_3^h$ selects y uniformly at random, computes $Y \leftarrow g^y$, computes ciphertext $c' \leftarrow \mathcal{E}_{pk_E}(U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'})$, and outputs $OutMsg_3 \leftarrow c'$ to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth** Let $(0, k)$ be a server instance waiting for the third protocol message. Let $InMsg_3$ be delivered to $(0, k)$ by $\mathcal{M}$. Let $OutMsg_2$ be the second protocol message output by $(0, k)$.

•• Suppose that $InMsg_3$ was not computed by a user instance. Then $\mathcal{RM}_3^h$ processes $InMsg_3$ as in the previous hybrid world.

•• Suppose that $InMsg_3$ was computed by a user instance. By the NUR (specifically, the uniqueness of the encryption randomness used), this instance is unique.

••• Suppose that $CID_{0k} = ID_{i'}$ for some initialized user i', that the user instance that computed $InMsg_3$ is a responder (i', j') with $PID_{i'j'} = OID_{0k}$, and that (i', j') received as a second protocol message some message $InMsg_2$ containing the same tuple as $OutMsg_2$. Then $\mathcal{RM}_3^h$ has $(0, k)$ accept $InMsg_3$ without decrypting. Let Y be the group element that was chosen for (i', j'), let n be the hash index that was chosen for $(0, k)$, and let U be the group element in $OutMsg_2$. $\mathcal{RM}_3^h$ computes $(0, k)$'s response $OutMsg_4$ using (U, Y, n) and outputs $OutMsg_4$ to $\mathcal{M}_3^h$.

••• Suppose that the above conditions are not met. Then $(0, k)$ rejects $InMsg_3$ without decrypting.

Important Remark: The injectivity property of the encryption function (see paragraph 7.3) is important here for the ring master to keep track of which honest instance computed which encrypted message. Indeed, we can take the first protocol message as an example. In this hybrid world, i-originators do not compute encryptions of $(X, pw_i, ID_i, PID_{ij})$ anymore, with X a unique, local choice; they now ***all*** compute encryptions of $(1_G, 1^{\ell_{pw}}, ID_i, PID_{ij})$. If one such message is received by an appropriate server instance, the ring master needs to be able to tell unambiguously the origin of the message, i.e. which instance (among several with the same partner identity, of course) computed it. The only way to do this is to ensure that the messages are distinct, but by construction we cannot count on the plaintext to achieve this anymore. So, we use the encryption randomness instead. A similar remark holds for responders to whom identical messages may have been forwarded.

This completes our description of the way the third hybrid world works. We give an indication of what replacing the correct passwords with $1^{\ell_{pw}}$ gives us. The reason certain group elements are replaced by the neutral element will be explained later.

Intuitively, offline dictionary attacks are prevented because semantically secure encryption does not allow computationally bounded adversaries to compute any plaintext information. This is exploited in the changes above by replacing the real passwords with the dummy password $1^{\ell_{pw}}$ whenever a user generates a message. (This was the point of requiring that D not contain $1^{\ell_{pw}}$, although this is not a damaging restriction. In practice, a system can always enforce that at least one string of correct password length not be allowed as a password.) The technical description of semantic security basically states that the adversary cannot tell the difference, and so will behave essentially the same way.

Jumping ahead, aside from preventing offline dictionary attacks this change is essential for simulatability in two ways. First, as the reader may have already noticed in the proof sketch, we shall see when constructing the ideal-world adversary that it will have to behave like this ring master *without knowing the users' passwords.* Being able to encrypt a dummy password without the adversary noticing is the only way to do this. Secondly, this method allows singling out which messages are to be translated into "test password" operations.

Of course, using the IND-CCA-2 security of the encryption scheme, we have:

Proposition 3 *The transcript random variables $\mathcal{HW}_3(\mathcal{M})$ and $\mathcal{HW}_2(\mathcal{M})$ are computationally indistinguishable.*

Proof: As usual, let $\phi := \mathcal{HW}_3(\mathcal{M})$ and $\psi := \mathcal{HW}_2(\mathcal{M})$. Let $\mathcal{D}$ be a PPTA (in η, of course). We are to show that the quantity

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta, \phi) = 1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta, \psi) = 1\big]\Big|$$

is negligible in η. We do this by constructing a PPTA $\mathcal{B}$ that plays the games $\mathbb{G}_b^{ch-ad-IND-CCA-2}$ for $b \in \{0, 1\}$ with challenger $\mathcal{CH}$, against the encryption scheme Enc (see appendix B.1). More specifically, during the game it plays with $\mathcal{CH}$, $\mathcal{B}$ will first run $\mathcal{M}$ as a subroutine, simulating the ring master and recording the transcript. Once the interaction with $\mathcal{M}$ ends, $\mathcal{B}$ will run $\mathcal{D}$ on input the transcript obtained in the first stage, and answer whatever $\mathcal{D}$ answers.

We now detail the construction of $\mathcal{B}$. For $b \in \{0, 1\}$, at the beginning of the game $\mathbb{G}_b^{ch-ad-IND-CCA-2}$ between $\mathcal{B}$ and $\mathcal{CH}$, $\mathcal{B}$ receives as input the candidate public key encryption key pk_E computed by $\mathcal{CH}$ on input 1^η. $\mathcal{B}$ then sets up all of the other parameters for its interaction with $\mathcal{M}$ itself, gives all public parameters to $\mathcal{M}$, and receives dictionary D in return.

From here on, $\mathcal{B}$ runs the interaction exactly as $\mathcal{RM}_2^h$ or $\mathcal{RM}_3^h$ would, except in the way encrypted messages as produced and processed upon receipt, since $\mathcal{B}$ does not have control over the decryption key sk_E. The operations that need to be modified involve the user and server message deliveries.

- **Computing the first protocol message** Let (i, j) be an originator instance that $\mathcal{M}$ prompts to compute the first protocol message. $\mathcal{B}$ Selects an exponent x for (i, j) respecting the NUR, computes $X \leftarrow g^x$, and then makes a "challenge" query to $\mathcal{CH}$ on input the pair of messages

$$(1_G, 1^{\ell_{pw}}, ID_i, PID_{ij}) \text{ and } (X, pw_i, ID_i, PID_{ij})$$

Upon receipt of $\mathcal{CH}$'s answer c, $\mathcal{B}$ sets $OutMsg_1 \leftarrow c$, and outputs c to $\mathcal{M}$.

• **Receiving the first protocol message and computing the second** Let $(0,k)$ be a server instance expecting to receive the first protocol message and suppose that $\mathcal{M}$ has $InMsg_1$ delivered to $(0,k)$. There are several cases to consider.

•• Suppose first that $InMsg_1$ was not output by a user instance. Under this condition, $\mathcal{B}$ has never obtained $InMsg_1$ as the result of a "challenge" query, since this only occurs when $\mathcal{B}$ needs to encrypt messages for user instances. Therefore, $\mathcal{B}$ may make a "decryption" query to $\mathcal{CH}$ on input $InMsg_1$, and this is what it does. From this point on, $\mathcal{B}$ processes the obtained plaintext exactly as $\mathcal{RM}_2^h$ or $\mathcal{RM}_3^h$ would.

•• Suppose now that $InMsg_1$ was output by a user instance. Then this instance is unique.

••• Suppose that $OID_{0k} = ID_i$ for some initialized user i, that the unique instance that computed $InMsg_1$ is an originator (i,j) with $PID_{ij} = CID_{0k}$, and that $InMsg_1$ is indeed the first protocol message output by (i,j). In this case, $\mathcal{B}$ simply has $(0,k)$ accept $InMsg_1$. Let X be the group element that was chosen by $\mathcal{B}$ for (i,j) at the time $InMsg_1$ was computed. $\mathcal{B}$ computes $OutMsg_2$ using X, and a hash index n chosen for $(0,k)$ respecting the NUR.

••• Suppose that the above conditions are not met. Then $\mathcal{B}$ has $InMsg_1$ rejected.

• **Receiving the second protocol message and computing the third** Let (i',j') be a responder instance waiting for the second protocol message, and let $InMSg_2$ be delivered to (i',j'). $\mathcal{B}$ processes this signed message as it would be in the second hybrid world. If the message is accepted, $\mathcal{B}$ selects an exponent y respecting NUR, computes $Y \leftarrow g^y$, and makes a "challenge" query to $\mathcal{CH}$ on input the pair of messages

$$(U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}) \text{ and } (U, Y, n, pw_{i'}, PID_{i'j'}, ID_{i'})$$

where U and n are the originator group element and hash index in $InMsg_2$. When $\mathcal{CH}$ answers with encryption c, $\mathcal{B}$ sets $OutMsg_3 \leftarrow c$ and outputs c to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth** Let $(0,k)$ be a server instance waiting for the fourth protocol message, and let $InMsg_4$ be delivered to $(0,k)$. Let $OutMsg_2$ be the second protocol message output by $(0,k)$. $\mathcal{B}$ has several cases to consider.

•• Suppose that $InMSg_4$ was not generated by a user instance. In this case, $\mathcal{B}$ never obtained $InMSg_4$ as the response to a "challenge" query, so $\mathcal{B}$ may may a "decryption" query on input $InMsg_4$. It does so, and pursues the protocol as normal upon examining the obtained plaintext.

•• Suppose now that $InMsg_4$ was computed by a user instance. Then this instance is unique.

••• Suppose that $CID_{0k} = ID_{i'}$ for some initialized user i', that the unique instance that output $InMsg_4$ is a responder of the form (i',j') with $PID_{i'j'} = OID_{0k}$), and that (i',j') received as second protocol message some message $InMsg_2$ containing the same tuple as $OutMsg_2$. Then $\mathcal{B}$ has $(0,k)$ accept the message and computes $OutMsg_4$ using the originator group element U in $OutMsg_2$, the hash index n chosen for $(0,k)$, and the group element Y chosen for (i',j').

••• Suppose that the above conditions are not met. Then the message is simply rejected by $(0,k)$.

This completes the description of $\mathcal{B}$'s handling of encryption in its interactions with the adversary. Note first that the NUR are respected for the encryption randomness as well, since this is built into the way the challenger answers challenge queries in the challenge-adaptive IND-CCA-2 security game $\mathbb{G}_b^{ch-ad-IND-CCA-2}$, see appendix B.1.) Note also that in accordance with the rules of the IND-CCA-2 security game, $\mathcal{B}$ never asks to decrypt messages that have been requested as challenge encryptions. $\mathcal{B}$ can make decisions in these cases without having to decrypt.

It is clear that $\mathcal{B}$ is a PPTA; we now analyze $\mathcal{B}$'s advantage in winning the IND-CCA-2 security game.

We first need to relate the internal workings of the interaction between $\mathcal{B}$ and $\mathcal{M}$ with the second and third hybrid worlds. This depends on the challenge bit b the challenger $\mathcal{CH}$ is working with at the beginning of the game.

Suppose that $b = 0$. In this case, whenever $\mathcal{B}$ submits a pair of messages to obtain a new challenge ciphertext, the message that is encrypted by $\mathcal{CH}$ is the one in which the new group element and user password are respectively replaced by G's neutral element 1_G and the dummy password $1^{\ell pw}$. Since $\mathcal{B}$ is built to treat encrypted messages exactly as $\mathcal{RM}_3^h$ does, we can conclude that the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to the one between $\mathcal{RM}_3^h$ and $\mathcal{M}$.

Suppose now that $b = 1$. Here, the message that is encrypted by the challenger contains the real chosen group element and password, as is done by $\mathcal{RM}_2^h$ in the second hybrid world. Now we need to make sure that $\mathcal{B}$ responds to encrypted messages the same way $\mathcal{RM}_2^h$ does. This is clearly the case if the message received by the server is not user-generated. If the message is user-generated, it is as well because the conditions $\mathcal{B}$ checks for are exactly those that can be verified by examining the contents of the plaintext. Hence, if $b = 1$ the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to the on between $\mathcal{RM}_2^h$ and $\mathcal{M}$.

In particular, if $b = 0$ the obtained transcript is sampled according to ϕ and if $b = 1$ it is sampled according to ψ. Since by construction $\mathcal{B}$ outputs 1 if and only if $\mathcal{D}$ outputs 1, we can therefore conclude that

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta, \phi) = 1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta, \psi) = 1\big]\Big| = \\ \Big|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big]\Big|$$

The latter being negligible in η by the security of Enc, the proposition is proved. ■

8.6 The Fourth Hybrid World: Originator Group Elements the Adversary Cannot Guess and Statistically Secure Confirmation Codes

A Preliminary Discussion In this hybrid world and in the next, we will be dealing with certain cases of the adversary delivering the fifth protocol message: the confirmation code. In this hybrid world, we will also deal with a special case of the adversary delivering the fourth protocol message, but let us focus first on making some interesting comments about the code.

This benign-looking object is actually quite subtle to handle. In studying the security of **Prot1**, we noticed that depending on the configuration in which a responder instance, say (i', j'), finds itself, the confirmation code it expects to receive is, in the eyes of the adversary, of varying nature. Before continuing the proof, let us elaborate a little on this.

The confirmation code is an object that can only be computed given the knowledge of local random choices. If this were not the case, the protocol would be insecure. If the adversary finds itself in a position where it cannot a priori compute a confirmation code, it is natural to require that the adversary could only output this code with negligible probability. The corresponding change that is made to get from one hybrid to the next in such a case is then to simply make responders that receive candidate codes from the adversary reject them systematically. Only with negligible probability would a responder then reject a valid code illegally output by the adversary, thus assuring the statistical closeness of transcripts.

The issue is now identifying those configurations where the adversary should not be able to compute the code. In **Prot1**, they can be classified into two categories: the ones where the confirmation code is *statistically* secure and the ones where it is *computationally* secure. The former category relies solely on the entropy smoothing theorem while the latter relies on the DDH assumption.

This dichotomy was a big surprise to us at first, but in hindsight it does have an intuitive explanation. First, we explain our surprise, then we provide the explanation.

Surprise! **Prot1** is similar to Shoup's **DHKE-1** protocol in [35], and the proof of security of **DHKE-1** also has to argue the systematic rejection of certain confirmation codes. In turns out however that in **DHKE-1**, these codes are all computationally secure: they can only be computed if the adversary can compute the Diffie-Hellman function. Given the similarity of our protocol and **DHKE-1**, we simply expected the situation would be the same.

We were wrong.

Two Examples with One Major Difference Let us now explain what the difference is through two examples.

In our first example, consider the case where the first three protocol messages are computed and delivered correctly between (i,j), $(0,k)$, and (i',j'), with the usual notations. Since the third protocol message was delivered, a fourth was computed. Suppose that this fourth message is not yet delivered to (i,j), but that adversary $\mathcal{M}$ sends some string $\kappa \in \{0,1\}^{\ell_{CC}}$ to (i',j') anyway. κ must absolutely be rejected in this case, otherwise (i',j') will compute the key thinking (i,j) has as well, which is not true. Now, the adversary has at its disposal (g,X,Y,n) to try to compute the ℓ_{CC}-long suffix of $H_n(g^{xy})$. Clearly, it can do so if it can solve the computational Diffie-Hellman problem on input (g,X,Y). Fortunately, it cannot assuming DDH. Thus, in this case the code is computationally secure.

We turn now to our second example. Suppose that a responder (i',j') is partnered to a statically corrupted user - i.e. controlled directly by adversary $\mathcal{M}$ - and is communicating with this user through $(0,k)$. Suppose that $\mathcal{M}$ has the first and second protocol messages delivered correctly, and puts the third on hold. At this point, $\mathcal{M}$ has chosen some originator group element U that is not G's neutral element, $\mathcal{M}$ has observed the hash index n chosen by the server, and (i',j') has chosen an exponent $y \in \mathbb{Z}_q^*$ uniformly at random and computed the third protocol message containing $Y = g^y$. According to the rules of our ideal world, if $\mathcal{M}$ delivers some string κ to (i',j') now, this message must be rejected, because we require that even when exposing a responder, the server must have completed its exchange. But there does not seem to be any way to justify this, since $\mathcal{M}$ has complete control over U. What saves us is the fact that $\mathcal{M}$ *has not had a chance to observe Y yet, because the third message is encrypted and has not been delivered to the server.* In other words, $\mathcal{M}$ needs to compute the ℓ_{CC}-long suffix of $H_n(U^y)$ without any knowledge of y to hope to get (i',j') to accept. But since $U \neq 1_G$ and y is uniform, the entropy smoothing theorem tells us that $H_n(U^y)$ is statistically close to uniform, despite U having been computed by $\mathcal{M}$. Thus, in this case the code is statistically secure, modulo of course the security of the encryption scheme.

To sum up, the major difference between these two examples lies in whether or not the third message was decrypted, because that event gives the adversary more knowledge. Returning to our earlier mention of **DHKE-1**, all of the secure codes there are so computationally for the very simple reason that there is no encryption (it would be useless in that setting): the responder's group element is available to the adversary from the get-go.

Before returning to the proof, we explain what this has to do with the fourth protocol message. The special case of the fourth message delivery we deal with is in fact very simple, and natural: unless the first protocol message computed by an originator was actually decrypted by an appropriate server instance, any fourth protocol message delivered to this originator must be systematically rejected. The reasoning behind this is exactly the same as that justifying the rejection of statistically secure codes: if the first protocol message was never delivered, the originator group element was never made available to the adversary, and guessing this element to have a correct fourth message delivered to the originator is hopeless.

Treating with a single hybrid world both the rejection of statistically secure confirmation codes and the rejection of originator group elements the adversary cannot guess is actually a). the most convenient to pursue the proof and b). logical, since the arguments of both rejections are essentially of identical nature.

Returning to the Proof Itself Transforming the previous hybrid world into this one involves simply making changes to the treatment of the fourth and fifth message deliveries. Let $\mathcal{M}$ be an adversary and $\mathcal{RM}_4^h$ be the ring master.

- **Receiving the fourth protocol message and computing the fifth** Let (i,j) be an originator instance expecting to receive the fourth protocol message and suppose that $\mathcal{M}$ has $InMsg_4$ delivered to (i,j). Let $OutMsg_1$ be the first protocol message (i,j) computed.

Suppose that $OutMsg_1$ was never delivered as a first protocol message to a server instance $(0,k)$ with partner identities $PIDS_{0k} = (ID_i, PID_{ij})$. We shall say that (i,j) **is a statistically lonely originator.**

In this case, $\mathcal{RM}_4^h$ simply has (i,j) reject $InMsg_4$ and terminate.

If (i,j) is not a statistically lonely originator, $\mathcal{RM}_4^h$ computes (i,j)'s response as in the previous hybrid world.

• **Receiving the fifth protocol message** Let (i', j') be a responder instance expecting to receive the fifth protocol message and suppose that $\mathcal{M}$ has $InMsg_5$ delivered to (i', j'). Let $InMsg_2$ be the second protocol message (i', j') has received and let $OutMsg_3$ be the third protocol message (i', j') has computed. Let U and n be the originator group element and hash index contained in $InMsg_2$. We know that there exists a unique server instance $(0, k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second protocol message $OutMsg_2$ containing U and n.

Suppose that $OutMsg_3$ was never delivered to $(0, k)$. We shall say that (i', j') **is a statistically lonely responder.**

In this case, $\mathcal{RM}_4^h$ simply has (i', j') reject $InMsg_5$ and terminate.

If (i', j') is not a statistically lonely responder, $\mathcal{RM}_4^h$ computes (i', j')'s response as in the previous hybrid world.

This completes our description of the fourth hybrid world. We now compare it to the previous hybrid world.

Proposition 4 *The transcript random variables $\mathcal{HW}_4(\mathcal{M})$ and $\mathcal{HW}_3(\mathcal{M})$ are statistically close.*

Proof: We set $\phi := \mathcal{HW}_4(\mathcal{M})$ and $\psi := \mathcal{HW}_3(\mathcal{M})$. Let $\mathcal{T}$ be the set of transcripts that either ϕ or ψ can evaluate to. We wish to show that

$$\sum_{t \in \mathcal{T}} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right|$$

is negligible in η.

As usual, we begin by partitioning $\mathcal{T}$ into adequate sets to work with.

1). Let $\mathcal{T}_3$ be the set of transcripts t such that in t there exists a statistically lonely originator that accepted a fourth protocol message or there exists a statistically lonely responder that accepted a fifth protocol message.

Clearly, such a transcript can only be yielded by ψ, since the rules of the fourth hybrid world do not apply to the third hybrid world.

2). Let $\mathcal{T}_4$ be the set of transcripts t verifying either of the following properties.

2.a). There exists in t a statistically lonely originator (i, j) that automatically rejected a fourth protocol message $InMsg_4$ that passed the format check and the signature verification equation, and such that $InMsg_4$ contains X as the originator group element, where X is (i, j)'s group element.

2.b). There exists in t a statistically lonely responder (i', j') that automatically rejected a fifth protocol message $InMsg_5$ equal to the confirmation code (i', j') expects.

It should also be clear that such a transcript can only be yielded by ϕ. Indeed, the rules imposed in the fourth hybrid world force these situations to happen.

3). Let $\mathcal{T}_O$ be all remaining possible transcripts. These can be described as those transcripts in which if a statistically lonely originator (i, j) rejected a fourth protocol message $InMsg_4$, $InMsg_4$ either did not contain as an originator group element (i, j)'s chosen element, or was of incorrect format, or failed the signature verification, and if a statistically lonely responder (i', j') rejected a fifth protocol message $InMsg_5$, then $InMsg_5$ was not equal to the expected confirmation code.

Of course, these transcripts can be yielded by both ϕ and ψ.

It is easy to see that $\mathcal{T}_O$, $\mathcal{T}_3$, and $\mathcal{T}_4$ indeed form a partition of $\mathcal{T}$. Therefore, we have

$$\sum_{t \in \mathcal{T}} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| = \sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| + \mathbb{P}[\psi \in \mathcal{T}_3] + \mathbb{P}[\phi \in \mathcal{T}_4]$$

We now study these three expressions.

We begin with $\sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right|$. We claim that for all $t \in \mathcal{T}_O$, it holds that $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t]$, and thus the total sum is 0. Indeed, let $t \in \mathcal{T}_O$. By definition this means that none of the events that define elements of the sets $\mathcal{T}_3$ or $\mathcal{T}_4$ has occurred during the interaction. Since these events are precisely those that differentiate the actions of the ring master in the third and fourth hybrid worlds, we conclude that the view of the adversary is identically distributed in both worlds up until the point they do occur.

We next deal with $\mathbb{P}[\psi \in \mathcal{T}_3]$ and $\mathbb{P}[\phi \in \mathcal{T}_4]$. We will show that we can construct a PPTA $\mathcal{B}$ that uses $\mathcal{M}$ as a subroutine, plays against a challenger $\mathcal{CH}$ in the $\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}$ game (see appendix A.3), and for which it holds that

$$\mathbb{P}[\psi \in \mathcal{T}_3] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_4]$$

The negligibility of the expression in the middle then yields the result.

We construct PPTA $\mathcal{B}$ by directly describing its internal workings as it plays game $\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}$ against challenger $\mathcal{CH}$ on input security parameter 1^η. $\mathcal{B}$ will be essentially simulating the ring master in an interaction with $\mathcal{M}$, using the queries it may submit to $\mathcal{CH}$ to get group elements and hash indexes for user instances.

At the beginning of $\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}$, on input 1^η, $\mathcal{CH}$ constructs the tuple of group parameters and hash functions $\big((q, G, g), \{H_n\}_n\big)$, and gives this tuple to $\mathcal{B}$. $\mathcal{B}$ then generates all of the other public parameters for **Prot1**, and hands them over to $\mathcal{M}$. $\mathcal{M}$, of course, returns to $\mathcal{B}$ a password dictionary D.

• **Generating the first protocol message** Let (i, j) be an originator that $\mathcal{M}$ asks to have compute the first protocol message. $\mathcal{B}$ computes its answer exactly as $\mathcal{RM}_4^h$ does, but instead of choosing an exponent for (i, j), it makes a "prepare exponent" query to its challenger $\mathcal{CH}$, and records the (updated) counter C_{ij}.

• **Receiving the first protocol message and computing the second** Let $(0, k)$ be a server expecting to receive the first protocol message and suppose $\mathcal{M}$ has $InMsg_1$ delivered to $(0, k)$. $\mathcal{B}$ accepts or rejects the message exactly as $\mathcal{RM}_4^h$ would.

Suppose $InMsg_1$ is accepted. $\mathcal{B}$ makes a "hash index" query to get n for $(0, k)$. If $InMsg_1$ was not generated by a user instance, $\mathcal{B}$ computes $OutMsg_2$ as $\mathcal{RM}_4^h$ would. If $InMsg_1$ was generated by a user instance, let (i, j) be this unique instance. $\mathcal{B}$ makes a "recover exponent" query on input C_{ij} to get z_{ij}, computes $Z_{ij} \leftarrow g^{z_{ij}}$, and computes $OutMsg_2$ with Z_{ij}.

• **Receiving the second protocol message and computing the third** Let (i', j') be a responder instance waiting for the second protocol message and suppose that $\mathcal{M}$ has $InMsg_2$ delivered to (i', j'). $\mathcal{B}$ has $InMsg_2$ accepted or rejected as $\mathcal{RM}_4^h$ would. If $InMsg_2$ is accepted, $\mathcal{B}$ makes a "prepare exponent" query and records the value of $C_{i'j'}$. It computes $OutMsg_3$ as $\mathcal{RM}_4^h$ would and outputs $OutMsg_3$ to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth** Let $(0, k)$ be a server instance waiting for the third protocol message. Suppose $InMsg_3$ is delivered to $(0, k)$. $\mathcal{B}$ has the message accepted or rejected as $\mathcal{RM}_4^h$ would.

Suppose $InMsg_3$ is accepted. If it was not generated by a user instance, the response is computed as $\mathcal{RM}_4^h$ would compute it. If $InMsg_3$ was generated by a user instance, let (i', j') be this unique responder. $\mathcal{B}$ makes a "recover exponent" query on input $C_{i'j'}$ to get $z_{i'j'}$, computes $Z_{i'j'} \leftarrow g^{z_{i'j'}}$, and computes $OutMsg_4$ using $Z_{i'j'}$. $OutMsg_4$ is output to $\mathcal{M}$.

• **Delivering the fourth protocol message and computing the fifth** Let (i, j) be an originator instance expecting to receive the fourth protocol message, and suppose that $InMsg_4$ is delivered to (i, j) by $\mathcal{M}$. $\mathcal{B}$ first checks to see if $InMsg_4$ is of the form (U, V, n, σ), where U and V are non-trivial group elements, n is a hash index, and σ is a string. If not, $\mathcal{B}$ has (i, j) reject the message. Otherwise, there are two cases to consider.

•• Suppose (i, j) is not a statistically lonely originator. In this case, by construction of $\mathcal{B}$, $\mathcal{B}$ has made a "recover exponent" request on input C_{ij} to get z_{ij}, and has computed $Z_{ij} \leftarrow g^{z_{ij}}$. $\mathcal{B}$ then treats $InMsg_4$ as $\mathcal{RM}_4^h$ would, using the group element Z_{ij} as (i, j)'s originator group element. If $InMsg_4$ is accepted, $\mathcal{B}$ computes $H_n(V^{z_{ij}})$, which is then parsed into SK_{ij} and $OutMsg_5$. $OutMsg_5$ is output to $\mathcal{M}$.

•• Suppose now that (i, j) is a statistically lonely originator. $\mathcal{B}$ has therefore not made a "recover exponent" request on input C_{ij}. In this case, $\mathcal{B}$ first verifies the signature. If the signature is invalid, $\mathcal{B}$ rejects the message. If the signature is valid, $\mathcal{B}$ makes a "guess group element" request on input (U, C_{ij}). If this guess fails, $\mathcal{B}$ has $InMsg_4$ rejected, (i,j) is terminated, and the game continues. If the guess succeeds, $\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}$ ends with total output 1.

• **Receiving the fifth protocol message** Let (i',j') be a responder instance waiting for the fifth protocol message, and suppose that $\mathcal{M}$ has $InMsg_5 \in \{0,1\}^{\ell_{CC}}$ delivered to (i',j'). Since (i',j') is expecting a fifth message, it already accepted a second one, $InMsg_2$. Let U and n be the originator group element and hash index present in $InMsg_2$. We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second protocol message $OutMsg_2$ on input (U,n), and that n was obtained through a "hash index" query.

•• Suppose that (i',j') is not a statistically lonely responder. Then by construction $\mathcal{B}$ has made a "recover exponent" query on input $C_{i'j'}$ to get $z_{i'j'}$ and computed $Z_{i'j'} \leftarrow g^{z_{i'j'}}$ as the responder group element. In this case, $\mathcal{B}$ computes $H_n(U^{z_{i'j'}})$, parses it, and compares the obtained confirmation code to $InMsg_5$. If they are equal, $InMsg_5$ is accepted, and otherwise it is rejected.

•• Suppose that (i',j') is a statistically lonely responder. Therefore, by construction of $\mathcal{B}$, no "recover exponent" query has been made on input $C_{i'j'}$ for the instance (i',j'). In this case, $\mathcal{B}$ makes a "guess hash suffix" query on input $(U, n, C_{i'j'}, InMsg_5)$, has (i',j') reject $InMsg_5$, and the game continues.

This completes the description of how $\mathcal{B}$ runs its interaction with $\mathcal{M}$. It should be clear that $\mathcal{B}$ is a PPTA. We now turn to relating $\mathcal{B}$'s success probability, i.e. the probability that the total output of $\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}$ is 1, to the probabilities $\mathbb{P}[\phi \in \mathcal{T}_3]$ and $\mathbb{P}[\psi \in \mathcal{T}_4]$.

By construction, the events "$\psi \in \mathcal{T}_3$" in the third world, "$\phi \in \mathcal{T}_4$" in the fourth world, and "$\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_{\ell_{CC}}-ES}(1^\eta)\big) = 1$" in the game occur if and only if at some point in the interaction between $\mathcal{M}$ and the ring master one of the following two events first occurs: a statistically lonely originator receives a fourth protocol message that passes the signature check and containing the correct (hidden) group element or a statistically lonely responder receives the expected (hidden) confirmation code. Since this combination of requirements is the same in all three interactions, we will have the result provided up until these conditions first occur, all three of these interactions are identically distributed. But this is also the case by construction of the two hybrid worlds and algorithm $\mathcal{B}$. This completes the proof. ■

Remark: The proof above shows why **Prot1** technically requires the originator's group element to be a part of the message returned to the originator. Indeed, one might wonder why this had to be the case, since in reality the originator should have its own group element in memory anyway. The problem is that in the above construction, $\mathcal{B}$ has to verify a signature first, and needs an candidate originator group element to do so. There does not seem to be any non-convoluted way around this.

8.7 The Fifth Hybrid World: Computationally Secure Confirmation Codes

As previously announced, this fifth hybrid world is used to get rid of confirmation codes that are computationally secure. These are essentially the codes that should be output by originators waiting to receive a fourth protocol message from the server during a correct exchange with an honest responder. Delivery of the first and third protocol messages - computed by the originator and responder, respectively - make both instances' group elements public, but this alone should not be sufficient for the adversary to compute the correct confirmation code.

Again, moving from the previous world to this one only involves changing one of user message deliveries, that of the fifth message.

• **Receiving the fifth protocol message, *revisited*** Let (i',j') be a responder expecting to receive the fifth protocol message, and suppose that $\mathcal{M}$ has $InMsg_5 \in \{0,1\}^{\ell_{CC}}$ delivered to (i',j'). Let $InMsg_2$ be the second protocol message received by (i',j') and let $OutMsg_3$ be the third protocol message computed by (i',j'). By the NUR and the security of the signatures, there exists a unique server instance $(0,k)$ that computed a second protocol message $OutMsg_2$ containing the same tuple as $InMsg_2$. Let $InMsg_1$ be the first protocol message received by $(0, k)$.

•• Suppose that $PID_{i'j'} = ID_i$ for some initialized user i, that $PIDS_{0k} = (ID_i, ID_{i'})$, and that there exists an originator (i, j) that computed $OutMsg_1$ as its first protocol message. Let X be the group element chosen for (i, j). Suppose that either $InMsg_1 = OutMsg_1$ or there exists a server instance $(0, k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ that received and accepted $OutMsg_1$ and $(0, k)$ received and accepted a first protocol message $InMsg_1$ containing the group element X. In particular, (i, j) is not a statistically lonely originator, and by the NUR, (i, j) is unique.

Suppose further that $(0, k)$ received $OutMsg_3$ as third protocol message and computed $OutMsg_4$. In particular, (i', j') is not a statistically lonely responder.

Suppose finally that one of the three following conditions is satisfied:

CS1). (i, j) has not received a fourth protocol message;

CS2). (i, j) has received and rejected a fourth protocol message;

CS3). (i, j) has received and accepted a fourth protocol message $InMsg_4$, there exist a server instance $(0, k^{(2)})$ with $k^{(2)} \neq k$ and a responder instance $(i', j'^{(2)})$ with $j'^{(2)} \neq j'$ such that $(0, k^{(2)})$ accepted a third protocol message computed by $(i', j'^{(2)})$, and the fourth protocol message $OutMsg_4^{(2)}$ computed by $(0, k^{(2)})$ contains the same tuple as $InMsg_4$.

We shall say that (i', j') **is a computationally lonely responder**.

In this case, $\mathcal{RM}_5^h$ has (i', j') reject $InMsg_5$ automatically.

•• Suppose that (i', j') is not a computationally lonely responder. Then $\mathcal{RM}_5^h$ computes (i', j')'s response as in the previous hybrid.

Remark: We point out in the conditions above that neither (i, j) nor (i', j') are statistically lonely. This makes complete sense in view of the computational assumption we wish to exploit in this world which amounts to the adversary being unable to compute a sufficiently long suffix of $H_n(g^{xy})$ given (n, g^x, g^y): indeed, it is precisely because (i, j) and (i', j') ***are not*** statistically lonely that g^x ***and*** g^y ***are made available to*** $\mathcal{M}$.

This completes the description of the fifth hybrid world. We turn to proving:

Proposition 5 *The transcript random variables $\mathcal{HW}_5(\mathcal{M})$ and $\mathcal{HW}_4(\mathcal{M})$ are statistically close.*

Proof: Set $\phi := \mathcal{HW}_5(\mathcal{M})$ and $\psi := \mathcal{HW}_4(\mathcal{M})$, and let $\mathcal{T}$ be the set of all possible transcripts either of these variables can evaluate to. We partition $\mathcal{T}$ as follows:

1). Let $\mathcal{T}_4$ be the set of transcripts in which there exists a responder (i', j') that accepted a fifth protocol message while computationally lonely. Clearly, elements in $\mathcal{T}_4$ can only be yielded by ψ.

2). Let $\mathcal{T}_5$ be the set of transcripts in which there exists a responder (i', j') that rejected a correct confirmation code while computationally lonely. Such transcripts can only be yielded by ϕ.

3). Let $\mathcal{T}_O$ be the set of remaining transcripts. These can be described as follows: if a computationally lonely responder rejected a confirmation code, this code was incorrect, and if a responder accepted a confirmation code, this responder was not computationally lonely upon receipt of this code.

By construction, it is clear that these three sets indeed form a partition of $\mathcal{T}$. We deduce from this that:

$$\sum_{t \in \mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \mathbb{P}[\psi \in \mathcal{T}_4] + \mathbb{P}[\phi \in \mathcal{T}_5]$$

We first assert that $\sum_{t \in \mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = 0$. Indeed, the adversary's view is identically distributed in both hybrid worlds unless one of the events defining $\mathcal{T}_4$ or $\mathcal{T}_5$ occurs. This implies that for all $t \in \mathcal{T}_O$, $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t]$.

Next, we show that

$$\mathbb{P}[\psi \in \mathcal{T}_4] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_5]$$

where $\mathcal{B}$ is a PPTA we exhibit below, playing game $\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}$ described in appendix A.2 against a challenger $\mathcal{CH}$. Of course, the negligibility of the middle term then yields what we want.

We describe $\mathcal{B}$ by explaining its actions in an interaction with $\mathcal{CH}$. At the beginning of $\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}$, on input 1^η, $\mathcal{CH}$ generates the group and hash parameters $\big((q, G, g), \{H_n\}_n\big)$, and gives these to $\mathcal{B}$. $\mathcal{B}$ then computes the remaining setup for the protocol, hands all of the public parameters to its subroutine $\mathcal{M}$, and receives from $\mathcal{M}$ the description of a dictionary space D. The interaction between $\mathcal{M}$, $\mathcal{B}$, and $\mathcal{CH}$ then proceeds as follows.

• **Computing the first protocol message** If $\mathcal{M}$ asks to have originator (i, j) compute the first protocol message, $\mathcal{B}$ responds by making a "left group element" query to receive (L, X) from $\mathcal{CH}$. It then computes $OutMsg_1$ as $\mathcal{RM}_4^h$ would, and outputs $OutMsg_1$ to $\mathcal{M}$.

• **Receiving the first protocol message and computing the second** Let $(0, k)$ be a server instance waiting for the first protocol message, and let $InMsg_1$ be delivered to $(0, k)$. $\mathcal{B}$ processes this message exactly as $\mathcal{RM}_4^h$ would. If the message is accepted, $\mathcal{B}$ makes a "hash index" query to get hash key n from $\mathcal{CH}$, and then computes $OutMsg_2$ using n and the originator group element obtained from $InMsg_1$.

• **Receiving the second protocol message and computing the third** Let (i', j') be a responder instance to whom $\mathcal{M}$ has a second protocol message $InMsg_2$ delivered. $\mathcal{B}$ processes the message as $\mathcal{RM}_4^h$; if the message is accepted, $\mathcal{B}$ makes a "right group element" query to obtain (R, Y), and computes $OutMsg_3$ as $\mathcal{RM}_4^h$ would. $OutMsg_3$ is output to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth** This operation is treated exactly like in the fourth hybrid world. (We do not detail this more as no queries need to be made by $\mathcal{B}$ at this stage.)

• **Receiving the fourth protocol message and computing the fifth** Let (i, j) be an originator waiting for the fourth protocol message, let (i, j) receive $InMsg_4$, and let X be the group element chosen for (i, j). $InMsg_4$ is processed by $\mathcal{B}$ exactly as it would be by $\mathcal{RM}_4^h$.

Suppose it is accepted. In particular, recall that this implies that (i, j) is not statistically lonely. Furthermore, there exists a unique server instance $(0, k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that received a first protocol message either equal to the one (i, j) output, or containing X, and that output a fourth protocol message $OutMsg_4$ containing the same tuple as $InMsg_4$. Let $OutMsg_2$ be the second message computed by $(0, k)$ and let $InMsg_3$ be the third protocol message received by $(0, k)$.

•• Suppose that $PID_{ij} = ID$ is not the identity of an initialized user. In this case, $InMsg_3$ was not generated by any user instance. Let V be the responder group element resulting from $InMsg_3$'s delivery. $\mathcal{B}$ makes a "left exponent" query on input (L, X) to get exponent x, computes $MK_{ij} \leftarrow H_n(V^x)$, parses it into $SK_{ij} \in \{0,1\}^{\ell_{SK}}$ and $\kappa_{ij} \in \{0,1\}^{\ell_{CC}}$, sets $OutMsg_5 \leftarrow \kappa_{ij}$, and outputs $OutMsg_5$ to $\mathcal{M}$.

•• Suppose now that $PID_{ij} = ID_{i'}$ for some initialized user i'. There are two subcases to consider.

••• Suppose that $InMsg_3$ was not computed by any user instance. Let V be the group element obtained from $InMsg_3$. Then $\mathcal{B}$ makes a "left exponent" query on input (L, X), and responds using x, as just above.

••• Suppose that $InMsg_3$ was computed by a user instance. Then this instance is unique, is a responder (i', j') with $PID_{i'j'} = ID_i$, and received a second protocol message $InMsg_2$ containing the same tuple as $OutMsg_2$. Let Y be the group element chosen for (i', j'). Y was obtained through a "right group element" query. $\mathcal{B}$ makes a "hashed Diffie-Hellman" query on input $\big((L, X), (R, Y), n\big)$, where n is the hash key obtained for $(0, k)$ through a "hash index" query. $\mathcal{CH}$ responds with $H_n(g^{xy})$ which $\mathcal{B}$ uses to compute SK_{ij} and $OutMsg_5$.

Important Remark: It is very important to have the "hashed Diffie-Hellman" query here, rather than just ask for exponent x to compute (i, j)'s master key, because we still cannot discount the possibility of (i, j)'s first protocol message being forwarded to different server instances, communicating in turn with different responders. Below, we shall see that we need to be able to perform "guess hash suffix" queries on those instances as well, and can only do so if x has not been queried.

• **Receiving the fifth protocol message** Finally, let (i',j') be a responder expecting to receive the confirmation code, and let $InMsg_5$ be delivered to (i',j'). Let $InMsg_2$ be the second protocol message received by (i',j'), $OutMsg_3$ be the third protocol message computed by (i',j'), and Y be the group element chosen for (i',j') through a "right group element" query. We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second protocol message $OutMsg_2$ on input the same tuple as $InMsg_2$. Let $InMsg_1$ be the first protocol message received by $(0,k)$ and n be the hash index chosen for $(0,k)$ using a "hash index" query.

If $OutMsg_3$ was not delivered to $(0,k)$, (i',j') is statistically lonely. Accordingly, $InMsg_5$ is automatically rejected. We therefore suppose that $OutMsg_3$ was delivered to $(0,k)$. Let $OutMsg_4$ be the fourth protocol message computed by $(0,k)$.

•• First, suppose that $PID_{i'j'} = ID$ is not the identity of an initialized user. In this case, $InMsg_1$ was not computed by a user instance. Let U be the originator group element obtained from $InMsg_1$. $\mathcal{B}$ makes a "right exponent" query on input (R,Y) to obtain y, computes $H_n(U^y)$, parses it into $SK \in \{0,1\}^{\ell_{SK}}$ and $\kappa_{i'j'} \in \{0,1\}^{\ell_{CC}}$, and compares $InMsg_5$ to $\kappa_{i'j'}$. If they are equal, $\mathcal{B}$ has (i',j') accept $InMsg_5$, and sets $SK_{i'j'} \leftarrow SK$ as (i',j')'s session key. Otherwise, $InMsg_5$ is rejected.

•• Next suppose that $PID_{i'j'} = ID_i$ for some initialized user i. There are several subcases to consider.

••• Suppose that the following condition ***is not met***: *there exists an originator instance (i,j) with $PID_{ij} = ID_{i'}$ that computed a first protocol message $OutMsg_1$, and **either** $InMsg_1 = OutMsg_1$ **or** $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$, and $InMsg_1$ contains the same originator group element as the second protocol message computed by* $(0,k^{(1)})$. In this case, $InMsg_1$ was not computed by a user instance; let U be the group element it contains. In this situation, $\mathcal{B}$ makes a "right exponent" query on input (R,Y) to get y, and pursues the protocol in the same way as directly above.

••• Suppose the condition just described ***is met***, i.e. *there exists an originator instance (i,j) with $PID_{ij} = ID_{i'}$ that computed a first protocol message $OutMsg_1$, and **either** $InMsg_1 = OutMsg_1$, **or** $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$, and $InMsg_1$ contains the same originator group element as the second protocol message computed by* $(0,k^{(1)})$. Then (i,j) is unique; let X be (i,j)'s group element. It was obtained through a "left group element" query.

•••• Suppose no fourth protocol message was delivered to (i,j). This is condition **CS1**. In this case, $\mathcal{B}$ has made neither a "left exponent query" on input (L,X), nor a "hashed Diffie-Hellman" query with the inputs (L,X) and (R,Y). He can thus make a "guess hash suffix" query on input $\big((L,X),(R,Y),n,InMsg_5\big)$, and then have (i',j') reject $InMsg_5$.

•••• Suppose now that some fourth protocol message $InMsg_4$ was delivered to (i,j).

••••• Suppose that $InMsg_4$ was rejected by (i,j). This is condition **CS2**. Similarly to the situation just above, $\mathcal{B}$ has made neither a "left exponent query" on input (L,X), nor a "hashed Diffie-Hellman" query involving (L,X) and (R,Y). Thus, a "guess hash suffix" query is made on input $\big((L,X),(R,Y),n,InMsg_5\big)$, and $InMsg_5$ begin rejected.

••••• Now suppose that $InMsg_4$ was accepted by (i,j); (i,j) has thus computed a session key SK_{ij} and output a fifth protocol message $OutMsg_5$. Then we know that there exists a unique server instance $(0,k^{(2)})$ with $PIDS_{0k^{(2)}} = (ID_i, ID_{i'})$ that computed some fourth protocol message $OutMsg_4^{(2)}$ containing the same data as $InMsg_4$. Furthermore, $(0,k^{(2)})$ necessarily received as a first protocol message either $OutMsg_1$ or another message containing X. Let $InMsg_3^{(2)}$ be the third protocol message received and accepted by $(0,k^{(2)})$ and let $n^{(2)}$ be the hash key obtained for $(0,k^{(2)})$.

•••••• Suppose that $InMsg_3^{(2)}$ was not computed by a user instance. Let V be the responder group element obtained from $InMsg_3^{(2)}$. According to the way fourth protocol messages are handled (see above), $\mathcal{B}$ has already made a "left exponent" query on input (L,X) to get x and has computed SK_{ij} and $OutMsg_5$ using $H_{n^{(2)}}(V^x)$. $\mathcal{B}$ now makes a "right exponent" query on input (R,Y) to get y, computes $H_n(X^y)$, parses it into $SK \in \{0,1\}^{\ell_{SK}}$ and $\kappa_{i'j'} \in \{0,1\}^{\ell_{CC}}$, and compares $\kappa_{i'j'}$ and $InMsg_5$. If they are equal, (i',j') accepts. Otherwise, it rejects.

●●●●●● Suppose now that $InMsg_3^{(2)}$ was computed by a user instance. Then this instance is necessarily a responder of the form $(i', j'^{(2)})$ with $PID_{i'j'^{(2)}} = ID_i$ and, by the NUR, is unique. Furthermore, $(i', j'^{(2)})$ has necessarily received as a second protocol message a message containing the same tuple as the second protocol message output by $(0, k^{(2)})$. Let $Y^{(2)}$ be the group element chosen for $(i', j'^{(2)})$. It was obtained using a "right group element" query.

●●●●●●● Suppose that $j'^{(2)} \neq j'$. Then, $OutMsg_3^{(2)} \neq OutMsg_3$. Since $OutMsg_3^{(2)}$ and $OutMsg_3$ were respectively accepted by $(0, k^{(2)})$ and $(0, k)$, we conclude that $k^{(2)} \neq k$ and $n^{(2)} \neq n$. We see then that condition **CS3** is satisfied. According to the treatment of fourth protocol message deliveries, $\mathcal{B}$ has made a "hashed Diffie-Hellman" query on input $\big((L, X), (R, Y^{(2)}), n^{(2)}\big)$ to obtain SK_{ij} and $OutMsg_5$. It has not made a "left exponent" query on (L, X), a "right exponent" query on (R, Y), or a "hashed Diffie-Hellman" query involving both (L, X) and (R, Y). Therefore, $\mathcal{B}$ may make, and makes, a "guess hash suffix" query on input $\big((L, X), (R, Y), n, InMsg_5\big)$ and has (i', j') reject $InMsg_5$.

●●●●●●● Suppose that $j'^{(2)} = j'$. Then, $OutMsg_3^{(2)} = OutMsg_3$, and so $k^{(2)} = k$. In this case, $\mathcal{B}$ has made a "hashed Diffie-Hellman" query on input $\big((L, X), (R, Y), n\big)$ to obtain SK_{ij} and $OutMsg_5$. Thus, $\mathcal{B}$ compares $InMsg_5$ and $OutMsg_5$. If they are equal, (i', j') accepts and $SK_{i'j'} \leftarrow SK_{ij}$. Otherwise, (i', j') rejects.

This completes the description of $\mathcal{B}$. It is clearly a PPTA assuming $\mathcal{M}$ is. We now analyze $\mathcal{B}$'s advantage in winning the $\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}$ game.

By construction, the events "$\mathcal{B}\big(\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}(1^\eta)\big) = 1$" in the game just described, "$\phi \in \mathcal{T}_5$" in the fifth hybrid world, and "$\psi \in \mathcal{T}_4$" in the fourth hybrid world all occur if and only if at one point in the interaction between $\mathcal{M}$ and the ring master a configuration is reached in which $\mathcal{M}$ delivers a correct confirmation code to a computationally lonely responder. Up until this happens, the adversary's view and the ring masters' actions are all identically distributed. Hence, we conclude that indeed

$$\mathbb{P}[\psi \in \mathcal{T}_4] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-sf_{\ell_{CC}}-CDHES}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_5]$$

which completes the proof. ■

8.8 The Sixth Hybrid World: Replacing Correctly Exchanged Keys and Confirmation Codes with Random Strings

The last step before we are able to funnel our hybrids into the ideal world is to identify those exchanges in which the session key and confirmation code can be replaced with random strings without the adversary being able to tell the difference, and to make this substitution. The following definitions basically state what a correct exchange is from the points of view of both an originator and a responder, taking into consideration the changes leading up to the fifth hybrid world. We state the definitions, then comment them, and then describe the changes made to the interaction in the sixth hybrid world.

In what follows, i and i' are initialized users, (i, j) is an i-originator with $PID_{ij} = ID_{i'}$, (i', j') is an i'-responder with $PID_{i'j'} = ID_i$, and $(0, k)$ is a server instance with $PIDS_{0k} = (ID_i, ID_{i'})$.

Definition 1 *We shall say that (i, j) **has participated in a correct exchange with** (i', j') **through** $(0, k)$ if:*

1). *(i, j) computed a first protocol message $OutMsg_1$. Let X be the unique group element chosen for (i, j);*

2). *$(0, k)$ received and accepted $OutMsg_1$ or there exists a server instance $(0, k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ that received and accepted $OutMsg_1$ and $(0, k)$ received and accepted a first protocol message $InMsg_1$ containing the group element X. Let $OutMsg_2$ be the message computed by $(0, k)$ and let n be the unique hash index chosen for $(0, k)$;*

3). *(i', j') received and accepted a second protocol message $InMsg_2$ containing the same group element X and hash index n as $OutMsg_2$, and computed $OutMsg_3$. Let Y be the unique group element chosen for (i', j');*

4). *$(0,k)$ received and accepted $OutMsg_3$, and computed $OutMsg_4$;*

5). *(i,j) received and accepted a fourth protocol message $InMsg_4$ containing (X,Y,n), and computed $OutMsg_5$.*

Definition 2 *We shall say that (i',j')* ***has participated in a correct exchange with*** *(i,j)* ***through*** *$(0,k)$ if:*

1). *(i,j) computed a first protocol message $OutMsg_1$. Let X be the unique group element chosen for (i,j);*

2). *$(0,k)$ received and accepted $OutMsg_1$ or there exists a server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ that received and accepted $OutMsg_1$ and $(0,k)$ received and accepted a first protocol message $InMsg_1$ containing the group element X. Let $OutMsg_2$ be the message computed by $(0,k)$ and let n be the unique hash index chosen for $(0,k)$;*

3). *(i',j') received and accepted a second protocol message $InMsg_2$ containing the same group element X and hash index n as $OutMsg_2$, and computed $OutMsg_3$. Let Y be the unique group element chosen for (i',j');*

4). *$(0,k)$ received and accepted $OutMsg_3$, and computed $OutMsg_4$;*

5). *(i,j) received and accepted a fourth protocol message $InMsg_4$ containing (X,Y,n), and computed $OutMsg_5$;*

6). *(i',j') received and accepted $OutMsg_5$.*

Comments on the Definitions The protocol is really just a Diffie-Hellman Key Exchange enhanced with a mildly elaborate authentication mechanism. Fundamentally, (i,j) and (i',j') will have participated in a correct and secure exchange if the g^x chosen by (i,j) was indeed received by (i',j'), the g^y chosen by (i',j') was indeed received by (i,j), and both (i,j) and (i',j') received the n chosen by $(0,k)$. This is all that needs to be guaranteed for the exchange to be secure, and it explains the difference between the odd-looking point **2).** and the seemingly natural point **4).** in both definitions.

Suppose $\mathcal{M}$ gets its hands on group element X (really just X, nothing about its logarithm to the base g, or other information) and password pw_i. $\mathcal{M}$ can very well decide to re-encrypt $(X, pw_i, ID_i, ID_{i'})$ and have that forwarded to the server. From the point of view of the protocol run's honest participants, this event makes no difference whatsoever as far as security goes: $\mathcal{M}$ has the password, but did not use it to replace or otherwise alter (i,j)'s random choices. Thus, the DDH assumption can still be applied. The same argument can be made regarding responder (i',j'), group element Y, and password $pw_{i'}$.

The question now is how these situations can occur; $\mathcal{M}$ needs a concrete way to get X or Y for these bizarre events to happen. As it turns out, the fact that X and Y are encrypted in the first messages they appear in makes quite a bit of difference. To actually get group element X, $\mathcal{M}$ can in theory simply forward the first protocol message to some other server instance: this would be $(0,k^{(1)})$ in the definitions. In fact, by the security of the encryption, this is the ***only feasible way***. The case of the responder's group element Y is a different story however, because the third protocol message also contains the server instance's chosen hash index n, which doubles as a nonce. Therefore, the only way for $\mathcal{M}$ to get Y is to forward it to ***that single server instance***. But this wastes the only server instance that could still be involved in a correct exchange between (i,j) and (i',j'), so a correct exchange is no longer possible.

As for the conditions on the second and fourth protocol messages, specifically the fact that the received messages may be different from the sent ones, even if they certify the same data, they are due to the definition of a secure signature, see appendix B.2. As already stated in section 8.4, changing the signature "around" the message without altering said message does not do any harm in this case.

These considerations serve once again as a good example of the fact that the "matching conversation" criteria of [5], while quite natural intuitively, is actually far too rigid to be applicable in all cases.

Back to the Hybrid at Hand To correctly specify the changes made in this world, we will need to following lemma to justify the consistency of responders' session key assignments. Let (i,j), (i',j'), and $(0,k)$ be as in the definitions.

Lemma 4 *Let $(i', j'^{(2)})$ be another responder instance with $PID_{i'j'^{(2)}} = ID_i$ and $(0, k^{(2)})$ be another server instance with $PIDS_{0k^{(2)}} = (ID_i, ID_{i'})$. If (i, j) has participated in a correct exchange with (i', j') through $(0, k)$ and with $(i', j'^{(2)})$ through $(0, k^{(2)})$, then $j'^{(2)} = j'$, $k^{(2)} = k$, and $In/OutMsg_a^{(2)} = In/OutMsg_a$ for all $a \in \mathbb{N}$.*

Let $(i, j^{(2)})$ be another originator instance with $PID_{ij^{(2)}} = ID_{i'}$, and let $(0, k^{(2)})$ be another server instance with $PIDS_{0k^{(2)}} = (ID_i, ID_{i'})$. If (i', j') has participated in a correct exchange with (i, j) through $(0, k)$ and with $(i, j^{(2)})$ through $(0, k^{(2)})$, then $j^{(2)} = j$, $k^{(2)} = k$, and $In/OutMsg_a^{(2)} = In/OutMsg_a$ for all a.

Proof: Suppose that (i, j) has participated in a correct exchange with (i', j') through $(0, k)$ and with $(i', j'^{(2)})$ through $(0, k^{(2)})$. Then $InMsg_4$, $OutMsg_4$, and $OutMsg_4^{(2)}$ contain the same (X, Y, n) tuple. By NUR, this implies $k^{(2)} = k$, and so $OutMsg_3 = OutMsg_3^{(2)}$. By NUR again, this implies that $j'^{(2)} = j'$.

Suppose that (i', j') has participated in a correct exchange with (i, j) through $(0, k)$ and with $(i, j^{(2)})$ through $(0, k^{(2)})$. Then (i, j) and $(i, j^{(2)})$ used the same group element and $(0, k)$ and $(0, k^{(2)})$ used the same nonce, so $j^{(2)} = j$ and $k^{(2)} = k$.

The message equalities follow easily. ■

Remark: It was already clear from the definitions that if (i', j') participated in a correct exchange with (i, j) through $(0, k)$, then (i, j) participated in a correct exchange with (i', j') through $(0, k)$. Some thought reveals that the expected message equalities across these two correct exchanges hold.

Now, we exhibit the changes made to define this world. They involve how to compute session keys in correct exchanges, which occurs when the last expected protocol message is delivered to either instance. Let $\mathcal{RM}_6^h$ be the ring master.

- **Delivering the fourth protocol message and computing the fifth** Let (i, j) be an originator instance with $PID_{ij} = ID_{i'}$ for some initialized user i'. Suppose that adversary $\mathcal{M}$ has a fourth protocol message $InMsg_4$ delivered to (i, j). Suppose that there exist a responder instance (i', j') with $PID_{i'j'} = ID_i$ and a server instance $(0, k)$ with $PIDS_{0k} = (ID_i, ID_{i'})$ such that, having been delivered $InMsg_4$, (i, j) has participated in a correct exchange with (i', j') through $(0, k)$.

  In this case, $\mathcal{RM}_6^h$ selects SK_{ij} uniformly at random from $\{0, 1\}^{\ell_{SK}}$, selects κ_{ij} uniformly at random from $\{0, 1\}^{\ell_{CC}}$, sets $OutMsg_5 \leftarrow \kappa_{ij}$, and outputs $OutMsg_5$ to $\mathcal{M}$.

- **Delivering the fifth protocol message** Let (i', j') be a responder instance with $PID_{i'j'} = ID_i$ for an initialized user i and suppose that $\mathcal{M}$ has just delivered to (i', j') a fifth protocol message $InMsg_5$. Suppose that there exist an originator (i, j) with $PID_{ij} = ID_{i'}$ and a server instance $(0, k)$ with $PIDS_{0k} = (ID_i, ID_{i'})$ such that, having been delivered $InMsg_5$, (i', j') has participated in a correct exchange with (i, j) through $(0, k)$. In this case, (i, j) has already participated in a correct exchange with (i', j') through $(0, k)$, and so $\mathcal{RM}_6^h$ has already selected SK_{ij} and $OutMsg_5 = InMsg_5$ uniformly at random. Since by lemma 4, (i, j) and $(0, k)$ are the only originator and server instance that could have participated in a correct exchange with (i', j'), it is well-defined to set $SK_{i'j'} \leftarrow SK_{ij}$.

  $\mathcal{RM}_6^h$ sets $SK_{i'j'} \leftarrow SK_{ij}$.

These are the only changes made to the interaction in this world. We shall now exploit the DDHES property (see appendix A.1) to prove

Proposition 6 *The transcript random variables $\mathcal{HW}_6(\mathcal{M})$ and $\mathcal{HW}_5(\mathcal{M})$ are computationally indistinguishable.*

proof: The proof of this proposition follows the same type of structure as that of proposition 3. Fixing a PPTA $\mathcal{D}$ that tries to distinguish $\phi := \mathcal{HW}_6(\mathcal{M})$ and $\psi := \mathcal{HW}_5(\mathcal{M})$, we construct a PPTA $\mathcal{B}$ that runs both $\mathcal{D}$ and $\mathcal{M}$ as subroutines, while playing the games $\mathbb{G}_b^{ad-DDHES}$ for $b \in \{0, 1\}$ against some challenger $\mathcal{CH}$. Essentially, $\mathcal{B}$ first simulates the ring master for $\mathcal{M}$, using the queries made available to it via $\mathbb{G}_b^{ad-DDHES}$ to generate the session keys and confirmation codes for correct exchanges, and builds the interaction transcript.

Secondly, this transcript is fed as input to $\mathcal{D}$, and the resulting bit is $\mathcal{B}$'s final output in the game. As usual, $\mathcal{B}$ is described through its interaction with $\mathcal{M}$, and the scheduling of its queries to $\mathcal{CH}$.

At the beginning of the game, on input 1^η, $\mathcal{CH}$ constructs $\big((q,G,g),\{H_n\}_n\big)$ and gives this data to $\mathcal{B}$. $\mathcal{B}$ then generates itself the remaining parameters for **Prot1**, and gives all of the public data to $\mathcal{M}$, who responds with a dictionary D. From this point on, the only operations that require a precise description are those concerning message deliveries.

• **Generating the first protocol message** If $\mathcal{M}$ has originator (i,j) compute the first protocol message, $\mathcal{B}$ responds by making a "left group element" query to get (L,X), and otherwise computes $OutMsg_1$ as $\mathcal{RM}_5^h$ would.

• **Receiving the first protocol message and computing the second** If $\mathcal{M}$ has $InMsg_1$ delivered to server instance $(0,k)$, $\mathcal{B}$ processes this message as $\mathcal{RM}_5^h$ would. If the message is accepted, $\mathcal{B}$ makes a "hash index" query to get n for $(0,k)$, and computes $OutMsg_2$ with n.

• **Receiving the second protocol message and computing the third** If (i',j') receives message $InMsg_2$, $\mathcal{B}$ accepts or rejects it as $\mathcal{RM}_5^h$ would. If the message is accepted, $\mathcal{B}$ makes a "right group element" query to get (R,Y), and computes $OutMsg_3$ as $\mathcal{RM}_5^h$ would.

• **Receiving the third protocol message and computing the fourth** $\mathcal{B}$ deals with this operation as $\mathcal{RM}_5^h$ would. No queries need be made to $\mathcal{CH}$.

• **Receiving the fourth protocol message and computing the fifth** Let (i,j) be an originator expecting the fourth protocol message and suppose that $InMsg_4$ is delivered to (i,j). $\mathcal{B}$ processes this message as $\mathcal{RM}_5^h$ would. Suppose it is accepted. Recall that in particular, this implies that (i,j) is not statistically lonely, and therefore its group element X - previously obtained by a "left group element" query - has been made available to $\mathcal{M}$. We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k}=(ID_i,PID_{ij})$ that computed a fourth protocol message $OutMsg_4$ containing X. Let n be the hash index obtained for $(0,k)$ via a "hash index" query. Let $InMsg_1$ be the first protocol message $(0,k)$ received, let $OutMsg_2$ be the second message computed by $(0,k)$, and let $InMsg_3$ be the third message $(0,k)$ received.

•• Suppose first that $PID_{ij}=ID$ is not the identity of an initialized user. In this case, $InMsg_3$ was not generated by a user instance; let V be the group element it yielded upon receipt by $(0,k)$. Since (i,j) is just now required to compute a master key, and X is uniquely associated to (i,j), no "challenge" query on input (L,X) has been made to $\mathcal{CH}$ yet. Thus, $\mathcal{B}$ may make a "left exponent" query on input (L,X) to get x. It then computes $MK_{ij}\leftarrow H_n(V^x)$, and further computes SK_{ij} and $OutMsg_5$ from MK_{ij} as usual.

•• Suppose now that $PID_{ij}=ID_{i'}$ is the identity of some initialized i'. Then there are several cases to consider.

••• If $InMsg_3$ was not generated by a user instance, $\mathcal{B}$ answers as just described above: letting V be the group element obtained from $InMsg_3$, $\mathcal{B}$ makes a "left exponent" query with (L,X) to get x, computes $MK_{ij}\leftarrow H_n(V^x)$, and pursues the protocol from there.

••• If $InMsg_3$ was generated by a user instance, we know that this instance is unique and is of the form (i',j') with $PID_{i'j'}=ID_i$. Since $InMsg_3$ was accepted, we also know that (i',j') received a second protocol message $InMsg_2$ containing the same tuple as $OutMsg_2$. Thus, (i,j) has participated in a correct exchange with (i',j') through $(0,k)$, and by lemma 4 (i',j') and $(0,k)$ are unique in verifying this property. Note that (i',j') is not statistically lonely, so Y has indeed been made available to $\mathcal{M}$. The group element Y associated to (i',j') was obtained using a "right group element" query.

By the rules governing the response to receipt of the fifth message (see below), no "right exponent" query has been made on input (R,Y). Also, the third protocol message $(0,k)$ received was honestly generated, so no "left exponent" query has been made on input (L,X). No challenge query has yet been made on input $\big((L,X),(R,Y)\big)$ with a hash index different from n, because $(0,k)$ is the only server instance (i,j) could go through to participate in a correct exchange with (i',j').

$\mathcal{B}$ can thus make a "challenge" query on input $\big((L,X),(R,Y),n\big)$ to $\mathcal{CH}$ to get MK_{ij}. Recall that if $b=0$, MK_{ij} is selected by $\mathcal{CH}$ uniformly at random from $\{0,1\}^{\ell_{SK}+\ell_{CC}}$ and if $b=1$, MK_{ij} is computed by $\mathcal{CH}$ as $H_n(g^{xy})$.

Upon receipt of MK_{ij}, $\mathcal{B}$ parses MK_{ij} into $SK_{ij} \in \{0,1\}^{\ell_{SK}}$ and $OutMsg_5 \in \{0,1\}^{\ell_{CC}}$ as usual.

• **Receiving the fifth protocol message** Let (i',j') be a responder expecting the confirmation code, and let $InMsg_5$ be delivered to (i',j'). Let $InMsg_2$ be the second protocol message received by (i',j') and $OutMsg_3$ be the third protocol message output by (i',j'). We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second message $OutMsg_2$ containing the same data as $InMsg_2$. Let n be the hash index, and $InMsg_1$ be the first message received by $(0,k)$.

If $OutMsg_3$ was not delivered to $(0,k)$, $InMsg_5$ gets automatically rejected. So we can suppose that $OutMsg_3$ has been delivered to $(0,k)$ (i.e. (i',j') is not statistically lonely). Thus, the group element Y, chosen for (i',j') via a "right group element" query, is available to $\mathcal{M}$ through the fourth protocol message $OutMsg_4$ output by $(0,k)$.

•• Suppose that $PID_{i'j'} = ID$ is not the identity of a user. In this case, $InMsg_1$ was not generated by a user; let U be the group element obtained upon receipt of $InMsg_1$ by $(0,k)$. Since (i',j') has not participated in a correct exchange with an honest originator, no "challenge" query has been made involving (R,Y). Thus, $\mathcal{B}$ can make a "right exponent" query on input (R,Y) to get y. $\mathcal{B}$ then computes $MK_{i'j'} \leftarrow H_n(U^y)$, parses it into SK and $\kappa_{i'j'}$, and compares $InMsg_5$ to $\kappa_{i'j'}$. It accepts or rejects accordingly.

•• Now suppose that $PID_{i'j'} = ID_i$ for a user i. Then there are further subcases to consider, based on the origins of $InMsg_1$.

••• Suppose that the following ***is not statisfied***: *there exists an originator* (i,j) *with* $PID_{ij} = ID_{i'}$ *that generated a first protocol message* $OutMsg_1$ *and either* $InMsg_1 = OutMsg_1$ *or* $OutMsg_1$ *was delivered to some server instance* $(0,k^{(1)})$ *with* $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ *and* $InMsg_1$ *contains the same originator group element as the second protocol message output by* $(0,k^{(1)})$. In this case, $InMsg_1$ was not generated by a user, and (i',j') has not participated in a correct exchange with an originator. Let U be the group element obtained from $InMsg_1$. No "challenge" query has been made involving (R,Y), so $\mathcal{B}$ makes a "right exponent" query with (R,Y) to get y, computes $MK_{i'j'} \leftarrow H_n(U^y)$, parses it into SK and $\kappa_{i'j'}$, and compares $InMsg_5$ to $\kappa_{i'j'}$.

••• Suppose that the above-mentioned condition ***is satisfied***, i.e. *there exists an originator* (i,j) *with* $PID_{ij} = ID_{i'}$ *that generated a first protocol message* $OutMsg_1$ *and either* $InMsg_1 = OutMsg_1$ *or* $OutMsg_1$ *was delivered to some server instance* $(0,k^{(1)})$ *with* $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ *and* $InMsg_1$ *contains the same originator group element as the second protocol message output by* $(0,k^{(1)})$. Then (i,j) is unique. Let X be its group element, obtained with a "left group element" query.

If (i,j) did not receive a fourth protocol message, or received and rejected one, we know that (i',j') is computationally lonely, so $InMsg_5$ is rejected by $\mathcal{B}$. We can therefore suppose that (i,j) received and accepted some fourth protocol message $InMsg_4$. Then we also know that a unique server instance $(0,k^{(2)})$ generated some fourth protocol message $OutMsg_4^{(2)}$ containing the same tuple as $InMsg_4$, and that $(0,k^{(2)})$ received as a first message either $OutMsg_1$ or some other first protocol message containing X. Let $InMsg_3^{(2)}$ be the third message received by $(0,k^{(2)})$ and $n^{(2)}$ be $(0,k^{(2)})$'s hash index.

•••• Suppose that $InMsg_3^{(2)}$ was not generated by a user instance. In this case, (i,j) has not participated in a correct exchange with any responder, and according to the above rules on fourth message deliveries, $\mathcal{B}$ has computed (i,j)' response by making a "left exponent" query with (L,X) to get x and computing $MK_{ij} \leftarrow H_{n^{(2)}}(V^x)$, with V being the group element obtained from $InMsg_3^{(2)}$.

No "challenge" query on input (R,Y) has been made yet because by lemma 4, (i,j) is the only originator (i',j') could have a correct exchange with.

$\mathcal{B}$ makes a "right exponent query" on input (R,Y) to get y, and computes $MK_{i'j'} \leftarrow H_n(X^y)$, parses it into SK and $\kappa_{i'j'}$, and compares $InMsg_5$ and $\kappa_{i'j'}$.

•••• Now suppose that $InMsg_3^{(2)}$ was computed by a user instance. Then this instance is of the form $(i',j'^{(2)})$

with $PID_{i'j'^{(2)}} = ID_i$, and (i,j) has participated in a correct exchange with $(i', j'^{(2)})$ through $(0, k^{(2)})$. Let $Y^{(2)}$ be $(i', j'^{(2)})$'s group element. $\mathcal{B}$ has made a "challenge" query on input $\big((L, X), (R, Y^{(2)}), n^{(2)}\big)$ to get MK_{ij} (which is either random, or equal to $H_{n^{(2)}}(g^{xy^{(2)}})$) and has computed SK_{ij} and $OutMsg_5$ with it. By lemma 4, we have either $j'^{(2)} \neq j'$ and $k^{(2)} \neq k$ or $j'^{(2)} = j'$ and $k^{(2)} = k$.

If $j'^{(2)} \neq j'$ and $k^{(2)} \neq k$, (i', j') is computationally lonely (the originator (i,j) has at this point participated in a correct exchange with a different i'-responder and through a different server instance), so $\mathcal{B}$ simply rejects $InMsg_5$.

If $j'^{(2)} = j'$ and $k^{(2)} = k$, $\mathcal{B}$ compares $InMsg_5$ and $OutMsg_5$. If they are equal, (i', j') has participated in a correct exchange with (i,j) through $(0,k)$. Thus, $\mathcal{B}$ can set $SK_{i'j'} \leftarrow SK_{ij}$, which is well-defined, again by lemma 4. If they are not, $InMsg_5$ is rejected.

This completes the description of $\mathcal{B}$'s interactions with $\mathcal{M}$. It is clear that $\mathcal{B}$ is a PPTA. It is also clear by construction that if $\mathcal{CH}$ is playing $\mathbb{G}_0^{ad-DDHES}$, then the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to that between $\mathcal{RM}_5^h$ and $\mathcal{M}$ in the fifth hybrid world, and if $\mathcal{CH}$ is playing $\mathbb{G}_1^{ad-DDHES}$, this interaction is identically distributed to that in between $\mathcal{RM}_6^h$ and $\mathcal{M}$ in the sixth world. Therefore, we indeed have:

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta, \phi) = 1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta, \psi) = 1\big]\Big| = \Big|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ad-DDHES}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ad-DDHES}(1^\eta)\big) = 1\Big]\Big|$$

and since the right-hand term is negligible by the DDHES property, this completes the proof of the proposition. ■

8.9 The Ideal World

We are now ready to construct an ideal-world adversary $\mathcal{M}^*$ running against the ideal-word ring master $\mathcal{RM}^*$ from an arbitrary, PPTA network adversary $\mathcal{M}$, thereby completing the proof of theorem 1. $\mathcal{M}^*$ will essentially play the role of a sixth-hybrid world ring master to $\mathcal{M}$, running as a subroutine. $\mathcal{M}^*$ will use the operations it has access to in its interaction with $\mathcal{RM}^*$ to

a). answer (some of) $\mathcal{M}$'s operation requests and

b). classify $\mathcal{M}$'s actions into corresponding ideal-world actions.

Point **b).** is especially important, for it is through this classification that we can identify the conditions for, and compute, connection assignments, and identify online password tests.

The analysis of the resulting $\mathcal{M}^*$ will then be straightforward. By construction, we will have that the ideal-world transcript $\mathcal{IW}(\mathcal{M}^*)$ and sixth-hybrid-world transcript $\mathcal{HW}_6(\mathcal{M})$ are in fact ***identically distributed***. It is to be able to seamlessly carry out this simulation that we slowly factored out of the real world all of the "bad" events that could occur - albeit with negligible probability - that have ***strictly no hope of being captured*** in the ideal world. For example, if $\mathcal{M}$ forges a signature and this leads to the compromise of a user instance, the ideal world does not cover this, for exposing can only happen under the condition that a corruption has occurred. Passing from the first to the second hybrid world allows us to formally exclude such an event with probability exactly 1.

The final argument will then of course be that the sixth-hybrid-world transcript $\mathcal{HW}_6(\mathcal{M})$ and the real-world transcript $\mathcal{RW}(\mathcal{M})$ are computationally indistinguishable. This is what the string of propositions yields, by the fact that statistical closeness implies computational indistinguishability and the transitivity of computational indistinguishability.

On to constructing $\mathcal{M}^*$.

As usual, $\mathcal{M}^*$ is described through its behavior in an interaction with $\mathcal{RM}^*$. We do this carefully, one operation at a time. Let 1^η be the security parameter at the beginning of the interaction. Using 1^η, $\mathcal{RM}^*$ specifies two integers: the session key bitlength ℓ_{SK} and the password bitlength ℓ_{pw}. The tuple $(1^\eta, \ell_{SK}, \ell_{pw})$ is then given to $\mathcal{M}^*$, who passes it on to $\mathcal{M}$.

Server initialization

On input $(1^\eta, \ell_{SK}, \ell_{pw})$, $\mathcal{M}$ first asks $\mathcal{M}^*$ to initialize the server. $\mathcal{M}^*$ thus runs the algorithms for the protocol's parameters and thus gets

$$\big((q,G,g), \{H_n\}_n, \ell_{SK}, \ell_{CC}, (pk_E, sk_E), (pk_S, sk_S)\big)$$

It passes the public parameters $\big((q,G,g), \{H_n\}_n, \ell_{SK}, \ell_{CC}, (pk_E, pk_S)\big)$ on to $\mathcal{M}$, who generates a dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$. D and its password sampling and recognizing algorithms are returned to $\mathcal{M}^*$.

Now $\mathcal{M}^*$ performs an "initialize server" operation to $\mathcal{RM}^*$, on input $(0, D)$. It also performs an "implementation" operation on input

$$\Big(server\ \ public\ \ key, \big((q,G,g), \{H_n\}_n, \ell_{SK}, \ell_{CC}, (pk_E, pk_S)\big)\Big)$$

User initialization

When $\mathcal{M}$ asks to have user i initialized on input ID_i, $\mathcal{M}^*$ makes the same request to $\mathcal{RM}^*$. Recall that $\mathcal{RM}^*$ samples a password pw_i from D for i outside of $\mathcal{M}^*$'s view. Therefore, while $\mathcal{M}^*$ has control over the server's strong secret keying information, it does not a priori know honest users' passwords.

Setting up statically corrupted users

When $\mathcal{M}$ makes a "Set Password" request on input (ID, pw) with $pw \in D$, $\mathcal{M}^*$ makes the same request to $\mathcal{RM}^*$.

Initializing user and server instances

When $\mathcal{M}$ makes "initialize user instance" or "initialize server instance" requests, these are simply forwarded by $\mathcal{M}^*$ to $\mathcal{RM}^*$.

Dynamically corrupting users

When $\mathcal{M}$ makes a "reveal user password" operation on input i, $\mathcal{M}^*$ will also simply forward the request to $\mathcal{RM}^*$. $\mathcal{RM}^*$'s response is password pw_i - which $\mathcal{M}^*$ thus learns - and forwards to $\mathcal{M}$.

Revealing session keys

When $\mathcal{M}$ makes a "reveal session key" query on input some user instance (i,j) or (i',j'), $\mathcal{M}^*$ simply makes the same request to $\mathcal{RM}^*$. $\mathcal{RM}^*$ responds with the value of SK^*_{ij}, and $\mathcal{M}^*$ sets $SK_{ij} \leftarrow SK^*_{ij}$, and gives SK_{ij} to $\mathcal{M}$.

Message deliveries to originator instances and connection assignments for the role *open*

Let (i,j) be an originator instance.

First of all, any message delivery $\mathcal{M}$ makes to (i,j) is recorded by $\mathcal{M}^*$ in an "implementation" operation made to $\mathcal{RM}^*$ on input

$$\big((i,j), InMsg, OutMsg, status_{ij}\big)$$

where of course $InMsg$ is the message delivered to (i,j), $OutMsg$ is the message output by (i,j), and $status_{ij}$ is equal to *accept*, *reject*, or *continue*. To indicate that (i,j) is to generate the first protocol message, $InMsg$ must be set to ε. If (i,j) does not generate a message, $OutMsg$ is set to ε. If (i,j) rejects the message, $\mathcal{M}^*$ makes a "terminate user instance" request to $\mathcal{RM}^*$ on input (i,j).

Next, suppose $\mathcal{M}$ has (i,j) generate the first protocol message. $\mathcal{M}^*$ computes (i,j)'s response as it is done by $\mathcal{RM}^h_6$ in the sixth hybrid world. Notice in particular that $\mathcal{M}^*$ does not need to know pw_i to do this, since it encrypts $1^{\ell_{pw}}$ instead.

Now, we examine $\mathcal{M}^*$'s response when $\mathcal{M}$ has a fourth protocol message $InMsg_4$ delivered to (i,j). $InMsg_4$ is accepted or rejected by $\mathcal{M}^*$ following the same rules as in the sixth hybrid world. Suppose $InMsg_4$ is accepted; we know that (i,j) is not statistically lonely, and we know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (ID_{ij}, PID_{ij})$ that computed a fourth protocol message $OutMsg_4$

containing the same tuple as $InMsg_4$. Let $OutMsg_2$ be the second message output by $(0,k)$, and $InMsg_3$ the third message received by $(0,k)$. Let $X = g^x$ be the group element $\mathcal{M}^*$ chose for (i,j), and n be the hash index $\mathcal{M}^*$ chose for $(0,k)$.

• If $PID_{ij} = ID$ is not the identity of an honest user, let V be the group element obtained from $InMsg_3$. $\mathcal{M}^*$ makes a "start session" request to $\mathcal{RM}^*$ on input (i,j), and (i,j) is exposed through $(0,k)$. $\mathcal{M}^*$ computes $MK_{ij} \leftarrow H_n(V^x)$, parses MK_{ij} into $SK_{ij} \in \{0,1\}^{\ell_{SK}}$ and $OutMsg_5 \in \{0,1\}^{\ell_{CC}}$, specifies SK_{ij} as (i,j)'s ideal-world session key SK^*_{ij} to $\mathcal{RM}^*$, and outputs $OutMsg_5$ to $\mathcal{M}$.

• Suppose now that $PID_{ij} = ID_{i'}$ for an initialized i'.

•• If $InMsg_3$ was not generated by a user instance, then by the rules governing the deliveries of message to server instances (see below), i''s password is known to $\mathcal{M}^*$. In this case as well, $\mathcal{M}^*$ starts (i,j)'s session, and (i,j) is exposed through $(0,k)$.

•• If $InMsg_3$ was generate by a user instance, we know that this instance is unique, is of the form (i',j') with $PID_{i'j'} = ID_i$, and (i,j) has participated in a correct exchange with (i',j') through $(0,k)$. In this case, (i,j)'s session is started and (i,j) is opened for connection from (i',j') through $(0,k)$. $\mathcal{RM}^*$ selects session key SK^*_{ij} uniformly at random from $\{0,1\}^{\ell_{SK}}$. $\mathcal{M}^*$ selects $OutMsg_5$ uniformly at random from $\{0,1\}^{\ell_{CC}}$, and outputs $OutMsg_5$ to $\mathcal{M}$.

Message deliveries to responder instances and connection assignments for the role *connect*

Let (i',j') be a responder instance.

First, the comments made regarding "implementation" and "termination" operations at the beginning of the paragraph on messages delivered to originators apply here as well.

Next, suppose that $\mathcal{M}$ has a second protocol message delivered to (i',j'). The message is processed by $\mathcal{M}^*$ as $\mathcal{RM}_6^h$ would process it; in particular, if it is accepted, the third protocol message is computed with $1^{\ell_{pw}}$ in place of $pw_{i'}$, which $\mathcal{M}^*$ does not a priori know.

Now we turn to treating the delivery of the fifth protocol message, the confirmation code. Let $InMsg_5$ be delivered to (i',j') by $\mathcal{M}$. Let $InMsg_2$ be the second protocol message received by (i',j'), let $Y = g^y$ be the group element chosen by $\mathcal{M}^*$ for (i',j'), and let $OutMsg_3$ be the third message output by (i',j'). We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a second protocol message $OutMsg_2$ containing the same tuple as $InMsg_2$. Let n be $(0,k)$'s hash index. Let $InMsg_1$ be the first message received by $(0,k)$.

Suppose first that $OutMsg_3$ was not delivered to $(0,k)$. In this case, (i',j') is statistically lonely, so $InMsg_5$ is rejected. This is in perfect accordance with the ideal-world rules because recall that we can only start (i',j')'s session if $(0,k)$'s exchange is completed. Since $OutMsg_3$ was not delivered to $(0,k)$, this is not the case (see server message deliveries below).

From now on, we suppose that $OutMsg_3$ was delivered to $(0,k)$, so $(0,k)$'s exchange is completed. Let $OutMsg_4$ be the fourth message output by $(0,k)$.

• Suppose $PID_{i'j'} = ID$ is not the identity of a user. Let U be the group element obtained from $InMsg_1$. In this case, $\mathcal{M}^*$ computes $MK_{i'j'} \leftarrow H_n(U^y)$ and parses it into SK and $\kappa_{i'j'}$. If $InMsg_5 = \kappa_{i'j'}$, (i',j')'s session is started and (i',j') is exposed through $(0,k)$. $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow SK$. If $InMsg_5 \neq \kappa_{i'j'}$, $InMsg_5$ is rejected.

• Suppose that $PID_{i'j'} = ID_i$ for some initialized i.

•• Suppose that the following ***is not*** true: there exists an originator (i,j) with $PID_{ij} = ID_{i'}$ that generated a first protocol message $OutMsg_1$ and either $InMsg_1 = OutMsg_1$ or $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ and $InMsg_1$ contains the same originator group element as the second protocol message output by $(0,k^{(1)})$. In this case, $InMsg_1$ was not generated by a user, (i',j') has not participated a in a correct exchange with any i-originator, and according to the treatment of server message deliveries (below) pw_i is known to $\mathcal{M}$. Letting U be the group element obtained from $InMsg_1$, $\mathcal{M}^*$ computes $MK_{i'j'} \leftarrow H_n(U^y)$ directly, parses it into SK and $\kappa_{i'j'}$, and either starts (i',j')'s session - exposing through $(0,k)$ - or rejects $InMsg_5$, based on whether $InMsg_5 = \kappa_{i'j'}$ or not.

•• Suppose that there exists an originator (i,j) with $PID_{ij} = ID_{i'}$ that generated a first protocol message $OutMsg_1$ and either $InMsg_1 = OutMsg_1$ or $OutMsg_1$ was delivered to some server instance $(0,k^{(1)})$ with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$ and $InMsg_1$ contains the same originator group element as the second protocol message output by $(0,k^{(1)})$. Then (i,j) is unique. Let $X = g^x$ be the group element chosen for (i,j).

If (i,j) either has not received a fourth message or has rejected one, we know that (i',j') is computationally lonely. Thus, $InMsg_5$ is rejected. This situation is clearly the only option we have according to the ideal-world rules because (i',j') has no i-originator to connect to yet and it cannot be exposed since nothing has forced $pw_{i'}$ or pw_i to be known by $\mathcal{M}^*$.

From now on, we suppose that (i,j) has received and accepted some fourth protocol message $InMsg_4$. Then there exists a unique server instance $(0,k^{(2)})$ that computed a fourth message $OutMsg_4^{(2)}$ containing the same tuple as $InMsg_4$, and that received group element X in its first protocol message. Let $InMsg_3^{(2)}$ be the third protocol message received by $(0,k^{(2)})$ and let $n^{(2)}$ be $(0,k^{(2)})$'s hash index.

••• If $InMsg_3^{(2)}$ was not computed by any user instance, then (i,j) has not participated in a correct exchange with any responder, and by the server message delivery rules below, $pw_{i'}$ is known to $\mathcal{M}^*$. Thus, (i,j) was exposed through $(0,k^{(2)})$. In this situation, $\mathcal{M}^*$ computes $MK_{i'j'} \leftarrow H_n(X^y)$ directly, parses it into SK and $\kappa_{i'j'}$, and either exposes (i',j') through $(0,k)$ or rejects $InMsg_5$, based on whether $InMsg_5 = \kappa_{i'j'}$ or not. (According to our model, this is the only scenario that can lead to KCI.)

••• If $InMsg_3^{(2)}$ was computed by a user instance, this instance is unique, is of the form $(i',j'^{(2)})$, and (i,j) has participated in a correct exchange with $(i',j'^{(2)})$ through $(0,k^{(2)})$. So at this point, (i,j) was opened for connection from $(i',j'^{(2)})$ through $(0,k^{(2)})$. $\mathcal{RM}^*$ selected a session key SK^*_{ij} uniformly at random for (i,j), and $\mathcal{M}^*$ selected $OutMsg_5$ uniformly at random as the fifth protocol message. Furthermore, we know that either $j'^{(2)} = j'$ and $k^{(2)} = k$ or $j'^{(2)} \neq j'$ and $k^{(2)} \neq k$.

If $j'^{(2)} = j'$ and $k^{(2)} = k$, $\mathcal{M}^*$ compares $OutMsg_5$ and $InMsg_5$. If they are equal, (i',j')'s session is started, and (i',j') is connected to (i,j) through $(0,k)$. Thus, $\mathcal{RM}^*$ sets $SK^*_{i'j'} \leftarrow SK^*_{ij}$. If they are not equal, (i',j') rejects $InMsg_5$

If $j'^{(2)} \neq j'$ and $k^{(2)} \neq k$, (i',j') is again computationally lonely (the only i-originator it could possibly connect to is open for connection from a different i'-responder through a different server instance) so (i',j') rejects $InMsg_5$.

Message deliveries to server instances and password testing

Let $(0,k)$ be a server instance.

Similarly to the cases of the user instances, the comments made regarding "implementation" and "termination" operations at the beginning of the paragraph on messages delivered to originators apply here.

Now, we need to show how message deliveries made to servers are translated into "exchange completed" operations, and - most importantly - "test password" operations.

Delivering the first protocol message Suppose $\mathcal{M}$ has some first protocol message $InMsg_1$ delivered to $(0,k)$.

• If $InMsg_1$ was computed by a user instance, $\mathcal{M}^*$ can process $InMsg_1$ exactly as $\mathcal{RM}_6^h$ does.

• Suppose that $InMsg_1$ was not computed by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_1$ to get some plaintext string w. If w is not of the form $(U, pw, OID_{0k}, CID_{0k})$, for U a non-trivial group element in G, and $pw \in D$, $InMsg_1$ is rejected. We can therefore suppose from here on that w is of correct format.

•• Suppose that $OID_{0k} = ID$ where ID was not assigned to a user. Then ID was necessarily registered through a "set password" operation, so $\mathcal{M}^*$ knows pw_{ID}. $\mathcal{M}^*$ can thus imply compare pw and pw_{ID}. If they are equal, $InMsg_1$ is accepted and $\mathcal{M}^*$ computes $OutMsg_2$. Otherwise, $\mathcal{M}^*$ rejects.

•• Now suppose that $OID_{0k} = ID_i$ for some initialized user i.

••• If pw_i is known to $\mathcal{M}^*$, i.e. if i has been the target of a "reveal password" query or a successful password guess (defined just below), $\mathcal{M}^*$ can compare pw and pw_i. If they match, the protocol continues and otherwise $InMsg_1$ is rejected.

••• If pw_i is not known to $\mathcal{M}^*$, $\mathcal{M}^*$ makes a "test instance password" query on input $\big(i,(0,k),pw\big)$ to $\mathcal{RM}^*$. If the test succeeds, $\mathcal{M}^*$ continues the protocol, and otherwise it rejects $InMsg_1$. Note that if the test succeeds, $\mathcal{M}^*$ learns pw_i.

Delivering the third protocol message Suppose $\mathcal{M}$ has some third message $InMsg_3$ delivered to $(0,k)$. Let U and n be the originator group element and hash index held by $(0,k)$.

• If $InMsg_3$ was computed by a user instance, $\mathcal{M}^*$ can compute the response as $\mathcal{RM}_6^h$.

• Suppose now that $InMsg_3$ was not computed by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_3$ to get plaintext w. If w is not of the form $(U,V,n,pw,OID_{0k},CID_{0k})$ for some non-trivial group element V and password pw, the message is rejected. Let us therefore suppose w is of the correct form.

•• If $CID_{0k} = ID$ where ID was not issued to a user, ID was necessarily the input to a "set password" operation, and thus pw_{ID} is known to $\mathcal{M}^*$, which can thus compute the appropriate response. If the message is accepted, an "exchange completed" operation is made to $\mathcal{RM}^*$ on input $(0,k)$.

•• If $CID_{0k} = ID_{i'}$ for some initialized i', there are two more cases.

••• If $pw_{i'}$ is already known to $\mathcal{M}^*$, $\mathcal{M}^*$ can already compute the appropriate response. If $InMsg_3$ is accepted, an "exchange completed" operation is made.

••• If $pw_{i'}$ is not known, $\mathcal{M}^*$ makes a "test instance password" query on input $\big(i,(0,k),pw\big)$. If the test is successful, $InMsg_3$ is accepted, an "exchange completed" operation is made, and $pw_{i'}$ is known to $\mathcal{M}^*$. If not, the message is rejected.

Coin collection Once $\mathcal{M}$ has finished interacting with $\mathcal{M}^*$, $\mathcal{M}^*$ makes one final "implementation" operation to place all of $\mathcal{M}$'s random coins in the transcript.

This completes the description of how $\mathcal{M}^*$ is constructed from $\mathcal{M}$ and how $\mathcal{M}^*$ interacts with $\mathcal{RM}^*$.

Proposition 7 *$\mathcal{M}^*$ (as constructed above) faithfully follows the rules of the ideal world, and the transcript random variables $\mathcal{HW}_6(\mathcal{M})$ and $\mathcal{IW}(\mathcal{M}^*)$ are identically distributed.*

Proof: The first statement of the proposition follows by construction. All of the ideal-world operations that $\mathcal{M}^*$ makes are legal in the sense that all of the ideal-world conditions that need to be satisfied for them to be made indeed are. Next, we need to check that connection assignments are a strict function of the transcript up to the current "start session" operation. This can be seen by construction as well. Observe that all connection assignments are determined by conditions on previously made message deliveries, and between connection assignments the conditions are mutually exclusive.

The second part of the statement is clear for most operations, since $\mathcal{M}^*$ essentially runs $\mathcal{RM}_6^h$. Only two points need to be checked: **a).** that session key assignments are identically distributed in both worlds in the case of correct exchanges and **b).** that server instance responses are identically distributed in both worlds in case the delivery of messages that are not generated by users.

Point **a).** is true because in both worlds originators that are assigned session keys when they have participated in a correct exchange received uniformly distributed ones. The only difference is that in the sixth hybrid world, $\mathcal{RM}_6^h$ chooses it while in the ideal world, $\mathcal{RM}^*$ chooses it. This difference does not change $\mathcal{M}$'s view however since in both worlds, $\mathcal{M}$ will receive a session key if and only if it makes a "reveal session key" query. Until this happens, the session keys have no effect on the adversary's behavior.

As for point **b).**, it is true because in both interactions - between $\mathcal{M}$ and $\mathcal{RM}_6^h$ in the sixth hybrid and between $\mathcal{M}$ and $\mathcal{M}^*$ in the ideal world - $\mathcal{M}$ has the messages it concocts itself accepted if and only if they decrypt to the correct password. How this is checked - directly by $\mathcal{RM}_6^h$, who has access to all passwords, or via "test instance password" queries by $\mathcal{M}^*$, who only has access to corrupted passwords - is irrelevant since this check is outside of $\mathcal{M}$'s view.

This completes the proof of the proposition, and therefore of theorem 1. ■

9 Prot2 is Secure Against Password-and-State-Adaptive Network Adversaries

As in the previous section, this section begins with a sketch of how the simulation is carried out to establish theorem 2. The remainder of the section is the proof. Given that **Prot1** and **Prot2** share many ingredients - e.g. signatures and encryption - some parts of the proof below are unfortunately boringly similar to some parts in the previous section. We detail them anyway, for a). readers who may be interested in one protocol and not the other and b). pedagogical reasons.

There are a few notable differences between the proofs of security of both protocols, which will be of course more apparent in the details. Perhaps the most important is that **Prot2** does not achieve key confirmation by using the suffix of a hash; instead, the last protocol message is signed. On one hand, this simplifies the overall proof; indeed, dealing with confirmation codes in **Prot1** was quite the battle, and much more delicate than dealing with signatures. On the other hand, our aim is to study **Prot2** in the password-and-state-adaptive network adversary model, so we have to juggle with instance corruptions that reveal the session key prematurely to the adversary. Making sure that we carry out the simulation consistently is made more challenging by this.

9.1 The Proof Sketch for Prot2

As in section refPoSProt1, $\mathcal{RM}^*$ is the ideal-world ring master, $\mathcal{M}^*$ is the ideal-world adversary under construction, and $\mathcal{M}$ is the real-world adversary $\mathcal{M}^*$ is using as a subroutine. $\mathcal{M}^*$ uses its queries to $\mathcal{RM}^*$ to deal with queries from $\mathcal{M}$.

We omit the setup phase, which is straightforward. We also omit all the queries $\mathcal{M}$ makes that have ideal-world counterparts, as these are directly forwarded by $\mathcal{M}^*$ to $\mathcal{RM}^*$. We only show how message deliveries are interpreted.

Text in italics explains informally how some of the security properties of the involved primitives intervene.

The case of an originator instance

- **Computing the first message**

  Suppose (i,j) is asked to compute the first protocol message. $\mathcal{M}^*$ selects $x \leftarrow \mathbb{Z}_q^*$ randomly, computes $X \leftarrow g^x$, and outputs $OutMsg_1 \leftarrow (X, ID_i, PID_{ij})$ to $\mathcal{M}$.

- **Receiving the fourth protocol message and computing the fifth**

  Suppose (i,j) is waiting for the fourth protocol message, and $\mathcal{M}$ has $InMsg_4$ delivered to (i,j). Let $X = g^x$ be the group element chosen for (i,j). If $InMsg_4$ is not of the form (V, n, σ) for a non-trivial group element V, a hash index n, and string σ, $\mathcal{M}^*$ rejects. We therefore suppose $InMsg_4$ is of correct format.

  $\mathcal{M}^*$ then computes the signature verification equation on input

$$(1, X, V, n, ID_i, PID_{ij}, \sigma)$$

If verification fails, $InMsg_4$ is rejected. If it succeeds, the unforgeability of the signature scheme shows that there exists a server instance $(0,k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that computed some fourth protocol message $OutMsg_4$ on input (X, V, n). Thus, $(0,k)$ chose n, and by the size of the hash index space, $(0,k)$ is unique. $(0,k)$ necessarily received as a first message the $OutMsg_1$ computed by (i,j). Let $OutMsg_2$ be the second message computed by $(0,k)$, and let $InMsg_3$ be the third message received by $(0,k)$. $\mathcal{M}^*$ computes

$$OutMsg_5 \leftarrow \mathcal{E}_{pk_E}(2, X, V, n, 1^{\ell_{pw}}, ID_i, PID_{ij})$$

and outputs $OutMsg_5$ to $\mathcal{M}$. $\mathcal{M}^*$ starts (i,j)'s session and now we must specify how to compute (i,j)'s session key.

The semantic security of Enc implies that $\mathcal{M}$'s behavior will not change even though pw_i is replaced by $1^{\ell_{pw}}$. Since pw_i is not readily available to $\mathcal{M}^*$, this is $\mathcal{M}^*$'s only option.

•• Suppose $PID_{ij} = ID$ was not assigned to a user. Then $InMsg_3$ was not generated by a user instance. In this case, (i,j) is exposed through $(0,k)$. $\mathcal{M}^*$ specifies the session key $SK^*_{ij} \leftarrow H_n(V^x)$.

•• Suppose $PID_{ij} = ID_{i'}$ for some initialized i'.

••• Suppose $InMsg_3$ was not computed by a user instance. Since $InMsg_3$ was accepted, by the way server instances are simulated (see below), we know that $pw_{i'}$ is known to $\mathcal{M}^*$. In this case too, (i,j) is exposed through $(0,k)$. $\mathcal{M}^*$ specifies the session key $SK^*_{ij} \leftarrow H_n(V^x)$.

••• Suppose $InMsg_3$ was computed by a user instance. This instance is necessarily a responder of the form (i',j') with $PID_{i'j'} = ID_i$, (i',j') previously received $OutMsg_2$, and by the size of the encryption randomness space, (i',j') is unique. Furthermore, (i',j') also chose $V = g^v$ randomly.

•••• Suppose (i,j) was corrupted after computing its first protocol message. Then $\mathcal{M}^*$ has already released (X,x) to $\mathcal{M}$. (i,j) is exposed through $(0,k)$ using the relaxed exposure rule. $\mathcal{M}^*$ specifies the session key $SK^*_{ij} \leftarrow H_n(V^x)$.

•••• Suppose (i,j) was not corrupted after computing its first protocol message.

••••• Suppose (i',j') was corrupted after sending $InMsg_3$. Then $\mathcal{M}^*$ has computed $MK_{i'j'} \leftarrow H_n(X^v)$ and has released $(MK_{i'j'}, X, V, n)$ to $\mathcal{M}$. (i,j) is then exposed through $(0,k)$ using the special exposure rule, and (i',j') becomes bound. $\mathcal{M}^*$ specifies the session key $SK^*_{ij} \leftarrow H_n(V^x)$.

••••• Suppose (i',j') was not corrupted after sending $InMsg_3$. In this case, (i,j) is opened from connection from (i',j') through $(0,k)$. $\mathcal{RM}^*$ selects SK^*_{ij} randomly from $\{0,1\}^{\ell_{SK}}$.

The DDH assumption ensures that $\mathcal{M}$ will not change its behavior despite this substitution.

• **Corrupting an originator**

•• Suppose $\mathcal{M}$ corrupts (i,j) before (i,j) has sent a first protocol message. Then $\mathcal{M}^*$ returns the string $InternalState_{ij} \leftarrow \varepsilon$ to $\mathcal{M}$.

•• Suppose $\mathcal{M}$ corrupts (i,j) after (i,j) sent its first protocol message and before receiving a fourth message. Then $\mathcal{M}^*$ has chosen $X = g^x$ for (i,j), and $\mathcal{M}^*$ returns $InternalState_{ij} \leftarrow (X,x)$ to $\mathcal{M}$.

The case of a responder instance

• **Receiving the second protocol message and computing the third**

Suppose (i',j') receives from $\mathcal{M}$ a second protocol message $InMsg_2$. If $InMsg_2$ is not of the form $(U, n, PID_{i'j'}, ID_{i'})$ for a non-trivial group element U and a hash index n, the message is rejected. Otherwise, $\mathcal{M}^*$ selects $y \leftarrow \mathbb{Z}^*_q$ randomly, computes

$$OutMsg_3 \leftarrow \mathcal{E}_{pk_E}(1, U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'})$$

and outputs $OutMsg_3$ to $\mathcal{M}$.

Here again $\mathcal{M}^*$ is forced to replace $pw_{i'}$ with $1^{\ell_{pw}}$ but Enc's properties ensure that $\mathcal{M}$ does not notice the difference. Notice also that Y is replaced by 1_G. An explanation for this can be found in section 9.7. For the proof sketch, exhibiting this change is not crucial, but this way we remain consistent with the simulator constructed in the full proof. Note that when we described above the originator's computation of the fifth message, only the password was replaced. This is because the originator's chosen group element was already made available to the adversary in the first message, so replacing it with 1_G in the fifth message buys us nothing.

• **Receiving the sixth protocol message**

Suppose $\mathcal{M}$ delivers $InMsg_6$ to (i',j'). Let $InMsg_2$ be the second protocol message received by (i',j'), let $OutMsg_3$ be the third message computed by (i',j'), and let $Y = g^y$ be the group element and chosen for (i',j'). Let U and n be the originator group element and hash index in $InMsg_2$.

•• Suppose that $OutMsg_3$ was not accepted by any server instance.

••• Suppose that (i', j') was not corrupted after receiving $InMsg_2$. Then $\mathcal{M}^*$ simply has $InMsg_6$ rejected.

This happens because since Enc is non-malleable, the only way to have a server instance give information on Y is to deliver exactly $OutMsg_3$ to $(0, k)$. If this does not happen, $\mathcal{M}$ has another option to obtain Y: corrupting (i', j'). (This was not an option in the model used to study ***Prot1****.)*

••• Suppose that (i', j') was corrupted after receiving $InMsg_2$. Under these conditions, $\mathcal{M}^*$ has computed $MK_{i'j'} \leftarrow H_n(U^y)$, and $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ was released to $\mathcal{M}$.

•••• If $InMsg_6$ is not of the form (Y, σ) for some string σ, $InMsg_6$ is rejected.

•••• Suppose $InMsg_6$ is of the form (Y, σ). $\mathcal{M}^*$ computes the signature verification equation on input

$$(2, U, Y, n, PID_{i'j'}, ID_{i'}, \sigma)$$

••••• If verification fails, $InMsg_6$ is rejected.

••••• If verification succeeds, by the unforgeability of the signature, there exists a server instance $(0, k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed a sixth message $OutMsg_6$ on input (U, Y, n). By the size of the hash index space, $(0, k)$ is unique, and must have computed $InMsg_2$. Also, since $OutMsg_3$ was not accepted by $(0, k)$, $(0, k)$ must have received a third protocol message $InMsg_3$ containing (U, Y, n). Since $(0, k)$ computed a sixth message, its exchange has been completed. $\mathcal{M}^*$ starts (i', j')'s session.

Let $InMsg_1$ and $InMsg_5$ be the first and fifth messages received by $(0, k)$ and let $OutMsg_4$ be the fourth message computed by $(0, k)$.

•••••• Suppose $PID_{i'j'} = ID$ was not assigned to a user. In this case, $\mathcal{M}^*$ exposes (i', j') through $(0, k)$, and specifies $SK_{i'j'} \leftarrow MK_{i'j'}$.

•••••• Suppose now that $PID_{i'j'} = ID_i$ for some initialized user i.

••••••• Suppose that the following **is not verified**: there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2, U, Y, n, 1^{\ell_{pw}}, ID_i, ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2, U, Y, n, ID_i, ID_{i'})$. In this case, $InMsg_5$ is not generated by a user and no originator instance has participated in a correct exchange with (i', j') through $(0, k)$. Also, by the response to server message deliveries (see below), pw_i is known to $\mathcal{M}^*$. (i', j') is exposed through $(0, k)$. $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow MK_{i'j'}$ to $\mathcal{RM}^*$.

••••••• Suppose that the above-mentioned condition **is verified**, i.e. there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2, U, Y, n, 1^{\ell_{pw}}, ID_i, ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2, U, Y, n, ID_i, ID_{i'})$. Then (i, j) must have output $InMsg_1$, must have received a fourth protocol message containing the same tuple as $OutMsg_4$, and by the size of G, (i, j) is unique. *(Since the originator's group element is actually encrypted even in the case of an honestly generated message, we do not need the size of the encryption randomness to make this argument.)* Furthermore, $U = g^u$ has been chosen for (i, j). The response to receiving the fourth protocol message shows that (i, j)'s session has been started. Since $InMsg_3$ was not generated by a user, we know that (i, j) has been exposed through $(0, k)$, and that SK^*_{ij} was set to $H_n(Y^u)$ by $\mathcal{M}^*$. In this case, (i', j') is exposed through $(0, k)$ using the relaxed exposure rule. $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow MK_{i'j'}$ to $\mathcal{RM}^*$.

•• Suppose now that $OutMsg_3$ was accepted by a server instance, thereby revealing Y to $\mathcal{M}$. By the rules governing server message deliveries (see below), this server instance $(0, k)$ has $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$, and has computed $InMsg_2$. By the size of the hash index space, $(0, k)$ is unique. Let $InMsg_1$ be the first protocol message received by $(0, k)$.

••• If $InMsg_6$ is not of the form (Y, σ) for some string σ, $InMsg_6$ is rejected.

••• Suppose $InMsg_6$ is of the form (Y, σ). $\mathcal{M}^*$ computes the signature verification equation on input

$$(2, U, Y, n, PID_{i'j'}, ID_{i'}, \sigma)$$

•••• If verification fails, $InMsg_6$ is rejected.

•••• If verification succeeds, by the unforgeability of the signature we know that instance $(0, k)$ necessarily computed a sixth message $OutMsg_6$ on input (U, Y, n). Furthermore, $(0, k)$'s exchange has been completed. Let $OutMsg_4$ be the fourth message computed by $(0, k)$ and let $InMsg_5$ be the fifth message computed by $(0, k)$. Since the verification equation has passed, $\mathcal{M}^*$ accepts the message and starts (i', j')'s session. We now need to determine which connection assignment is used and how the session key is computed.

••••• Suppose that $PID_{i'j'} = ID$ is not assigned to a user. In this case, $InMsg_1$ and $InMsg_5$ were not computed by a user. (i', j') is exposed through $(0, k)$, and $\mathcal{M}^*$ specifies $SK_{i'j'} \leftarrow H_n(U^y)$.

••••• Suppose now that $PID_{i'j'} = ID_i$ for some user i.

•••••• Suppose that the following **is not verified**: there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2, U, Y, n, 1^{\ell_{pw}}, ID_i, ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2, U, Y, n, ID_i, ID_{i'})$. In this case, $InMsg_5$ is not generated by a user and no originator instance has participated in a correct exchange with (i', j') through $(0, k)$. Also, by the response to server message deliveries (see below), pw_i is known to $\mathcal{M}^*$. (i', j') is exposed through $(0, k)$. $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow H_n(U^y)$ to $\mathcal{RM}^*$.

•••••• Suppose that the above-mentioned condition **is verified**, i.e. there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2, U, Y, n, 1^{\ell_{pw}}, ID_i, ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2, U, Y, n, ID_i, ID_{i'})$. Then (i, j) must have output $InMsg_1$, must have received a fourth protocol message containing the same tuple as $OutMsg_4$, and by the size of G, (i, j) is unique. Furthermore, $U = g^u$ has been chosen for (i, j). The response to receiving the fourth protocol message shows that (i, j)'s session has been started.

••••••• Suppose that (i, j) has been corrupted after sending $InMsg_1$. Then $\mathcal{M}^*$ has released the string $InternalState_{ij} \leftarrow (U, u)$ to $\mathcal{M}$. According to the rules on delivering the fourth protocol message, (i, j) has already been exposed using the relaxed exposure rule, and $\mathcal{M}^*$ has specified $SK^*_{ij} \leftarrow H_n(Y^u)$ to $\mathcal{RM}^*$.

•••••••• Suppose (i', j') has not been corrupted after sending $OutMsg_3$. (i', j') is exposed using the special exposure rule, (i, j) becomes bound, and $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow H_n(U^y)$ to $\mathcal{RM}^*$.

•••••••• Suppose (i', j') has been corrupted after sending $OutMsg_3$. (i', j') is exposed using the relaxed exposure rule, and $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow H_n(U^y)$.

••••••• Suppose that (i, j) has not been corrupted after sending $InMsg_1$.

•••••••• Suppose that (i', j') has been corrupted.

••••••••• If (i', j') was corrupted before $InMsg_4$ was delivered to (i, j), $\mathcal{M}^*$ has already computed $MK_{i'j'} \leftarrow H_n(U^y)$ and released $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ to $\mathcal{M}$. By the rules on originator instances, (i, j) has been exposed using the special exposure rule, $\mathcal{M}^*$ has specified $SK^*_{ij} \leftarrow H_n(Y^u)$, and (i', j') is now bound. (i', j') is now exposed using the relaxed exposure rule, and $\mathcal{M}^*$ specifies $SK^*_{i'j'} \leftarrow H_n(U^y)$.

••••••••• If (i', j') was corrupted after $InMsg_4$ was delivered to (i, j), $\mathcal{M}^*$ has already opened (i, j) for connection from (i', j') through $(0, k)$, and so SK^*_{ij} was selected uniformly at random from $\{0, 1\}^{\ell_{SK}}$. Corruption having occurred after this event, $\mathcal{M}^*$ has prematurely connected (i', j') to (i, j) through $(0, k)$ using the special connection rule *(this happens regardless of whether $(0, k)$ has completed its exchange or not, which is allowed by the ideal-world rules)*, received $SK^*_{i'j'} \leftarrow SK^*_{ij}$ from $\mathcal{RM}^*$, and set $MK_{i'j'} \leftarrow SK^*_{i'j'}$ to give $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ to $\mathcal{M}$. (i, j) was bound when this event happened. In this case, (i', j') stays connected to (i, j) through $(0, k)$, with session key $SK^*_{i'j'}$.

•••••••• Suppose that (i', j') has not been corrupted. Then (i, j) has been opened for connection from (i', j') through $(0, k)$, and $\mathcal{RM}^*$ has selected SK^*_{ij} randomly from $\{0, 1\}^{\ell_{SK}}$. (i', j') is connected to (i, j) through $(0, k)$. It gets session key $SK^*_{i'j'} \leftarrow SK^*_{ij}$.

• **Corrupting a responder**

•• Suppose $\mathcal{M}$ corrupts (i', j') before it receives a second protocol message. Then $\mathcal{M}^*$ returns the string $InternalState_{i'j'} \leftarrow \varepsilon$ to $\mathcal{M}$.

•• Suppose $\mathcal{M}$ corrupts (i',j') after it receives a second protocol message. Let $InMsg_2$ be the message received, and let $OutMsg_3$ be the message output. Let $Y = g^y$ be the group element chosen by (i',j'), and let U and n be the group element and hash index in $InMsg_2$.

••• Suppose $PID_{i'j'} = ID_i$ for an initialized user i, and that there exist a user instance (i,j) with $PID_{ij} = ID_{i'}$ and server instance $(0,k)$ with $PIDS_{0k} = (ID_i, ID_{i'})$ such that **1).** (i,j)'s first message $OutMsg_1$ was received by $(0,k)$, **2).** $(0,k)$ computed $InMsg_2$, **3).** $(0,k)$ received $OutMsg_3$, **4).** (i,j) received and accepted a fourth message $InMsg_4$ containing (U,Y,n), and **5).** (i,j) was not corrupted after sending its first protocol message.

In this case, (i,j)'s session was started, (i,j) has been opened for connection from (i',j') through $(0,k)$, and $\mathcal{RM}^*$ has set SK^*_{ij} randomly. (i',j') is prematurely connected to (i,j) through $(0,k)$, (i,j) becomes bound, and $\mathcal{RM}^*$ gives $SK^*_{i'j'} \leftarrow SK^*_{ij}$ to $\mathcal{M}^*$. $\mathcal{M}^*$ sets $MK_{i'j'} \leftarrow SK^*_{i'j'}$, and gives $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ to $\mathcal{M}$.

••• Suppose the above conditions do not hold. $\mathcal{M}^*$ computes $MK_{i'j'} \leftarrow H_n(U^y)$ and gives to $\mathcal{M}$ the value $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$.

The case of a server instance

• **Receiving the first protocol message and computing the second**

Suppose $\mathcal{M}$ has a first message $InMsg_1$ delivered to $(0,k)$. If $InMsg_1$ is not a tuple of the form (U, OID_{0k}, CID_{0k}) for some non-trivial group element U, $InMsg_1$ is rejected. Otherwise, $InMsg_1$ is accepted, $\mathcal{M}^*$ selects n randomly, and $OutMsg_2 \leftarrow (U, n, OID_{0k}, CID_{0k})$ is output to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth**

Suppose $\mathcal{M}$ has $InMsg_3$ delivered to $(0,k)$. Let $InMsg_1$ be the first message received by $(0,k)$ and let $OutMsg_2$ be the second message output by $(0,k)$. Let n be the hash index chosen for $(0,k)$, and let U be the group element in $InMsg_1$.

•• Suppose first that $InMsg_3$ was not generated by a user instance. In this case, $\mathcal{M}^*$ decrypts $InMsg_3$ to get a plaintext w.

••• Suppose that w is not of the form $(1, U, V, n, pw, OID_{0k}, CID_{0k})$, where V is a non-trivial group element and pw is a password. Then $InMsg_3$ is rejected.

••• Suppose w is of the form $(1, U, V, n, pw, CID_{0k}, OID_{0k})$.

•••• If $CID_{0k} = ID$ is not the identity of an initialized user, ID was the input to a "set password" operation, and so pw_{ID} is known to $\mathcal{M}^*$. If $pw \neq pw_{ID}$, $InMsg_3$ is rejected. Otherwise, $InMsg_3$ is accepted, and $\mathcal{M}^*$ computes $OutMsg_4 \leftarrow \mathcal{S}_{sk_S}(2, U, V, n, OID_{0k}, CID_{0k})$, and $OutMsg_4$ is given to $\mathcal{M}$.

•••• Suppose now that $CID_{0k} = ID_{i'}$ for some initialized i'. If $pw_{i'}$ is known to $\mathcal{M}^*$, $\mathcal{M}^*$ compares pw and $pw_{i'}$ and computes the appropriate response. If $pw_{i'}$ is not known to $\mathcal{M}^*$, $\mathcal{M}^*$ makes a "test instance password" request on input $\big((0,k), i', pw\big)$. If the answer is negative, $InMsg_3$ is rejected. Otherwise, the appropriate $OutMsg_4$ is computed, and $pw_{i'}$ becomes known to $\mathcal{M}^*$.

The non-malleability of Enc is crucial here. The intuition behind the reasoning is that since $\mathcal{M}$ cannot feasibly modify an honestly encrypted value to change e.g. a group element, it has to re-encrypt a plaintext of its own, and therefore input a candidate password. In short, non-malleability makes it possible to interpret all encryptions created by the adversary as password tests.

•• Now suppose that $InMsg_3$ was generated by a user instance. Then by the size of the encryption randomness space this instance is unique.

••• Suppose that $CID_{0k} = ID_{i'}$ for some user i', that the unique instance that computed $InMsg_3$ is a responder instance (i',j') with $PID_{i'j'} = OID_{0k}$, and that (i',j') received and accepted $OutMsg_2$. In this case, $\mathcal{M}^*$ has $(0,k)$ accept $InMsg_3$ without decrypting. Let Y be the group element chosen by (i',j'). $OutMsg_4$ is computed using Y.

Here, $\mathcal{M}^*$ can observe directly which messages are supposed to be accepted, so it does not need to decrypt. This is technically important to be able to reason with IND-CCA-2 security, see section 9.6.

••• Suppose that the above conditions are not met. Then $\mathcal{M}^*$ has $(0,k)$ reject $InMsg_3$ without decrypting.

• **Receiving the fifth protocol message and computing the sixth**

Suppose $(0,k)$ is waiting for the fifth protocol message. Let $InMsg_1$ be the first message received, let $OutMsg_2$ be the second message sent, let $InMsg_3$ be the third message received, and let $OutMsg_4$ be the fourth message sent. Let U be the originator group element received, n be the chosen hash index, and V be the responder group element received. Let $InMsg_5$ be delivered to $(0,k)$.

•• Suppose that $InMsg_5$ was not generated by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_5$, to get plaintext w.

••• If w is not of the form $(2, U, V, n, pw, OID_{0k}, CID_{0k})$ where $pw \in D$, $InMsg_5$ is rejected.

••• If w is of the form $(2, U, V, n, pw, OID_{0k}, CID_{0k})$, $\mathcal{M}^*$ examines OID_{0k}.

•••• If $OID_{0k} = ID$ was not assigned to a user, ID was the input to a "set password" operation, so $\mathcal{M}^*$ has pw_{ID}. If $pw = pw_{ID}$, $InMsg_5$ is accepted, $OutMsg_6$ is computed as normal, and $(0,k)$ is the input to a "exchange completed" request. Otherwise, $InMsg_5$ is rejected.

•••• If $OID_{0k} = ID_i$ for some initialized user i, $\mathcal{M}^*$ checks whether pw_i is known or not. If pw_i is known, $\mathcal{M}^*$ carries out the comparison to compute the response. If pw_i is not known, $\mathcal{M}^*$ makes a "test instance password" query on input $\big((0,k), i, pw\big)$ and computes the response based on $\mathcal{RM}^*$'s answer.

•• Suppose now that $InMsg_5$ was generated by a user instance. Then this instance is unique.

••• Suppose that $OID_{0k} = ID_i$ for some user i, that the unique user instance that computed $InMsg_5$ is an originator of the form (i,j) with $PID_{ij} = CID_{0k}$), that (i,j) computed $InMsg_1$, and that (i,j) accepted a fourth protocol message $InMsg_4$ containing $(2, U, V, n)$. In this case, $\mathcal{M}^*$ accepts $InMsg_5$ without decrypting, and computes $OutMsg_6$ as normal. $(0,k)$ is the input to an "exchange completed" request.

••• Suppose that the above conditions are not all met. $InMsg_5$ is rejected.

The sketch of **Prot2**'s security proof is now complete.

9.2 The Real World

Let $\mathcal{M}$ be an adversary, and let $\mathcal{RM}$ be the real-world ring master. The game played between $\mathcal{M}$ and $\mathcal{RM}$ is exactly the one described in paragraph 6.2. We just specify here certain details about how the server $\mathcal{T}$ is run, and how unerased internal state is revealed.

Let $\eta \in \mathbb{N}$ be the security parameter.

• **Initialize server** $\mathcal{RM}$ first constructs (q, G, g) and $\mathcal{H} := \{H_n\}_n$ from η. Let $\mathcal{K}_H$ be the space of hash indexes. The H_n functions map into $\{0,1\}^{\ell_{SK}}$, for all $n \in \mathcal{K}_H$. The integer ℓ_{SK} is all assumed to be long enough in η. $\mathcal{RM}$ then runs $\mathcal{K}_E(1^\eta)$ and $\mathcal{K}_S(1^\eta)$ to respectively obtain (pk_E, sk_E) and (pk_S, sk_S), and the server public/private key pair is set to $(pk_{\mathcal{T}}, sk_{\mathcal{T}}) \leftarrow \big((pk_E, pk_S), (sk_E, sk_S)\big)$.

The tuple $(1^\eta, q, G, g, \mathcal{H}, \ell_{pw}, pk_{\mathcal{T}})$ is given to $\mathcal{M}$, who can now begin its interaction with $\mathcal{RM}$. Recall that $\mathcal{M}$ first constructs a non-empty dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$, which it gives to $\mathcal{RM}$, along with an efficient algorithm to sample elements of D uniformly at random, and an efficient algorithm to recognize whether an element of $\{0,1\}^{\ell_{pw}}$ is in D or not. The server has access to $sk_{\mathcal{T}}$.

• **Unerased internal state** We make explicit here what is actually leaked by a "corrupt instance" operation according to the protocol specification.

Let (i,j) be an originator instance expecting to receive the fourth protocol message. It has necessarily already sent the first protocol message, and so $\mathcal{RM}$ has chosen an exponent x uniformly at random, computed $X \leftarrow g^x$, and given $InMsg_1 \leftarrow (X, ID_i, PID_{ij})$ to $\mathcal{M}$. Here we have $InternalState_{ij} = (X, x)$.

Let (i',j') be a responder instance expecting to receive the sixth protocol message. At this point, $\mathcal{RM}$ has chosen exponent y uniformly at random for (i',j'), and computed $Y \leftarrow g^y$. Furthermore, (i',j') has previously received some group element U and hash key n through a second protocol message, and $\mathcal{RM}$ has computed the master key $MK_{i'j'} \leftarrow H_n(U^y)$. In this situation, $InternalState_{i'j'} = (MK_{i'j'}, U, Y, n)$.

Initialized user instances that are in neither of the above two cases have empty unerased internal state, i.e. $InternalState_{**} \leftarrow \varepsilon$.

9.3 The First Hybrid World: Delaying the Computation of Responder Master Keys

Let $\mathcal{M}$ be an adversary in this world. The change made to the real world to obtain this world is minimal: instead of having a responder instance (i',j') compute its master key upon receipt of the third protocol message, the ring master $\mathcal{RM}_1^h$ computes this value only if the adversary chooses to corrupt (i',j') between the time it receives the third protocol message, and the time it receives the sixth protocol message. The view $\mathcal{M}$ has of the execution is strictly identical to the one it would have in the real world. Therefore:

Proposition 8 *The transcript random variables $\mathcal{RW}(\mathcal{M})$ and $\mathcal{HW}_1(\mathcal{M})$ are identically distributed.*

Before continuing, let us reassert how transcripts are built in the hybrid worlds: they are built following the same rules as in the real world.

9.4 The Second Hybrid World: Unique Random Choices

In the second hybrid world, the ring master $\mathcal{RM}_2^h$ interacts with PPTA adversary $\mathcal{M}$ exactly like $\mathcal{RM}_1^h$ does with $\mathcal{M}$ in the first hybrid world except in how exponents, hash indexes, and server nonces are chosen.

Specifically, at the "initialize server" operation, $\mathcal{RM}_2^h$ initializes the sets $\mathcal{CE}$ of chosen exponents, $\mathcal{CHI}$ of chosen hash indexes, and $\mathcal{CER}$ of chosen encryption randomness, which start out empty. Whenever a user instance is called upon to generate a group element, $\mathcal{RM}_2^h$ selects the exponent uniformly at random from $\mathbb{Z}_q^* - \mathcal{CE}$, and adds it to $\mathcal{CE}$. Whenever a server instance must produce a new a hash index, $\mathcal{RM}_2^h$ chooses it in $\mathcal{K}_H - \mathcal{CHI}$, and adds it to $\mathcal{CHI}$. Whenever a user instance must select an encryption randomness, $\mathcal{RM}_2^h$ chooses it from $\mathcal{R}_E - \mathcal{CER}$, and adds it to $\mathcal{CER}$.

As in the previous section, we call these the Nonce Uniqueness Rules (NUR), and their purpose is to uniquely associate instances to the random choices they make. More precisely, it allows the ring master to uniquely associate delivered messages to the instances that computed them. We now prove:

Proposition 9 *The transcript random variables $\mathcal{HW}_1(\mathcal{M})$ and $\mathcal{HW}_2(\mathcal{M})$ are statistically close.*

Proof: Let $\mathcal{T}$ be the set of transcripts that can be yielded by ϕ or ψ. We wish to show that the expression

$$\sum_{t\in\mathcal{T}} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right|$$

is negligible in η.

We define two subsets of $\mathcal{T}$, as follows:

1). Let $\mathcal{T}_O$ be the set of transcripts $t \in \mathcal{T}$ such that t was obtained from an interaction between the ring master and $\mathcal{M}$ in which the ring master made only pairwise distinct exponent, hash key, and encryption randomness choices. Clearly, ϕ will always yield values in $\mathcal{T}_O$, and ψ may yield values in $\mathcal{T}_O$.

2). Let $\mathcal{T}_U$ be the set of transcripts $t \in \mathcal{T}$ such that t was obtained from an interaction between the ring master and $\mathcal{M}$ in which the ring master chose an exponent, or hash key, or encryption randomness at least twice. Clearly, ψ may yield values in $\mathcal{T}_U$.

By definition, we have $\mathcal{T} = \mathcal{T}_O \cup \mathcal{T}_U$.

Lemma 5 *$\mathcal{T}_O$ and $\mathcal{T}_U$ are disjoint.*

Proof: Let t be in both sets. This means that there is a configuration ci_O of the interaction between the ring master and $\mathcal{M}$ in which the ring master made only pairwise distinct exponent, hash key, and encryption randomness choices that yielded t, and a configuration ci_U of the interaction between the ring master and $\mathcal{M}$ in which the ring master chose an exponent, or hash key, or encryption randomness at least twice that also yielded t.

Suppose that in ci_U the ring master chose the same exponent x twice. If this exponent was selected for two different originator instances, then the same group element X was placed directly in t as the output of these two different originator instances. But this could only be yielded by ci_O if the ring master in that configuration had chosen the same exponent twice as well, which is a contradiction. Suppose now that x was selected for an originator instance and a responder instance. Then the corresponding group element X was placed directly in t as the output of the originator, and a string c of the form $\mathcal{E}_{pk_E}(..., X, ...)$ was placed in t as the output of the responder, where pk_E is the public key used in ci_U. Since c was also placed in t as the output of the responder in ci_O, c is also of the form $\mathcal{E}_{pk'_E}(..., Y, ...)$, where pk'_E is the public key used in ci_O, and Y is the group element chosen for the responder in ci_O. Notice that we must have $pk_E = pk'_E$ as this public key is placed directly in t. This in turn implies that $sk_E = sk'_E$. Therefore, decrypting c in ci_O gives the same plaintext as in ci_U and thus $Y = X$. This shows again that in ci_O the ring master must have chosen the same exponent twice, which is a contradiction. The case where x was selected for two different responders leads to a contradiction with the same argument.

The reasoning in the case of two hash indexes being the same across two different server instances is treated exactly as is the case of the same exponent being chosen across two originators, since hash indexes are immediately placed in the transcript.

We now deal with the encryption randomness. Suppose that in ci_U the ring master chose the same encryption randomness twice across two different user instances. Then a string c of the form $\mathcal{E}_{pk_E}(m; r)$ was placed in the transcript and another string c' of the form $\mathcal{E}_{pk_E}(m'; r)$ was placed in the transcript as well, where r designates the same encryption randomness used for both messages m and m'. Since c and c' appear in the transcript, they were also produced by configuration ci_O. Since we already remarked that both configurations use the same public/private key pair for encryption, we thus know that c and c' also decrypt in ci_O to m and m^p respectively. Let r_1 and r_2 be the encryption randomness the ring master used in ci_O to compute c and c'. We get the equalities $\mathcal{E}_{pk_E}(m; r) = c = \mathcal{E}_{pk_E}(m; r_1)$ and $\mathcal{E}_{pk_E}(m'; r) = c' = \mathcal{E}_{pk_E}(m'; r_2)$. But we made the assumption that Enc is injective in the randomness argument (see paragraph 7.3) for fixed messages. Thus, these equalities imply $r_1 = r = r_2$. Since $r_1 = r_2$ is impossible in ci_O, we again have a contradiction. This concludes the proof of the lemma. ■

Two immediate corollaries of the lemma are first that $\mathcal{T} = \mathcal{T}_O \sqcup \mathcal{T}_U$ and second that ϕ cannot yield values in $\mathcal{T}_U$. Thus, we can write

$$\sum_{t \in \mathcal{T}} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| = \sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| + \mathbb{P}[\psi \in \mathcal{T}_U]$$

and we now must analyze $\sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right|$ and $\mathbb{P}[\psi \in \mathcal{T}_U]$. To this end, let $\mathcal{X}$ denote the random variable consisting of the exponent, hash index, and encryption randomness choices that $\mathcal{RM}_1^h$ makes, let $\mathcal{Z}$ denote the space of values this random variable may reach, and let $\widetilde{\mathcal{Z}}$ be the subset of $\mathcal{Z}$ consisting of those values that respect the NUR.

By definition of $\mathcal{T}_U$, we have that $\psi \in \mathcal{T}_U$ if and only if $\mathcal{X} \notin \widetilde{\mathcal{Z}}$. Thus, $\mathbb{P}[\psi \in \mathcal{T}_U] = \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$.

Let us now work on $\sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right|$. We have:

$$\begin{aligned}\sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| &= \sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}] - \mathbb{P}[\psi = t | \mathcal{X} \notin \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]\right| \\ &= \sum_{t \in \mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t | \mathcal{X} \in \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}]\right|\end{aligned}$$

where the fact that $\mathbb{P}[\psi = t|\mathcal{X} \notin \widetilde{\mathcal{Z}}] = 0$ is a consequence of the lemma, t being an element of $\mathcal{T}_O$. Next, conditioned on $\mathcal{X} \in \widetilde{\mathcal{Z}}$, $\mathcal{RM}_1^h$'s actions are identically distributed to those of $\mathcal{RM}_2^h$. Thus, for $t \in \mathcal{T}_O$ we necessarily have $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t|\mathcal{X} \in \widetilde{\mathcal{Z}}]$ and we can write:

$$\begin{aligned}
\sum_{t\in\mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| &= \sum_{t\in\mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t|\mathcal{X} \in \widetilde{\mathcal{Z}}]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}]\right| \\
&= \sum_{t\in\mathcal{T}_O} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\phi = t]\mathbb{P}[\mathcal{X} \in \widetilde{\mathcal{Z}}]\right| \\
&= \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}] \sum_{t\in\mathcal{T}_O} \mathbb{P}[\phi = t] \\
&= \mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]
\end{aligned}$$

In the end, we have obtained that $\sum_{t\in\mathcal{T}} \left|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\right| = 2\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$, so all that is left to prove is that $\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$ is a negligible quantity.

Lemma 6 *$\mathbb{P}[\mathcal{X} \notin \widetilde{\mathcal{Z}}]$ is negligible.*

Proof: The proof is ***strictly identical*** to that of lemma 2. ■

9.5 The Third Hybrid World: Secure Signatures

The third hybrid world is defined by having the ring master $\mathcal{RM}_3^h$ behave exactly like $\mathcal{RM}_2^h$, except in how it computes responses to the reception of messages that should be signed. Only the "deliver user message" operation is modified.

- **Delivering the fourth protocol message and computing the fifth** Let (i, j) be an originator instance expecting to receive the fourth protocol message. Suppose $\mathcal{M}$ has $InMsg_4$ delivered to (i, j). Let X be the group element chosen for (i, j).

  $\mathcal{RM}_3^h$ first sees if $InMsg_4$ is of the form (V, n, σ) for a group element $V \neq 1_G$, a hash index n, and a string σ. If not, $InMsg_4$ is rejected. Otherwise, $\mathcal{RM}_3^h$ checks the signature on input $(1, X, V, n, ID_i, PID_{ij}, \sigma)$. If the verification fails, $InMsg_4$ is rejected. If it succeeds, $\mathcal{RM}_3^h$ checks to see if there exists a server instance $(0, k)$ that computed a fourth protocol message on input $(1, X, V, n, ID_i, PID_{ij})$. If no such instance exists we shall say that $InMsg_4$ was *forged*. In this case, $InMsg_4$ is rejected even though the verification equation has passed. Otherwise, $InMsg_4$ is accepted, and (i, j) continues the protocol according to specification.

- **Delivering the sixth protocol message** Let (i', j') be a responder instance expecting to receive the sixth protocol message, and suppose that $InMsg_6$ is delivered to (i', j') by $\mathcal{M}$. Let Y be the group element chosen for (i', j'). Let U and n be the originator group element and hash index (i', j') received in the second protocol message.

  $\mathcal{RM}_3^h$ checks that $InMsg_6$ is of the form (Y, σ) for some string σ. If not, the message is rejected. Otherwise, $\mathcal{RM}_3^h$ computes the verification equation on input $(2, U, Y, n, PID_{i'j'}, ID_{i'}, \sigma)$. If it fails, the message is rejected. If it succeeds, $\mathcal{RM}_3^h$ checks to see if there exists a server instance $(0, k)$ that computed a sixth protocol message on input $(2, U, Y, n, PID_{i'j'}, ID_{i'})$. If no such instance exists we shall say that $InMsg_6$ was forged, and $InMsg_6$ is rejected, despite the verification equation having passed. Otherwise, $InMsg_6$ is accepted and (i', j') pursues its computations.

Proposition 10 *The transcript random variables $\mathcal{HW}_2(\mathcal{M})$ and $\mathcal{HW}_3^h(\mathcal{M})$ are statistically close.*

Proof: Let $\phi := \mathcal{HW}_3(\mathcal{M})$ and $\psi := \mathcal{HW}_2(\mathcal{M})$. Let $\mathcal{T}$ be the set of all possible transcripts that ϕ or ψ can evaluate to. As usual, we partition $\mathcal{T}$ into sets to work with:

1). $\mathcal{T}_2$ is the sets of transcripts in $\mathcal{T}$ in which there exists a user instance that accepted a fourth or sixth protocol message that was forged. Clearly, ψ may yield such a transcript.

2). $\mathcal{T}_3$ is the set of transcripts in $\mathcal{T}$ in which there exists a user instance that rejected a fourth or sixth protocol message that was forged. ϕ may yield such a transcript.

3). $\mathcal{T}_O$ is the set of remaining transcripts, i.e. those in which if a user instance accepted a fourth or sixth protocol message, then this message was not forged, and if a user instance rejected a fourth or sixth protocol message, this message failed either the format check or the signature verification equation. Both ψ and ϕ may yield such transcripts.

Again, it is clear that $\mathcal{T} = \mathcal{T}_O \cup \mathcal{T}_2 \cup \mathcal{T}_3$. Also, it is clear by definition that $\mathcal{T}_O$ is disjoint from $\mathcal{T}_2$ and $\mathcal{T}_3$. Let $t \in \mathcal{T}_2 \cap \mathcal{T}_3$. Since $t \in \mathcal{T}_2$, there exists a user instance that accepted a fourth or sixth message that was forged. By the rules that $\mathcal{RM}_3^h$ follows, this cannot happen in the third hybrid world. Therefore, t was obtained in an interaction with $\mathcal{RM}_2^h$. However, t is also an element of $\mathcal{T}_3$, i.e. there is a user instance that rejected a four or sixth protocol message that was forged. Since this cannot occur in the second hybrid world, we have a contradiction. Thus, we have shown that $\mathcal{T}_2$ and $\mathcal{T}_3$ are disjoint as well. Therefore:

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \mathbb{P}[\phi \in \mathcal{T}_3] + \mathbb{P}[\psi \in \mathcal{T}_2]$$

and we study the three terms in this sum.

First of all, for all $t \in \mathcal{T}_O$ we have $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t]$ because by construction the interaction between the adversary and ring master is identically distributed in both worlds provided no forged fourth or sixth messages are accepted. Thus, we obtain

$$\sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = 0$$

Next, we examine $\mathbb{P}[\psi \in \mathcal{T}_2]$ and $\mathbb{P}[\phi \in \mathcal{T}_3]$. We show that we can define a PPTA $\mathcal{B}$ that uses $\mathcal{M}$ as a subroutine, plays against a challenger $\mathcal{CH}$ in the $\mathbb{G}^{EU-ACMA}$ game (see appendix B.2), and for which we have

$$\mathbb{P}[\psi \in \mathcal{T}_2] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_3]$$

The security of the signature scheme will then yield the desired result.

To begin the game on input 1^η, $\mathcal{CH}$ runs $\mathcal{K}_S(1^\eta)$ to get (pk_S, sk_S), and gives pk_S to $\mathcal{B}$. $\mathcal{B}$ generates the remaining public parameters himself and hands them to $\mathcal{M}$, which returns a dictionary D back to $\mathcal{B}$. $\mathcal{B}$ then plays ring master's role in an interaction with $\mathcal{M}$, respecting the NUR the same way $\mathcal{RM}_2^h$ or $\mathcal{RM}_3^h$ do. We show now how $\mathcal{B}$ treats the generation and reception of signed messages.

• **Receiving the third protocol message and computing the fourth** Let $InMsg_3$ be delivered to a server instance $(0, k)$. $\mathcal{B}$ accepts or rejects this message according to protocol specification. If it is accepted, let U be the originator group element, V be the responder group element, and n be $(0, k)$'s hash index. clB makes a "signature" query to $\mathcal{CH}$ on input $(1, U, V, n, OID_{0k}, CID_{0k})$. Once $\mathcal{CH}$ answers with a string σ, $\mathcal{B}$ outputs $OutMsg_4 \leftarrow (V, n, \sigma)$ to $\mathcal{M}$.

• **Receiving the fourth protocol message and computing the fifth** Let $InMsg_5$ be delivered to an originator (i, j). Let X be the group element chosen for (i, j). $\mathcal{B}$ first checks if $InMsg_4$ is of the form (V, n, σ) for a non-trivial group element V, a hash index n, and a string σ. If not, $InMsg_4$ is rejected. Otherwise, $\mathcal{B}$ verifies the signature on input $(1, X, V, n, ID_i, PID_{ij}, \sigma)$. If the verification fails, $InMsg_4$ is rejected. If it succeeds, $\mathcal{B}$ checks to see if there exists a server instance $(0, k)$ that computed a fourth protocol message on input $(1, X, V, n, ID_i, PID_{ij})$. If no such instance exists then $\mathcal{B}$ has never made a "signature" query on input $(1, X, V, n, ID_i, PID_{ij})$. It therefore halts the game with $\mathcal{CH}$, outputting $\big((1, X, V, n, ID_i, PID_{ij}), \sigma\big)$ in a "forgery" query. Otherwise, $InMsg_4$ is accepted, and (i, j) continues the protocol according to specification.

- **Receiving the fifth protocol message and computing the sixth** Let $(0,k)$ be a server instance and let $InMsg_5$ be delivered to $(0,k)$. $\mathcal{B}$ accepts or rejects this message as $\mathcal{RM}_2^h$ or $\mathcal{RM}_3^h$ would. If it is accepted, let U be the originator group element, V be the responder group element, and n be $(0,k)$'s hash index. $\mathcal{B}$ makes a "signature" query on input $(2, U, V, n, CID_{0k}, OID_{0k})$ to get σ from $\mathcal{CH}$, and outputs $OutMsg_6 \leftarrow (V, \sigma)$ to $\mathcal{M}$.

- **Receiving the sixth protocol message** Let (i', j') be a responder instance and suppose that $InMsg_6$ is delivered to (i', j') by $\mathcal{M}$. Let Y be the group element chosen for (i', j'), and let U and n be the originator group element and hash index (i', j') received previously. $\mathcal{B}$ checks that $InMsg_6$ is of the form (Y, σ) for some string σ. If not, the message is rejected. Otherwise, $\mathcal{B}$ computes the verification equation on input $(2, U, Y, n, PID_{i'j'}, ID_{i'})$. If it fails, the message is rejected. If it succeeds, $\mathcal{B}$ checks to see if there exists a server instance $(0,k)$ that computed a sixth protocol message on input $(2, U, Y, n, PID_{i'j'}, ID_{i'})$. If no such instance exists, then $(2, U, Y, n, PID_{i'j'}, ID_{i'})$ was never input to a "signature" query. Thus, $\mathcal{B}$ outputs $\big((U, Y, n, PID_{i'j'}, ID_{i'}), \sigma\big)$ in a "forgery" query, and the game with $\mathcal{CH}$ ends, with total output 1. Otherwise, (i', j') pursues its computations as normal.

This completes the description of $\mathcal{B}$. It is certainly a PPTA. We now relate $\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1\Big]$ to $\mathbb{P}[\phi \in \mathcal{T}_2]$ and $\mathbb{P}[\psi \in \mathcal{T}_3]$. We have that the events "$\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1$", "$\phi \in \mathcal{T}_2$", and "$\psi \in \mathcal{T}_3$" occur if and only if in any of these three interactions adversary $\mathcal{M}$ outputs a forged signature at least once. But by construction, up until this event happens these three interactions are identically distributed. Therefore, we indeed have $\mathbb{P}[\psi \in \mathcal{T}_2] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_3]$. This proves the proposition. ■

9.6 The Fourth Hybrid World: Using the Encryption

In this hybrid world, we make use of the security of the encryption scheme. As in the previous section, it is mainly used to prevent offline dictionary attacks. Recall that in the previous section, we also used the encryption to hide group elements, and this had a profound effect on the rest of the proof, namely because of the presence of a confirmation code. While we do indeed hide one of the group elements in this world (the responder's), the effect is not so dramatic. It still cannot be completely ignored, because as we shall see, there are two ways for the adversary to reveal a responder's group element: the first and most obvious way is through delivery of the message to the appropriate server and the second is through an instance corruption. Indeed, careful examination of the protocol specification indicates that the responder's group element is a part of its UIS. Without it, the last signature cannot be verified.

The changes made in this world involve both the computation and treatment upon delivery of the third and fifth messages. Thus, we modify both the "deliver user message" and "deliver server message" operations. Let $\mathcal{M}$ be the adversary.

- **Delivering the second protocol message and computing the third** Let (i', j') be a responder that $\mathcal{M}$ has some second protocol message $InMsg_2$ delivered to, and suppose (i', j') accepts $InMsg_2$. Let U and n be the originator group element and hash index in $InMsg_2$ and Y be (i', j')'s group element. Ring master $\mathcal{RM}_4^h$ computes the encryption $OutMsg_3 \leftarrow \mathcal{E}_{pk_E}(1, U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'})$, and outputs $OutMsg_3$ to $\mathcal{M}$.

- **Delivering the third protocol message and computing the fourth** Let $(0,k)$ be a server instance waiting for the third protocol message, and suppose that $\mathcal{M}$ has $InMsg_3$ delivered to $(0,k)$. Let $OutMsg_2$ be the second protocol message computed by $(0,k)$.

•• Suppose $InMsg_3$ was not computed by any user instance. In this case, $\mathcal{RM}_4^h$ computes $(0,k)$'s response as in the previous hybrid world.

•• Suppose $InMsg_3$ was computed by some user instance. By the NUR (specifically, by the uniqueness of the used encryption randomness), this instance is unique.

●●● Suppose that $PIDS_{0k} = (OID_{0k}, ID_{i'})$ for some user i', that the instance that computed $InMsg_3$ is a responder of the form (i', j') with $PID_{i'j'} = OID_{0k}$, and that (i', j') received $OutMsg_2$ as the second protocol message. In this case, $(0, k)$ accepts $InMsg_3$ automatically without decrypting and computes $OutMsg_4$ using the group element in $OutMsg_2$ and the group element and hash index chosen for (i', j').

●●● Suppose that the situation is not that described above. $(0, k)$ automatically rejects the message without decrypting.

• **Delivering the fourth protocol message and computing the fifth** Let (i, j) be an originator expecting to get the fourth protocol message, and let $InMsg_4$ be delivered to (i, j). $InMsg_4$ is accepted or rejected according to the rules of $\mathcal{HW}_3$. Suppose that (i, j) accepts. Then we know that there exists a server instance $(0, k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that computed a fourth protocol message $OutMsg_4$ containing the same tuple as $InMsg_4$, the originator group element within $InMsg_4$ is the group element X chosen for (i, j), and by the NUR $(0, k)$ is unique. Let V be the responder group element $InMsg_4$ and n be $(0, k)$'s hash index. $\mathcal{RM}_4^h$ computes $OutMsg_5 \leftarrow \mathcal{E}_{pk_E}(2, X, V, n, 1^{\ell_{pw}}, ID_i, PID_{ij})$ and outputs $OutMsg_5$ to $\mathcal{M}$.

• **Delivering the fifth protocol message and computing the sixth** Let $(0, k)$ be a server instance waiting for the fifth protocol message. Let $OutMsg_4$ be the fourth protocol message output by $(0, k)$. Suppose that $\mathcal{M}$ has $InMsg_5$ delivered to $(0, k)$.

●● Suppose that no user instance computed $InMsg_5$. $\mathcal{RM}_4^h$ computes $(0, k)$'s response as in the previous world.

●● Suppose that some user instance computed $InMsg_5$. By the NUR, this instance is unique.

●●● Suppose that $PIDS_{0k} = (ID_i, CID_{0k})$ for an initialized user i, that the instance that computed $InMsg_5$ is an originator instance of the form (i, j) with $PID_{ij} = CID_{0k}$, and that (i, j) received a fourth protocol message $InMsg_4$ containing the same tuple as $OutMsg_4$. In this case, $\mathcal{RM}_4^h$ has $(0, k)$ automatically accept $InMsg_5$ without decrypting, and computes $OutMsg_6$ using the data in $OutMsg_4$.

●●● In any case other than the one described above, $(0, k)$ automatically rejects $InMsg_5$.

Important Remark: The injectivity property of the encryption function (see paragraph 7.3) is important here for the ring master to keep track of which responder instances compute which third protocol messages, in the case that several of them have been delivered identical second protocol messages. (A similar situation happens in the analysis of **Prot1**.) Indeed, the third protocol message - when computed by a responder - no longer contains the unique group element that responder chose, for it has been replaced by 1_G. To be sure that the third message can be traced back to a unique instance, we must have the guarantee that even if the same plaintext is encrypted several times, the resulting ciphertext will always be different. Our only option is to use the changing encryption randomness to do this.

This completes our description of how messages are computed in this world.

Proposition 11 *The transcript random variables $\mathcal{HW}_3(\mathcal{M})$ and $\mathcal{HW}_4(\mathcal{M})$ are computationally indistinguishable.*

Proof: Let $\phi := \mathcal{HW}_4(\mathcal{M})$, $\psi := \mathcal{HW}_3(\mathcal{M})$, and $\mathcal{D}$ be a PPTA. To show that

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta, \phi) = 1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta, \psi) = 1\big]\Big|$$

is negligible in η we construct a PPTA $\mathcal{B}$ that plays the games $\mathbb{G}_b^{ch-ad-IND-CCA-2}$ for $b \in \{0, 1\}$ with challenger $\mathcal{CH}$ against the encryption scheme Enc (see appendix B.1). $\mathcal{B}$ first simulates the ring master for $\mathcal{M}$ running as a subroutine, and the transcript is built. Once the interaction with $\mathcal{M}$ ends, $\mathcal{B}$ runs $\mathcal{D}$ on input the obtained transcript and outputs to $\mathcal{CH}$ whatever bit $\mathcal{D}$ outputs.

We now detail the construction of $\mathcal{B}$. For $b \in \{0, 1\}$, at the beginning of $\mathbb{G}_b^{ch-ad-IND-CCA-2}$, $\mathcal{B}$ receives as input the candidate encryption key pk_E computed by $\mathcal{CH}$ on input 1^η. $\mathcal{B}$ then sets up all of the other parameters for its interaction with $\mathcal{M}$ itself, gives all public parameters to $\mathcal{M}$, and receives dictionary D in return.

From here on, $\mathcal{B}$ runs the interaction exactly as $\mathcal{RM}_3^h$ or $\mathcal{RM}_4^h$ would, except in the way encrypted messages as produced and processed upon receipt. The operations that need to be modified involve the user and server message deliveries.

• **Receiving the second protocol message and computing the third** Let (i', j^p) be a responder that $\mathcal{M}$ has the second protocol message $InMsg_2$ delivered to. Suppose that $InMsg_2$ is accepted, and let U be the originator group element and n be the hash index in the message. $\mathcal{B}$ selects and exponent $\in \mathbb{Z}_q^*$ uniformly at random respecting the NUR and computes $Y \leftarrow g^y$. It then makes a "challenge" query to $\mathcal{CH}$ on input the pair of messages

$$(1, U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'}) \text{ and } (1, U, Y, n, pw_{i'}, PID_{i'j'}, ID_{i'})$$

to obtain a ciphertext c. $\mathcal{B}$ outputs $OutMsg_3 \leftarrow c$ to $\mathcal{M}$.

• **Receiving the third protocol message and computing the fourth** Let $(0, k)$ be a server instance waiting for the third protocol message and suppose that $InMsg_3$ is delivered to $(0, k)$. Let $OutMsg_2$ be the second message output by $(0, k)$.

•• Suppose $InMsg_3$ was not computed by any user instance. In this case, $\mathcal{B}$ has not obtained $InMsg_3$ from $\mathcal{CH}$ as the result of a "challenge" query. It therefore makes a "decryption" query on input $InMsg_3$ and examines the plaintext according to protocol specification, as $\mathcal{RM}_3^h$ and $\mathcal{RM}_4^h$ would.

•• Suppose $InMsg_3$ was computed by a user instance. By the NUR, this instance is unique.

••• Suppose that $PIDS_{0k} = (OID_{0k}, ID_{i'})$ for some user i', that the instance that computed $InMsg_3$ is a responder of the form (i', j') with $PID_{i'j'} = OID_{0k}$, and that (i', j') received $OutMsg_2$ as the second protocol message. In this case, $\mathcal{B}$ accepts $InMsg_3$ and computes $OutMsg_4$ using the group element and has index in $OutMsg_2$ and the group element chosen for (i', j').

••• Suppose that the situation is not that described above. Then $InMsg_3$ is automatically rejected.

• **Receiving the fourth protocol message and computing the fifth** Let (i, j) be an originator instance that receives the fourth protocol message $InMsg_4$ from $(0, k)$. Let X be the group element chosen for (i, j). $\mathcal{B}$ has $InMsg_4$ accepted according to the same rules as $\mathcal{RM}_3^h$ or $\mathcal{RM}_4^h$. If $InMsg_4$ is accepted, there exists a unique server instance $(0, k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that computed a fourth protocol message $OutMsg_4$ containing the same tuple as $InMsg_4$, and the originator group element within $InMsg_4$ is the group element X. Let V be the responder group element in $InMsg_4$ and n be $(0, k)$'s hash index. $\mathcal{B}$ makes a "challenge" query on input the pair of messages

$$(2, X, V, n, 1^{\ell_{pw}}, ID_i, PID_{ij}) \text{ and } (2, X, V, n, pw_i, ID_i, PID_{ij})$$

to get challenge c. It then outputs $OutMsg_5 \leftarrow c$ to $\mathcal{M}$.

• **Receiving the fifth protocol message and computing the sixth** Let $(0, k)$ be a server instance waiting for the fifth protocol message and suppose that $InMsg_5$ is delivered to (i, j). Let $OutMsg_4$ be the fourth protocol message output by $(0, k)$.

•• Suppose that no user instance computed $InMsg_5$. In this case, $\mathcal{B}$ has not obtained $InMsg_5$ from a "challenge" query, so it makes a "decryption" query to $\mathcal{CH}$ and processes the obtained plaintext as $\mathcal{RM}_3^h$ or $\mathcal{RM}_4^h$ would.

•• Suppose that some user instance computed $InMsg_5$. By NUR, this instance is unique.

••• Suppose that $PIDS_{0k} = (ID_i, CID_{0k})$ for an initialized user i, that the instance that computed $InMsg_5$ is an originator instance of the form (i, j) with $PID_{ij} = CID_{0k}$, and that (i, j) received a fourth protocol message $InMsg_4$ containing the same tuple as $OutMsg_4$. In this case, $\mathcal{B}$ has $(0, k)$ automatically accept $InMsg_5$, and computes $OutMsg_6$ using the data in $OutMsg_4$.

••• In any case other than the one described above, $(0,k)$ automatically rejects $InMsg_5$.

This completes the description of $\mathcal{B}$, which is obviously polynomial-time. We turn to determining $\mathcal{B}$'s advantage in winning the security game. Consider the bit b which indexes $\mathbb{G}_b^{ch-ad-IND-CCA-2}$.

By construction of $\mathcal{B}$, if $b = 0$, whenever $\mathcal{B}$ submits a pair of messages in a "challenge" query, the message that is encrypted by $\mathcal{CH}$ is the one in which the password is replaced by $1^{\ell_{pw}}$ and, in the case of a responder, the new group element is replaced by 1_G. Since by definition $\mathcal{B}$ deals with encrypted message deliveries exactly as $\mathcal{RM}_4^h$, we conclude that the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to the one between $\mathcal{RM}_4^h$ and $\mathcal{M}$.

If $b = 1$, the message that is encrypted by the challenger contains the protocol-specified data, like in the third hybrid world. We need to make sure that $\mathcal{B}$ responds to encrypted messages the same way $\mathcal{RM}_3^h$ does. This is clearly the case if the message received by the server is not user-generated. If the message is user-generated, it is as well because the conditions $\mathcal{B}$ checks for are exactly those that can be verified by examining the contents of the plaintext. Hence, if $b = 1$ the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to the one between $\mathcal{RM}_3^h$ and $\mathcal{M}$.

In particular, if $b = 0$ the obtained transcript is sampled according to ϕ and if $b = 1$ it is sampled according to ψ. Since by construction $\mathcal{B}$ outputs 1 if and only if $\mathcal{D}$ outputs 1, we can therefore conclude that

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta,\phi) = 1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta,\psi) = 1\big]\Big| = \\ \bigg|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big]\bigg|$$

This proves the proposition, by virtue of the assumption on Enc's security. ■

9.7 The Fifth Hybrid World: Group Elements the Adversary Cannot Guess

In this world, we make a small modification to the way the sixth protocol message is treated upon delivery. Specifically, we force responder instances to reject the sixth protocol message under the joint conditions of being uncorrupted after computing the third protocol message and not having had the third protocol message delivered to an adequate server instance. This automatic rejection makes sense because until delivery of the third protocol message, the adversary should have no knowledge of the responder's group element, since it is encrypted. This is why in the previous world, the responder's element is replaced by 1_G in the encryption. The only other way the group element could be revealed is through an instance corruption of (i', j'). Indeed, Y is in (i', j')'s unerased internal state at this point.

This change is necessary to have a consistent definition of what a correct exchange is. Let $\mathcal{M}$ be the adversary.

• **Delivering the sixth protocol message** Let (i', j') be a responder expecting the sixth protocol message and let $InMsg_6$ be delivered to (i', j') by $\mathcal{M}$. Let $OutMsg_3$ be the third message computed by (i', j'). If $OutMsg_3$ was not accepted by a server instance and if (i', j') was not corrupted after computing $OutMsg_3$, we shall say that (i', j') is a *statistically lonely responder.*

In this case, $InMsg_6$ is automatically rejected.

Proposition 12 *The transcript random variables $\mathcal{HW}_5(\mathcal{M})$ and $\mathcal{HW}_4(\mathcal{M})$ are statistically close.*

Proof: Let $\phi := \mathcal{HW}_5(\mathcal{M})$, let $\psi := \mathcal{HW}_4(\mathcal{M})$, and let $\mathcal{T}$ be the set of transcripts ϕ or ψ may evaluate to. We define to following sets.

1). Let $\mathcal{T}_4$ be the set of transcripts in $\mathcal{T}$ such that there exists a statistically lonely responder instance (i', j') that accepted a sixth protocol message.

2). Let $\mathcal{T}_5$ be the set of transcripts in $\mathcal{T}$ such that there exists a statistically lonely responder instance (i', j') that rejected a sixth protocol message that was of correct format, passed the signature verification equation, was not forged, and contains the responder group element Y chosen for (i', j').

3). Let $\mathcal{T}_O$ be all remaining transcripts. These can be described as those transcripts in which all responder instances that accepted sixth protocol messages were not statistically lonely at the time this message was delivered, and all responder instances that rejected a sixth protocol message while being statistically lonely did so because at least one of the checks failed.

By definition, we have that $\mathcal{T} = \mathcal{T}_O \cup \mathcal{T}_4 \cup \mathcal{T}_5$. We also have that $\mathcal{T}_O$ is disjoint from $\mathcal{T}_4$ and $\mathcal{T}_5$. Let $t \in \mathcal{T}_4 \cap \mathcal{T}_5$. Since $t \in \mathcal{T}_4$, some statistically lonely responder accepted a sixth protocol message. But this only happens according to the rules of the fourth hybrid world, so t can only be sampled from ϕ. However, $t \in \mathcal{T}_5$ as well, which means that some responder that was statistically lonely rejected a sixth message that verified all the conditions necessary to be normally accepted. This cannot happen according to the rules in the fourth hybrid world, so we have a contradiction. This shows then that $\mathcal{T} = \mathcal{T}_O \sqcup \mathcal{T}_4 \sqcup \mathcal{T}_5$, that ϕ cannot takes values in $\mathcal{T}_4$, and that ψ cannot take values in $\mathcal{T}_5$. Thus:

$$\sum_{t\in\mathcal{T}} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| = \sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big| + \mathbb{P}[\phi \in \mathcal{T}_4] + \mathbb{P}[\psi \in \mathcal{T}_5]$$

We now study the three terms on the right of the equality.

For $\sum_{t\in\mathcal{T}_O} \big|\mathbb{P}[\phi = t] - \mathbb{P}[\psi = t]\big|$, observe that the interaction between the ring master and the adversary is identically distributed in both hybrid worlds up until the point when one of the events defining $\mathcal{T}_4$ or $\mathcal{T}_5$ occurs. This implies that for $t \in \mathcal{T}_O$, we necessarily have $\mathbb{P}[\phi = t] = \mathbb{P}[\psi = t]$, so the total sum is zero.

For $\mathbb{P}[\phi \in \mathcal{T}_4]$ and $\mathbb{P}[\psi \in \mathcal{T}_5]$, we shall construct an adversary $\mathcal{B}$ that plays against a challenger in the game $\mathbb{G}^{ad-GE-sf_{\ell_2}-ES}$ defined in appendix A.3, but with $\ell_2 = 0$ and without making any "hash index" or "guess hash suffix" queries. In other words, $\mathcal{B}$ is going to only ask for group elements to be prepared and will only try to guess group elements outside of its view. In the end, we will have by construction that

$$\mathbb{P}[\phi \in \mathcal{T}_4] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_0-ES}(1^\eta)\big) = 1\Big] = \mathbb{P}[\psi \in \mathcal{T}_5]$$

yielding the desired result.

We show how to build $\mathcal{B}$, using $\mathcal{M}$ as a subroutine. On input 1^η, the challenger $\mathcal{CH}$ constructs the data $\big((q, G, g), \{H_n\}_n\big)$ where the H_n map into $\{0,1\}^{\ell_{SK}+\ell_2} = \{0,1\}^{\ell_{SK}}$. $\big((q, G, g), \{H_n\}_n\big)$ is then given to $\mathcal{B}$, who generates all of the other parameters for $\mathcal{M}$ itself and gets a dictionary space D from $\mathcal{M}$. We now detail how message deliveries are treated.

- **Computing the first protocol message** Let (i, j) be an originator instance that $\mathcal{M}$ asks to have compute the first protocol message. In response, $\mathcal{B}$ makes a "prepare exponent" query, records the counter C_{ij}, and immediately makes a "recover exponent" query on input C_{ij} to get z_{ij}. It then sets $Z_{ij} \leftarrow g^{z_{ij}}$ as (i, j)'s group element, and continues the protocol.

- **Receiving the first protocol message and computing the second** Let $(0, k)$ be a server instance that receives the first protocol message $InMsg_1$. If the message is accepted, $\mathcal{B}$ selects a hash index n uniformly at random (respecting the NUR) and computes the second protocol message using n.

- **Receiving the second protocol message and computing the third** Let (i', j') be a responder that receives and accepts a second protocol message $InMsg_2$. $\mathcal{B}$ makes a "prepare exponent" query, records the counter $C_{i'j'}$, and computes the third protocol message as $OutMsg_3 \leftarrow \mathcal{E}_{pk_E}(1, U, 1_G, n, 1^{\ell_{pw}}, PID_{i'j'}, ID_{i'})$, where U and n are the originator group element and hash index found in $InMsg_2$.

- **Receiving the third protocol message and computing the fourth** Let $(0, k)$ be a server instance that $\mathcal{M}$ has a third protocol message $InMsg_3$ delivered to. Let $InMsg_1$ be the first message $(0, k)$ received, $OutMsg_2$ be the second message $(0, k)$ computed, and U and n be the associated originator group element and hash index.

  $InMsg_3$ is accepted or rejected by $\mathcal{B}$ following the same rules as $\mathcal{RM}_4^h$ or $\mathcal{RM}_5^h$. Suppose that $InMsg_3$ is accepted, and was computed by a user instance. Then we know that this instance is unique, and is a responder (i', j') with $ID_{i'} = CID_{0k}$ and $PID_{i'j'} = OID_{0k}$, and which received $OutMsg_2$ as second protocol message. In this case, $\mathcal{B}$ makes a "recover exponent" query on input counter $C_{i'j'}$ to recover the exponent $z_{i'j'}$ previously prepared for (i', j'). $\mathcal{B}$ then computes $(0, k)$'s response with responder group element $Z_{i'j'} \leftarrow g^{z_{i'j'}}$.

• **Receiving the sixth protocol message** Let (i', j') be a responder expecting the sixth protocol message, and suppose $InMsg_6$ is delivered to (i', j'). Let $InMsg_2$ be the second message received by (i', j') and $OutMsg_3$ be the third message output by (i', j'). $\mathcal{B}$ has previously made a "prepare exponent" query for (i', j'), and holds counter $C_{i'j'}$. Let U and n be the group element and hash index obtained from $InMsg_2$.

•• Suppose that (i', j') is not statistically lonely. In this case, $\mathcal{B}$ has already made a "recover exponent" query on input $C_{i'j'}$ to compute $Z_{i'j'}$, either because $OutMsg_3$ was delivered to a server instance and accepted by that instance, or because (i', j') was corrupted after receiving $InMsg_2$ (see below). $\mathcal{B}$ computes (i', j')'s response as $\mathcal{RM}_4^h$ or $\mathcal{RM}_5^h$ would.

•• Suppose now that (i', j') is statistically lonely. Then no "recover exponent" query has been yet made on input $C_{i'j'}$. $\mathcal{B}$ first checks to see if $InMsg_6$ is of the form (V, σ) for some non-trivial group element V and a string σ. If not, the message is rejected. Otherwise, $\mathcal{B}$ computes the signature verification equation on input $(2, U, V, n, PID_{i'j'}, ID_{i'}, \sigma)$. If verification fails, the message is rejected. If it succeeds, $\mathcal{B}$ checks to see if some server instance $(0, k)$ computed a sixth protocol message on input $(2, U, V, n, PID_{i'j'}, ID_{i'})$. If not, the message is rejected. Otherwise, $\mathcal{B}$ makes a "guess group element" query on input $(V, C_{i'j'})$. If the guess is correct, $\mathcal{B}$ ends the game with $\mathcal{CH}$ and the total output is 1. Otherwise, $InMsg_6$ is rejected and the interaction continues.

• **Corrupting a responder instance** Let (i', j') be a responder that $\mathcal{M}$ chooses to target in a "corrupt instance" operation. If this is done before (i', j') receives a second protocol message, $\mathcal{B}$ responds with $InternalState_{i'j'} \leftarrow \varepsilon$. If this is done after (i', j') accepts a second protocol message, then $\mathcal{B}$ has made a "prepare exponent" query for (i', j'). In this case, $\mathcal{B}$ makes a "recover exponent" query to get $z_{i'j'}$, computes $MK_{i'j'} \leftarrow H_n(U^{z_{i'j'}})$ and $Z_{i'j'} \leftarrow g^{z_{i'j'}}$, and answers with $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Z_{i'j'}, n)$.

$\mathcal{B}$'s description is complete; we analyze it now. It is clearly a PPTA. We now relate the probability $\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_0-ES}(1^\eta)\big) = 1\Big]$ to $\mathbb{P}[\phi \in \mathcal{T}_5]$ and $\mathbb{P}[\psi \in \mathcal{T}_4]$. The events "$\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_0-ES}(1^\eta)\big) = 1$", "$\phi \in \mathcal{T}_5$", and "$\psi \in \mathcal{T}_4$" occur if and only if in any of these three interactions adversary $\mathcal{M}$ manages to guess the responder group element of a statistically lonely responder. But up until this occurs, the three interactions are identically distributed. Therefore: $\mathbb{P}[\psi \in \mathcal{T}_4] = \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-GE-sf_0-ES}(1^\eta)\big) = 1\Big] = \mathbb{P}[\phi \in \mathcal{T}_5]$, and the proposition is proved. ■

Odd Remark: This proof shows the reason **Prot2** requires the responder's group element to be a part of the sixth protocol message, even though the responder should be holding it in memory anyway. The problem is that in the above construction, $\mathcal{B}$ needs an candidate group element to verify the signature first. (Actually, the exact same issue arises in the security analysis of **Prot1** when dealing with statistically lonely originators.) We have not yet tried to find a way around this.

9.8 The Sixth Hybrid World: Replacing Correctly Exchanged Session and Master Keys with Random Strings

In this hybrid world, we identify those exchanges that are to be considered correct, and therefore that yield session keys that should be replaced by random strings. Care must be taken regarding the computation of master keys on the responder's end. Indeed, the ability to corrupt a responder instance in such a way that its master key can be revealed to the adversary makes the simulation process more delicate, as depending on the time the corruption has occurred an originator instance may have already been assigned a random string as its session key. In any case, the master key the responder holds must absolutely be identical to the session key of the originator it may be partnered to.

To be able to isolate the problematic cases, we need a rather cumbersome set of definitions that separate correct exchanges from correct exchanges ***in which in addition neither of the two user instances has been corrupted***.

Let (i, j) be an originator, (i', j') be a responder, and $(0, k)$ be a server instance, with $PID_{ij} = ID_{i'}$, $PID_{i'j'} = ID_i$, and $PIDS_{0k} = (ID_i, ID_{i'})$.

Definition 3 *We shall say that (i, j) **has participated in a correct** (respectively, **correct and uncorrupted**) **exchange with** (i', j') **through** $(0, k)$ if conditions **1).** through **5).** involving **4′).** (respectively, conditions **1).** through **7).** involving **4).**) below are met:*

1). *(i, j) computed $OutMsg_1$. Let X be the group element chosen for (i, j);*

2). *$(0, k)$ received $OutMsg_1$ and computed $OutMsg_2$. Let n be the hash index chosen by $(0, k)$;*

3). *(i', j') received $OutMsg_2$ and computed $OutMsg_3$. Let Y be the group element chosen for (i', j');*

4). *$(0, k)$ received $OutMsg_3$ and computed $OutMsg_4$;*

4′). *$(0, k)$ received $OutMsg_3$ or (i', j') was corrupted after computing $OutMsg_3$ and $(0, k)$ received and accepted a third message $InMsg_3$ containing (X, Y, n). $(0, k)$ computed $OutMsg_4$;*

5). *(i, j) received and accepted a fourth protocol message $InMsg_4$ containing (X, Y, n), and computed $OutMsg_5$;*

6). *(i, j) was not corrupted after computing $OutMsg_1$;*

7). *(i', j') was not corrupted after computing $OutMsg_3$ and before (i, j) received $OutMsg_4$.*

Definition 4 *We shall say that (i', j') **has participated in a correct** (respectively, **correct and uncorrupted**) **exchange with** (i, j) **through** $(0, k)$ if conditions **1).** through **7).** involving **4′).** (respectively, conditions **1).** through **9).** involving **4).**) below are met:*

1). *(i, j) computed $OutMsg_1$. Let X be the group element chosen for (i, j);*

2). *$(0, k)$ received $OutMsg_1$ and computed $OutMsg_2$. Let n be the hash index chosen by $(0, k)$;*

3). *(i', j') received $OutMsg_2$ and computed $OutMsg_3$. Let Y be the group element chosen for (i', j');*

4). *$(0, k)$ received $OutMsg_3$ and computed $OutMsg_4$;*

4′). *$(0, k)$ received $OutMsg_3$ or (i', j') was corrupted after computing $OutMsg_3$ and $(0, k)$ received and accepted a third message $InMsg_3$ containing (X, Y, n). $(0, k)$ computed $OutMsg_4$;*

5). *(i, j) received and accepted a fourth protocol message $InMsg_4$ containing (X, Y, n), and computed $OutMsg_5$;*

6). *$(0, k)$ received and accepted either $OutMsg_5$ or a fifth message $InMsg_5$ containing (X, Y, n) and computed $OutMsg_6$;*

7). *(i', j') received and accepted a sixth protocol message $InMsg_6$ containing (X, Y, n);*

8). *(i, j) was not corrupted after computing $OutMsg_1$;*

9). *(i', j') was not corrupted after computing $OutMsg_3$.*

Comments on the Definitions The author agrees with the reader that these are horrid. Unfortunately, they are necessary. We make some comments in the same vein as those made when defining correct exchanges in the sixth hybrid world for **Prot1**.

Perhaps the most interesting points that merit some indications are the (mutually excluded by definition) points **4).** and **4′).**. Suppose the adversary does not deliver the third protocol message to the server instance. The security of the encryption ensures that the responder's group element remains hidden in this case. However, a correct exchange can still occur in the bizarre event that the adversary corrupts the instance (i', j') and then uses its newfound knowledge of the responder's group element to compute a new encryption around an otherwise unaltered plaintext. Now, this requires of course that the adversary also already know the responder's password; furthermore, the corruption of the instance ultimately leads to a compromised responder. But this is no reason to not consider the exchange correct as the data chosen by each user instance still reaches its intended destination.

We now describe the changes made to get form the fifth world to the sixth world. These involve changing how to process user instance message deliveries and master key computations. We must also pay attention to how corruptions are handled. To this end, we need the lemma below, which is extremely similar to lemma 4 in section 8.8. Let (i,j), $(0,k)$ and (i',j') be as in definitions 3 and 4.

Lemma 7 *Let $(i',j'^{(1)})$ be another responder instance with $PID_{i'j'^{(1)}} = ID_i$ and $(0,k^{(1)})$ be another server instance with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$. If (i,j) has participated in a correct exchange with (i',j') through $(0,k)$ and with $(i',j'^{(1)})$ through $(0,k^{(1)})$, then $j'^{(1)} = j'$, $k^{(1)} = k$, and $In/OutMsg_a^{(1)} = In/OutMsg_a$ for all $a \in \mathbb{N}$.*

Let $(i,j^{(1)})$ be another originator instance with $PID_{ij^{(1)}} = ID_{i'}$, and let $(0,k^{(1)})$ be another server instance with $PIDS_{0k^{(1)}} = (ID_i, ID_{i'})$. If (i',j') has participated in a correct exchange with (i,j) through $(0,k)$ and with $(i,j^{(1)})$ through $(0,k^{(1)})$, then $j^{(1)} = j$, $k^{(1)} = k$, and $In/OutMsg_a^{(1)} = In/OutMsg_a$ for all a.

Proof: Similar to that of lemma 4. ■

Remark: Note that the lemma holds regardless of whether or not the instances are corrupted.

• **Receiving the fourth protocol message and computing the fifth** Suppose $\mathcal{M}$ has $InMsg_4$ delivered to an originator instance (i,j). Suppose that $PID_{ij} = ID_{i'}$ for some initialized user i', and that there exist a responder instance (i',j') with $PID_{i'j'} = ID_i$ and a server instance $(0,k)$ with $PIDS_{0k} = (ID_i, ID_{i'})$ such that, having been delivered $InMsg_4$, (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$.

In this case, $\mathcal{RM}_6^h$ selects $SK_{ij} \leftarrow \{0,1\}^{\ell_{SK}}$ uniformly at random, and computes $OutMsg_5$ as in the previous hybrid world.

• **Receiving the sixth protocol message** Suppose that $\mathcal{M}$ has $InMsg_6$ delivered to some responder instance (i',j'). Suppose that $PID_{i'j'} = ID_i$ for some initialized i, and that there exist an originator instance (i,j) with $PID_{ij} = ID_{i'}$ and a server instance (i',j') with $PIDS_{0k} = (ID_i, ID_{i'})$ such that, having been delivered $InMsg_6$, (i',j') has participated in a correct exchange with (i,j) through $(0,k)$ and (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$.

In this case, $\mathcal{RM}_6^h$ has already selected SK_{ij} uniformly at random. Also, by lemma 7, (i,j) and $(0,k)$ are the only user and server instances that (i',j') could have participated in a correct exchange with. Thus, $\mathcal{RM}_6^h$ may set $SK_{i'j'} \leftarrow SK_{ij}$.

Remark: We require the exchange to be uncorrupted from the originator's point of view, but not necessarily from the responder's. This will make more sense to the reader after seeing how a special case of responder corruptions are handled below.

• **Corrupting a responder** Let (i',j') be a responder instance. Suppose that $PID_{i'j'} = ID_i$ for some initialized user i, and that there exist an originator (i,j) with $PID_{ij} = ID_{i'}$ and a server instance $(0,k)$ such that (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$. Suppose that $\mathcal{M}$ targets (i',j') in a "corrupt instance" operation after (i,j) has computed its fourth protocol message.

In this case, $\mathcal{RM}_6^h$ has already selected SK_{ij} uniformly at random. $\mathcal{RM}_6^h$ further sets $MK_{i'j'} \leftarrow SK_{ij}$, and uses $MK_{i'j'}$ to compute and return $InternalState_{i'j'}$ to $\mathcal{M}$. This assignment is well-defined again because by lemma 7, (i,j) and $(0,k)$ are the only instances (i',j') could have participated in a correct exchange with.

Remark, cont'd: Since (i,j) has already been assigned a key and at this point the responder's master key is supposed to be identical to this session key, the ring master has no choice but to assign the same value to $MK_{i'j'}$ to answer the corruption. Of course, this does not prevent in any way a correct sixth protocol message being at some point delivered to (i',j'). If this indeed happens, we must also have $SK_{i'j'} = SK_{ij}$. This is why we cannot impose the responder's exchange to be uncorrupted in the rule on delivering the sixth protocol message above.

This completes the description of the sixth hybrid world.

Proposition 13 *The transcript random variables $\mathcal{HW}_6(\mathcal{M})$ and $\mathcal{HW}_5(\mathcal{M})$ are computationally indistinguishable.*

Proof: Let $\mathcal{D}$ be a PPTA that tries to tell $\phi := \mathcal{HW}_6(\mathcal{M})$ and $\psi := \mathcal{HW}_5(\mathcal{M})$ apart. We construct a PPTA $\mathcal{B}$ that uses $\mathcal{D}$ and adversary $\mathcal{M}$ as a subroutines to try telling apart the two games $\mathbb{G}_b^{ad-DDHES}$ for $b = 0$ or $b = 1$. (See appendix A.) Specifically, $\mathcal{B}$ will get the group and hash family parameters from its challenger $\mathcal{CH}$ and pass these on to $\mathcal{M}$. It will then run an interaction with $\mathcal{M}$ according to the rules of the fifth hybrid world, using the queries of the game $\mathbb{G}_b^{ad-DDHES}$ to assign group elements, hash indexes, and master and session keys to user instances. Once a transcript has been built, this transcript is fed to $\mathcal{D}$, and $\mathcal{B}$'s final output will be whatever $\mathcal{D}$ outputs.

We need to make precise the way $\mathcal{B}$ uses its queries in $\mathbb{G}_b^{ad-DDHES}$ to simulate the ring master for $\mathcal{M}$. The setup $\mathcal{B}$ runs with $\mathcal{CH}$ allows it to get $\big((q, G, g), \{H_n\}_n\big)$. $\mathcal{B}$ then generates the remaining setup for **Prot2** itself, and runs the usual setup with $\mathcal{M}$. We now examine ho $\mathcal{B}$ handles message deliveries.

• **Computing the first protocol message** If $\mathcal{M}$ asks that an originator (i, j) compute the first protocol message, $\mathcal{B}$ makes a "left group element" query to get (L, X), and sets $OutMsg_1 \leftarrow (X, ID_i, PID_{ij})$.

• **Receiving the first protocol message and computing the second** If $\mathcal{M}$ asks to have $InMsg_1$ delivered to a server instance $(0, k)$ and this message is accepted, $\mathcal{B}$ makes a "hash index" query to get a hash index n from $\mathcal{CH}$ and computes $OutMsg_2 \leftarrow (U, n, OID_{0k}, CID_{0k})$, where U is the group element obtained in $InMsg_1$.

• **Receiving the second protocol message and computing the third** If $\mathcal{M}$ has some $InMsg_2$ delivered to a responder instance (i', j') and this message is accepted, $\mathcal{B}$ makes a "right group element" query to get (R, Y) and then computes $OutMsg_3$ with Y.

• **Receiving the fourth protocol message and computing the fifth** Let (i, j) be an originator expecting the fourth message and let $InMsg_4$ be delivered to (i, j). $InMsg_4$ is accepted or rejected by $\mathcal{B}$ using the same rules as in the fifth hybrid world. Suppose the message is accepted, and let V and n be the responder group element and hash index in $InMsg_4$. We know that there exists a unique server instance $(0, k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that computed some fourth protocol message on input (X, V, n) where $X = g^x$ is the group element chosen for (i, j) (via a "left group element" query) and n is the hash index chosen for $(0, k)$ (via a "hash index" query). We also know that $(0, k)$ must have received the first protocol message $OutMsg_1$ computed by (i, j). Let $InMsg_3$ be the third message received by $(0, k)$, and $OutMsg_2$ be the second message computed by $(0, k)$. Since (i, j) accepted $InMsg_4$, it computes $OutMsg_5$ as $\mathcal{RM}_6^h$ would; we now need to specify its session key.

•• Suppose $PID_{ij} = ID$ was not used to initialize a user. Then $InMsg_3$ was not computed by user instance. Under these conditions, $\mathcal{B}$ has not made any "challenge" query involving (L, X) as input (see below). Thus, it makes a "left exponent" query to get x, and computes $SK_{ij} \leftarrow H_n(V^x)$.

•• Suppose now that $PID_{ij} = ID_{i'}$ for an initialized i'.

••• Suppose that $InMsg_3$ was not computed by a user instance. Again, $\mathcal{B}$ has not made a "challenge" query query on input (L, X), so it can make a "left exponent" query on (L, X) to get x and compute $H_n(V^x)$.

••• Suppose now that $InMsg_3$ was computed by a user instance. Then we know that this instance is unique, and is a responder of the form (i', j') with $PID_{i'j'} = ID_i$ that received $OutMsg_2$ as a second protocol message. We also know that $V = g^v$ was obtained through a "right group element" query. At this point, (i, j) has participated in a correct exchange with (i', j') through $(0, k)$. We now reason according to whether this exchange is corrupted or not.

•••• Suppose that (i', j') was corrupted after having received $OutMsg_2$. Then according to the rules below, $\mathcal{B}$ has already made a "right exponent" query on input (R, V) to get v and has computed $MK_{i'j'} \leftarrow H_n(X^v)$ to give $InternalState_{i'j'} \leftarrow (MK_{i'j'}, X, V, n)$ to $\mathcal{M}$. Since (L, X) was not the input to a "challenge" query, $\mathcal{B}$ further makes a "left exponent" query on (L, X) to get x and sets $SK_{ij} \leftarrow H_n(V^x)$.

•••• Suppose now that (i', j') was not corrupted after having received $OutMsg_2$.

••••• If (i, j) was corrupted after computing (i, j), $\mathcal{B}$ has already made a "left exponent" query on input (L, X) to get x to give $InternalState_{ij} \leftarrow (X, x)$ to $\mathcal{M}$. It further computes $SK_{ij} \leftarrow H_n(V^x)$.

••••• If (i,j) was not corrupted after computing $OutMsg_1$, the exchange is uncorrupted. By the rules above, $\mathcal{B}$ has made neither a "left exponent" query on (L,X), nor a "right exponent" query on (R,Y). Since by lemma 7 the correct exchange could have only gone through the unique instance $(0,k)$, $\mathcal{B}$ and has also not yet made a "challenge" query involving both (L,X) and (R,V). It thus makes a "challenge" query on input $\big((L,X),(R,V),n\big)$ to get SK_{ij} from $\mathcal{CH}$.

• **Corrupting an originator** Let (i,j) be an originator waiting for the fourth protocol message and suppose $\mathcal{M}$ corrupts (i,j). At this point $\mathcal{B}$ has made a "left group element" query to get (L,X) for (i,j), where $X=g^x$. Since no fourth message was delivered, no key was asked to be computed and in particular, no "challenge" query on input (L,X) was made. Therefore, $\mathcal{B}$ may make a "left exponent" query on (L,X) to get x, and give $InternalState_{ij} \leftarrow (X,x)$ to $\mathcal{M}$.

• **Receiving the second protocol message and computing the third** Suppose $\mathcal{M}$ has a second protocol message $InMsg_2$ delivered to a responder (i',j') and that this message is accepted. Then $\mathcal{B}$ makes a "right group element" query (R,Y) and computes $OutMsg_3$ as $\mathcal{RM}_5^h$ would.

• **Receiving the sixth protocol message** Let (i',j') be a responder instance expecting to receive the sixth protocol message. Let $InMsg_6$ be delivered to (i',j'). Let $InMsg_2$ be the second message accepted by (i',j') and let $OutMsg_3$ be the third message computed by (i',j'). Let $Y=g^y$ be the responder group element obtained for (i',j') and let U and n be the originator group element and hash index received in $InMsg_2$.

$InMsg_6$ is treated exactly as in the fifth hybrid world. Suppose it is accepted. Then we need to assign (i',j') a session key. We already know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that computed $InMsg_2$ and computed a sixth protocol message $OutMsg_6$ on input (U,Y,n). Let $InMsg_1$ and $InMsg_5$ be the first and fifth messages received by $(0,k)$, and $OutMsg_4$ be the fourth message computed by $(0,k)$. We also know that (i',j') is not statistically lonely, so either $OutMsg_3$ was delivered to $(0,k)$ or (i',j') was corrupted after receiving $InMsg_2$.

•• Suppose first that $OutMsg_3$ was not delivered to $(0,k)$. Then we know that (i',j') was corrupted after receiving $OutMsg_3$. We also know that no originator instance has participated in a correct and uncorrupted exchange with (i',j'). Thus, $\mathcal{B}$ has made a "right exponent query" on input (R,Y) to get y, compute $MK_{i'j'} \leftarrow H_n(U^y)$, and release $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ to $\mathcal{M}$. In this case, $\mathcal{B}$ simply sets $SK_{i'j'} \leftarrow H_n(U^y)$.

•• Suppose now that $OutMsg_3$ was delivered to $(0,k)$.

••• Suppose that $PID_{i'j'} = ID$ is not the identity of an initialized user. Then (i',j') has not participated in any correct and uncorrupted exchange with an originator, and thus $\mathcal{B}$ has not made any "challenge" query involving (R,Y). $\mathcal{B}$ makes a "right exponent query" on (R,Y) to get y, and sets $SK_{i'j'} \leftarrow H_n(U^y)$.

••• Suppose now that $PID_{i'j'} = ID_i$ for some initialized i. Then we discuss the origins of $InMsg_5$ (and $InMsg_1$).

•••• Suppose that the following ***is not verified***: *there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2,U,Y,n,1^{\ell_{pw}},ID_i,ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2,U,Y,n,ID_i,ID_{i'})$*. Then $InMsg_5$ was not computed by any user instance, and again no "challenge" query was made involving (R,Y). $\mathcal{B}$ makes a "right exponent" query on input (R,Y) to get y, and computes $SK_{i'j'} \leftarrow H_n(U^y)$.

•••• Suppose that the afore-mentioned condition ***is verified***, i.e. *Then $InMsg_5$ was not computed by any user instance, and again no "challenge" query was made involving (R,Y). $\mathcal{B}$ makes a "right exponent" query on input (R,Y) to get y, and computes $SK_{i'j'} \leftarrow H_n(U^y)$*. Then, this instance is unique, and is necessarily an originator of the form (i,j) with $PID_{ij} = ID_{i'}$. Furthermore, $U=g^u$ was obtained in a "left group element" query, (i,j) computed $InMsg_1$, and (i,j) received some fourth protocol message $InMsg_4$ containing the same data as $OutMsg_4$. At this point, (i',j') has participated in a correct exchange with (i,j) through $(0,k)$. By lemma 7, we know that (i,j) and $(0,k)$ are unique in satisfying this. We now need to examine possible instance corruptions.

••••• Suppose (i,j) was corrupted after sending $InMsg_1$. Then $\mathcal{B}$ has already made a "left exponent" query to get u and release (U,u) to $\mathcal{M}$. Since (i,j) and $(0,k)$ are the unique originator and server instance that have participated in a correct exchange with (i',j'), no "challenge" query has been made involving Y. Thus, $\mathcal{B}$ makes a "right exponent" query on (R,Y) to get y, and computes $SK_{i'j'} \leftarrow H_n(U^y)$. (Note that at this point, $\mathcal{B}$ has also already set $SK_{ij} \leftarrow H_n(Y^u)$.)

••••• Suppose now that (i,j) was not corrupted after sending $InMsg_1$. Then no "left exponent" query has been made on input (L,X).

••••• If (i',j') was corrupted after sending $OutMsg_3$ and before $InMsg_4$ was delivered to (i,j), then no "challenge" query has been made involving (R,Y). $\mathcal{B}$ thus makes a "right exponent" query to get y and computes $SK_{i'j'} \leftarrow H_n(U^y)$. (At this point, $SK_{ij} = H_n(Y^u)$ as well.)

••••• Suppose that (i',j') was not corrupted after sending $OutMsg_3$ and before $InMsg_4$ was delivered to (i,j). Then no "right exponent" query was made on input (R,Y), and $\mathcal{B}$ has already made a "challenge" query on input $\big((L,U),(R,Y),n\big)$ to get SK_{ij}. $\mathcal{B}$ sets $SK_{i'j'} \leftarrow SK_{ij}$. By lemma 7, this assignment makes sense.

• **Corrupting a responder instance** Let (i',j') be a responder instance that $\mathcal{M}$ targets in a "corrupt instance" query after the second protocol message was received by (i',j').

• Suppose that $PID_{i'j'} = ID_i$ for some initialized user i and that there exist an originator (i,j) with $PID_{ij} = ID_{i'}$) and a server instance $(0,k)$ with $PIDS_{0k} = (ID_i, ID_{i'})$ such that (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$. In this case, $\mathcal{B}$ has already made a "challenge" query to obtain SK_{ij}. $\mathcal{B}$ further sets $MK_{i'j'} \leftarrow SK_{ij}$ to compute $InternalState_{i'j'}$ for $\mathcal{M}$.

• In any other case, $\mathcal{B}$ has not made a "challenge" query involving (i',j')'s group element Y. Thus, $\mathcal{B}$ makes a "right exponent" query to get y and computes $MK_{i'j'}$ with y and the originator group element and hash index from the second protocol message.

This completes the description of $\mathcal{B}$. We now analyze it. In game $\mathbb{G}_b^{ad-DDHES}$, if $b=0$, "challenge" queries return uniformly distributed and independent bitstrings. Therefore, by construction we have that the interaction between $\mathcal{B}$ and $\mathcal{M}$ is identically distributed to that between $\mathcal{RM}_6^h$ and $\mathcal{M}$. However, if $b=1$, "challenge" queries yield the exact hashed Diffie-Hellman value. Thus, the interaction between $\mathcal{B}$ and $\mathcal{M}$ is distributed as that between $\mathcal{RM}_5^h$ and $\mathcal{M}$. Since $\mathcal{B}$ outputs exactly what $\mathcal{D}$ outputs, we indeed obtain

$$\Big|\mathbb{P}\big[\mathcal{D}(1^\eta,\phi)=1\big] - \mathbb{P}\big[\mathcal{D}(1^\eta,\psi)=1\big]\Big| = \Big|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ad-DDHES}(1^\eta)\big)=1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ad-DDHES}(1^\eta)\big)=1\Big]\Big|$$

The term on the right being negligible in η by assumption, we have the result. ■

9.9 The Ideal World

Now we can build an ideal-world adversary $\mathcal{M}^*$ from a real-world adversary $\mathcal{M}$. Of course, $\mathcal{M}^*$'s role is to play the part of a sixth-hybrid-world ring master for $\mathcal{M}$ and to funnel $\mathcal{M}$'s actions into ideal-world operations. We are especially interested in how connection assignments are computed and how password testing is identified.

We now build $\mathcal{M}^*$ from $\mathcal{M}$.

Let $\eta \in \mathbb{N}$ be the security parameter. Specifying $\mathcal{M}^*$ means describing its interaction with $\mathcal{RM}^*$ according to the rules of the password-and-state-adaptive ideal world network adversary model. At the beginning of the interaction $\mathcal{RM}^*$ specifies the session key length ℓ_{SK} and the password length ℓ_{pw}; $(1^\eta, \ell_{SK}, \ell_{pw})$ is given to $\mathcal{M}^*$, who gives it to $\mathcal{M}$.

Server initialization

On input $(1^\eta, \ell_{SK}, \ell_{pw})$, $\mathcal{M}$ first asks $\mathcal{M}^*$ to initialize the server. $\mathcal{M}^*$ runs the parameter algorithms and gets

$$\big((q,G,g), \{H_n\}_n, \ell_{SK}, (pk_E, sk_E), (pk_S, sk_S)\big)$$

It passes $\big((q,G,g),\{H_n\}_n,\ell_{SK},pk_E,pk_S\big)$ on to $\mathcal{M}$, who generates a dictionary $D \subset \{0,1\}^{\ell_{pw}} - \{1^{\ell_{pw}}\}$. D and its password sampling and recognizing algorithms are returned to $\mathcal{M}^*$.

Now $\mathcal{M}^*$ performs an "initialize server" operation to $\mathcal{RM}^*$, on input $(0,D)$. It also performs an "implementation" operation on input

$$\Big(server\ \ public\ \ key, \big((q,G,g),\{H_n\}_n,\ell_{SK},pk_E,pk_S\big)\Big)$$

User initialization

When $\mathcal{M}$ asks to have user i initialized on input ID_i, $\mathcal{M}^*$ makes the same request to $\mathcal{RM}^*$. Recall that $\mathcal{RM}^*$ samples a password pw_i from D for i outside of $\mathcal{M}^*$'s view.

Setting up statically corrupted users

When $\mathcal{M}$ makes a "Set Password" request on input (ID,pw) with $pw \in D$, $\mathcal{M}^*$ makes an identical request to $\mathcal{RM}^*$.

Initializing user and server instances

When $\mathcal{M}$ makes "initialize user instance" or "initialize server instance" requests, these are forwarded by $\mathcal{M}^*$ to $\mathcal{RM}^*$.

Dynamically corrupting users

When $\mathcal{M}$ makes a "reveal user password" operation on input i, $\mathcal{M}^*$ forwards the request to $\mathcal{RM}^*$. $\mathcal{RM}^*$'s response is password pw_i, then given to $\mathcal{M}$.

Revealing session keys

When $\mathcal{M}$ makes a "reveal session key" query on input some user instance (i,j) or (i',j'), $\mathcal{M}^*$ simply the same request to $\mathcal{RM}^*$. $\mathcal{RM}^*$ responds with the value of SK^*_{ij}, and $\mathcal{M}^*$ sets $SK_{ij} \leftarrow SK^*_{ij}$, and gives SK_{ij} to $\mathcal{M}$.

Message deliveries to originator instances and connection assignments for the role *open*

Let (i,j) be an originator instance.

Any message delivery $\mathcal{M}$ makes to (i,j) is recorded by $\mathcal{M}^*$ in an "implementation" operation made to $\mathcal{RM}^*$ on input

$$\big((i,j),InMsg,OutMsg,status_{ij}\big)$$

where $InMsg$ is the message delivered to (i,j), $OutMsg$ is the message output by (i,j), and $status_{ij}$ is equal to *accept*, *reject*, or *continue*. To indicate that (i,j) is to generate the first protocol message, $InMsg$ must be set to ε. If (i,j) does not generate a message, $OutMsg$ is set to ε. If (i,j) rejects the message, $\mathcal{M}^*$ makes a "terminate user instance" request to $\mathcal{RM}^*$ on input (i,j).

Suppose first that $\mathcal{M}$ has (i,j) compute the first protocol message. Then $\mathcal{M}^*$ computes $OutMsg_1$ as $\mathcal{RM}^h_6$ would.

Suppose now that $\mathcal{M}$ has a fourth protocol message $InMsg_4$ delivered to (i,j). $\mathcal{M}^*$ has (i,j) accept or reject this message as $\mathcal{RM}^h_6$ would. Suppose it is accepted. Let X be the group element chosen for (i,j), and let V and n be the responder group element and hash index found in $InMsg_4$. We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (ID_i, PID_{ij})$ that chose n and computed a fourth protocol message $OutMsg_4$ on input (X,V,n). We also know that $(0,k)$ must have received as a first protocol message $OutMsg_1$ computed by (i,j). Let $InMsg_3$ be the third message received by $(0,k)$ and let $OutMsg_2$ be the second message computed by $(0,k)$.

$\mathcal{M}^*$ computes $OutMsg_5$ as $\mathcal{RM}^h_6$ would. In particular, $\mathcal{M}^*$ does not need to know pw_i to do this. Since this is the last message (i,j) expects to receive, $\mathcal{M}^*$ makes a "start session" request on input (i,j). We must now show how to assign a session key to (i,j). Let $X = g^x$ be the group element chosen for (i,j).

- If $PID_{ij} = ID$ is not the identity of an initialized user, (i,j) is exposed through $(0,k)$, and $\mathcal{M}^*$ specifies $SK^*_{ij} \leftarrow H_n(V^x)$.

• Suppose $PID_{ij} = ID_{i'}$ for some initialized i'.

•• If $InMsg_3$ was not generated by a user instance, then by the way server instances respond to third message deliveries (below) we know that $pw_{i'}$ is known to $\mathcal{M}^*$. In this case, (i,j) is also exposed through $(0,k)$ and $SK^*_{ij} \leftarrow H_n(V^x)$.

•• Now suppose that $InMsg_3$ was generated by a user instance. By the way server instances respond to third message deliveries, we know that this user instance is unique, is a responder (i',j') with $PID_{i'j'} = ID_i$, and has received $OutMsg_2$. Furthermore, $V = g^v$ was chosen for (i',j') by $\mathcal{M}^*$.

••• If (i,j) was corrupted after computing $OutMsg_1$, then $\mathcal{M}^*$ has released $InternalState_{ij} \leftarrow (X,x)$ to $\mathcal{M}$. (i,j) is then exposed using the relaxed exposure rule, and $SK^*_{ij} \leftarrow H_n(V^x)$.

••• Suppose (i,j) was not corrupted after computing $OutMsg_1$.

•••• Suppose (i',j') was corrupted after computing $InMsg_3$. In this case, $\mathcal{M}^*$ has computed $MK_{i'j'} \leftarrow H_n(X^v)$ and has released $InternalState_{i'j'} \leftarrow (MK_{i'j'}, X, V, n)$ to $\mathcal{M}$. (i,j) is exposed using the special exposure rule, $SK^*_{ij} \leftarrow H_n(V^x)$, and (i',j') becomes bound.

•••• Suppose (i',j') was not corrupted after computing $InMsg_3$. Then at this point, (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$. Thus, (i,j) is opened for connection from (i',j') through $(0,k)$, and $\mathcal{RM}^*$ selects SK^*_{ij} uniformly at random from $\{0,1\}^{\ell_{SK}}$.

Corrupting an originator instance

Let (i,j) be an originator instance. If $\mathcal{M}$ makes a "corrupt instance" operation on (i,j), $\mathcal{M}^*$ makes the same request on (i,j) to $\mathcal{RM}^*$. It also makes an "implementation" operation on input

$$\big(\text{"}internal\ \ state, (i,j), InternalState_{ij}\big)$$

where we must specify how $InternalState_{ij}$ is computed.

If (i,j) has not yet computed a first protocol message, $InternalState_{ij} \leftarrow \varepsilon$. If (i,j) has computed a first protocol message, $\mathcal{M}^*$ has chosen $X = g^x$ for (i,j); $InternalState_{ij} \leftarrow (X,x)$ in this case. $InternalState_{ij}$ is then given to $\mathcal{M}$.

Message deliveries to responder instances and connection assignments for the role *connect*

Let (i',j') be a responder instance.

"Implementation" operations and the $status_{i'j'}$ variable are dealt with as in the case of originator instances.

Suppose $\mathcal{M}$ has the second protocol message $InMsg_2$ delivered to (i',j'). If it is accepted, $\mathcal{M}^*$ computes the third protocol message of (i',j') as $\mathcal{RM}^h_6$. $\mathcal{M}^*$ does not need $pw_{i'}$ to do this.

Suppose now that $\mathcal{M}$ has the sixth message $InMsg_6$ delivered to (i',j'). It is accepted or rejected by $\mathcal{M}^*$ as $\mathcal{RM}^h_6$ would. Suppose that it is accepted. Let $Y = g^y$ be the group element chosen for (i',j'), let $OutMsg_3$ be the third message computed by (i',j'), and let $InMsg_2$ be the second message received by (i',j'). Let U and n be the originator group element and hash index present in $InMsg_2$.

We know that there exists a unique server instance $(0,k)$ with $PIDS_{0k} = (PID_{i'j'}, ID_{i'})$ that chose n and computed a sixth protocol message $OutMsg_6$ on input (U,Y,n). Thus, $(0,k)$ has been the input to a "exchange completed" operation in the ideal world. We also know that (i',j') is not statistically lonely, i.e. that $OutMsg_3$ was delivered to $(0,k)$ or that (i',j') was corrupted after computing $OutMsg_3$. $\mathcal{M}^*$ makes a "start session" request on input (i',j'), and we now examine how $SK^*_{i'j'}$ is computed.

• Suppose that $OutMsg_3$ was not delivered to $(0,k)$. Then (i',j') was corrupted after computing $OutMsg_3$. Since $OutMsg_3$ was not delivered, the special connection rule (see below) was not used. Thus, $\mathcal{M}^*$ has computed $MK_{i'j'} \leftarrow H_n(U^y)$ and given $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$ to $\mathcal{M}$. Let $InMsg_3$ be the third protocol message $(0,k)$ has received.

•• Suppose $PID_{i'j'} = ID$ is not the identity of an initialized user. Then (i',j') is exposed through $(0,k)$, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

•• Suppose now that $PID_{i'j'} = ID_i$ for some initialized i. Let $InMsg_1$ and $InMsg_5$ be the first and fifth protocol messages received by $(0,k)$ and let $OutMsg_4$ be the fourth message computed by $(0,k)$.

••• Suppose the following property ***is not verified***: *there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2,U,Y,n,1^{\ell_{pw}},ID_i,ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2,U,Y,n,ID_i,ID_{i'})$*. Then $InMsg_5$ was not computed by a user instance and by the way server instances treat fifth message deliveries (see below), we know that pw_i is known to $\mathcal{M}^*$. Thus, (i',j') is exposed through $(0,k)$, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

••• Suppose that *there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input the tuple $(2,U,Y,n,1^{\ell_{pw}},ID_i,ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains the string* $(2,U,Y,n,ID_i,ID_{i'})$. Then this instance is unique, is of the form (i,j) with $PID_{ij} = ID_{i'}$, and has picked $U = g^u$. It has also necessarily computed $InMsg_1$, and received a fourth protocol message $InMsg_4$ containing the same data as $OutMsg_4$. Thus, it has been the target of a "start session" operation, and since $InMsg_3$ was not computed by a user instance, (i,j) has been exposed through $(0,k)$ (according to the rules above). In this case, (i',j') is exposed through $(0,k)$ using the relaxed exposure rule, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

• Suppose now that $OutMsg_3$ was delivered to $(0,k)$.

•• If $PID_{i'j'} = ID$, (i',j') is exposed through $(0,k)$, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

•• Suppose $PID_{i'j'} = ID_i$ for some i.

••• Suppose the following property ***is not verified***: *there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input $(2,U,Y,n,1^{\ell_{pw}},ID_i,ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains $(2,U,Y,n,ID_i,ID_{i'})$*. Then $InMsg_5$ was not computed by a user instance and we know that pw_i is known to $\mathcal{M}^*$. (i',j') is exposed through $(0,k)$, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

••• Suppose that *there exists an originator instance that computed a fifth protocol message $OutMsg_5$ on input the tuple $(2,U,Y,n,1^{\ell_{pw}},ID_i,ID_{i'})$ and either $InMsg_5 = OutMsg_5$, or $InMsg_5$ contains the string* $(2,U,Y,n,ID_i,ID_{i'})$. Then this instance is unique, is of the form (i,j) with $PID_{ij} = ID_{i'}$, has picked $U = g^u$, has computed $InMsg_1$, and has received a fourth protocol message $InMsg_4$ containing the same data as $OutMsg_4$. It has been the target of a "start session" operation.

•••• Suppose (i',j') was corrupted after computing $OutMsg_3$.

••••• Suppose the corruption of (i',j') occurred before (i,j) received $InMsg_4$. (i',j') is exposed using the relaxed exposure rule and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

••••• Suppose the corruption of (i',j') occurred after (i,j) received $InMsg_4$.

•••••• Suppose (i,j) was not corrupted after sending $InMsg_1$. Then (i,j) was opened for connection from (i',j') through $(0,k)$, and (i',j') was already connected to (i,j) through $(0,k)$ using the special connection rule. It remains so.

•••••• Suppose (i,j) was corrupted after sending $InMsg_1$. Then (i,j) was exposed using the relaxed exposure rule. (i',j') is exposed using the relaxed exposure rule as well, and $SK^*_{i'j'} \leftarrow H_n(U^y)$.

•••• Suppose (i',j') was not corrupted after computing $OutMsg_3$.

••••• If (i,j) was corrupted after sending $InMsg_1$, (i,j) was exposed using the relaxed exposure rule. (i',j') is exposed using the special exposure rule, and (i,j) is now bound.

••••• If (i,j) was not corrupted after sending $InMsg_1$, (i,j) was opened for connection from (i',j') through $(0,k)$, $\mathcal{RM}^*$ has selected $SK^*_{ij} \leftarrow \{0,1\}^{\ell_{SK}}$ uniformly at random, and now (i',j') is connected to (i,j) through $(0,k)$. $\mathcal{RM}^*$ sets $SK^*_{i'j'} \leftarrow SK^*_{ij}$.

Corrupting a responder instance

Let (i',j') be a responder instance, and suppose that $\mathcal{M}$ makes a "corrupt instance" operation on (i',j'). $\mathcal{M}^*$ forwards the request to $\mathcal{RM}^*$, and makes an "implementation" operation on

$$\left(\text{''}internal\ \ state, (i',j'), InternalState_{i'j'}\right)$$

where $InternalState_{i'j'}$ is computed as follows.

If (i',j') has not accepted a second protocol message, $InternalState_{i'j'} \leftarrow \varepsilon$. Suppose (i',j') has accepted a second protocol message. Let U and n be the originator group element and hash index (i',j') received, and let Y be the group element (i',j') chose.

• If $PID_{i'j'} = ID_i$ for an initialized user i and there exist an originator instance (i,j) and a server instance $(0,k)$ such that (i,j) has participated in a correct and uncorrupted exchange with (i',j') through $(0,k)$, then we know (i,j) has been opened for connection from (i',j') through $(0,k)$ and has thus been assigned session key SK^*_{ij} randomly. (i',j') is connected to (i,j) through $(0,k)$ using the special connection rule, and $\mathcal{RM}^*$ gives SK^*_{ij} to $\mathcal{M}^*$. In this case, $InternalState_{i'j'} \leftarrow (SK^*_{ij}, U, Y, n)$.

• In any case other than that above, $\mathcal{M}^*$ computes the values $MK_{i'j'} \leftarrow H_n(U^y)$ and $InternalState_{i'j'} \leftarrow (MK_{i'j'}, U, Y, n)$.

Message deliveries to server instances and password testing

Let $(0,k)$ be a server instance.

"Implementation" operations and the $status_{0k}$ variable are dealt with as in the cases of originator and responder instances.

Delivering the first protocol message and computing the second This is handled in a completely straightforward way by $\mathcal{M}^*$. No password testing is done.

Delivering the third protocol message and computing the fourth Let $InMsg_3$ be delivered to $(0,k)$ by $\mathcal{M}$. Let U and n be the originator group element and hash index held by $(0,k)$.

• If $InMsg_3$ was computed by a user instance, $\mathcal{M}^*$ can compute $(0,k)$'s response following a procedure identical to that of $\mathcal{RM}^h_6$.

• Suppose now that $InMsg_3$ was not computed by a user instance. Then $\mathcal{M}^*$ decrypts $InMsg_3$ to get plaintext w. If w is not of the form $(1, U, V, n, pw, OID_{0k}, CID_{0k})$, for $pw \in D$, $(0,k)$ rejects. We suppose that w is of correct format.

• If $CID_{0k} = ID$ is not assigned to a user, then a "set password" operation was made on input ID and $\mathcal{M}^*$ knows pw_{ID}. pw is compared to pw_{ID}. If they match, $InMsg_3$ is accepted and the protocol continues. If not $InMsg_3$ is rejected.

• Suppose $CID_{0k} = ID_{i'}$ for some initialized i'.

•• Suppose $pw_{i'}$ is already known to $\mathcal{M}^*$ (i.e., i' has been the target of either a successful password guess as defined just below or a "reveal password" query). Then $\mathcal{M}^*$ can already compare pw and $pw_{i'}$ to determine the response.

•• Suppose $pw_{i'}$ is not known to $\mathcal{M}^*$. In this case, $\mathcal{M}^*$ makes a "test instance password" operation on input $\big(i', (0,k), pw\big)$, and computes $(0,k)$'s response based on $\mathcal{RM}^*$'s response.

Delivering the fifth protocol message and computing the sixth Let $InMsg_5$ be delivered to $(0,k)$ by $\mathcal{M}$.

• If $InMsg_5$ was computed by a user instance, $\mathcal{M}^*$ computes $(0,k)$'s response as $\mathcal{RM}^h_6$.

• If $InMsg_5$ was not computed by a user instance, $\mathcal{M}^*$ decrypts $InMsg_5$ to get plaintext w. We may suppose that w is of correct format $(2, U, V, n, pw, OID_{0k}, CID_{0k})$.

•• If $OID_{0k} = ID$ was not assigned to a user, $\mathcal{M}^*$ compares pw and pw_{ID}.

•• If $OID_{0k} = ID_i$ for some i:

••• If pw_i is already known to $\mathcal{M}^*$, $\mathcal{M}^*$ compares pw and pw_i.

••• If pw_i is not known to $\mathcal{M}^*$, $\mathcal{M}^*$ makes a "test instance password" operation request on input the string $\big(i, (0,k), pw\big)$.

Coin collection Once $\mathcal{M}$ has finished interacting with $\mathcal{M}^*$, $\mathcal{M}^*$ makes one final "implementation" operation to place all of $\mathcal{M}$'s random coins in the transcript.

This completes the description of how $\mathcal{M}^*$ is constructed from $\mathcal{M}$ and how $\mathcal{M}^*$ interacts with $\mathcal{RM}^*$.

Proposition 14 *$\mathcal{M}^*$ (as constructed above) faithfully follows the rules of the ideal world in the password-and-state-adaptive network adversary model, and the transcript random variables $\mathcal{HW}_6(\mathcal{M})$ and $\mathcal{IW}(\mathcal{M}^*)$ are identically distributed.*

Proof: For the first statement, all of the ideal-world operations that $\mathcal{M}^*$ makes are legit in the sense that all of the ideal-world conditions required for them to be made are satisfied. Next, Connection assignments are a function of the transcript up to the current "start session" operation because all connection assignments are determined by conditions on message deliveries, and across connection assignments these conditions are mutually exclusive.

The second part of the statement is obvious for most operations: $\mathcal{M}^*$ essentially just "runs" $\mathcal{RM}_6^h$. But two points need to be examined: **a).** that session key assignments are identically distributed in both worlds in the case of correct exchanges and **b).** that server instance responses are identically distributed in both worlds in the case of delivery of messages that are not generated by users.

Point **a).** is true because in both worlds originators that are assigned session keys when they have participated in a correct exchange received uniformly distributed keys. The only difference is that in the sixth hybrid world, $\mathcal{RM}_6^h$ chooses these keys while in the ideal world, $\mathcal{RM}^*$ chooses them. This difference does not change $\mathcal{M}$'s view however since in both worlds, $\mathcal{M}$ will receive a session key if and only if it makes a "reveal session key" query. Until this happens, the session keys have no effect on the adversary's behavior.

As for point **b).**, it is true because in both interactions - between $\mathcal{M}$ and $\mathcal{RM}_6^h$ in the sixth hybrid and between $\mathcal{M}$ and $\mathcal{M}^*$ in the ideal world - $\mathcal{M}$ has the messages it concocts itself accepted if and only if they decrypt to the correct password. How this is checked - directly by $\mathcal{RM}_6^h$, who has access to all passwords, or via "test instance password" queries by $\mathcal{M}^*$, who only has access to known passwords - is irrelevant since this check is outside of $\mathcal{M}$'s view.

This completes the proof of the proposition and therefore of theorem 2. ■

10 Honest-but-Curious Servers

In this section, we provide a framework capturing the security of 3-PAKE[spk]s against honest-but-curious servers. We leave it to the reader to show that **Prot1** and **Prot2** are secure in this sense. The proofs are considerably less involved than those above since the server does not deviate from the protocol and does not interfere with the network traffic. Thus, there are no complications due to confirmation codes or instance corruptions. We do not even have to worry about dictionary attacks, since the server has all of the passwords.

10.1 The Ideal World

We begin by a description of ideal-world adversarial behavior.

Initialize user, i, ID_i

$\mathcal{T}^*$ chooses an unused identity ID_i with which to initialize i. A password pw_i is chosen uniformly at random by $\mathcal{RM}^*$ and handed to $\mathcal{T}^*$.

Transcript: $(\text{"}initialize\ user\text{"}, i, ID_i)$

$\mathcal{T}^*$ uses this to bring into play new honest users.

Initialize user instance, (i, j), PID_{ij}

$\mathcal{T}^*$ asks to have a previously initialized user i initialize an instance (i, j) of that user. $\mathcal{T}^*$ assigns to (i, j) a partner identity PID_{ij} which is here required to be the identity of another initialized user.

Transcript: $\big(\text{"}initialize\ user\ instance\text{"}, (i, j), PID_{ij}\big)$

It is no longer necessary to attribute roles to the users essentially because all executions will be honestly carried out.

Start session, (i, j), (i', j')

$\mathcal{T}^*$ specifies a pair of previously initialized user instances (i, j) and (i', j') with $PID_{ij} = ID_{i'}$, $PID_{i'j'} = ID_i$, and indicates that a session key should be constructed for them. Neither should have been already participating in a session. $\mathcal{RM}^*$ chooses $SK^*_{ij} = SK^*_{i'j'}$ uniformly at random.

Transcript: $\big(\text{"}\mathit{start\ session}\text{"}, (i, j), (i', j')\big)$

Recall that $\mathcal{T}^*$ is passive. Accordingly, there can be no exposing.

Reveal session key, (i, j)**,** (i', j')

$\mathcal{T}^*$ specifies previously initialized user instances (i, j) and (i', j') that hold a common session key and receives that session key in return.

Transcript: $\big(\text{"}\mathit{reveal\ session\ key}\text{"}, (i, j), (i', j'), SK^*_{ij}\big)$

Implementation, *string*

Transcript: $(\text{"}\mathit{implementation}\text{"}, \mathit{string})$

10.2 The Real World

Here are an eavesdropping server's real capabilities.

The execution begins with $\mathcal{T}$ correctly generating its long term keying material $(pk_{\mathcal{T}}, sk_{\mathcal{T}})$. The public key $pk_{\mathcal{T}}$ will be made available to users.

Initialize user, i**,** ID_i

$\mathcal{T}$ chooses an identity ID_i with which to initialize i. ID_i should not have been used before for another user. A password pw_i is chosen, assigned to i, and handed $\mathcal{T}$. The user gains access to $pk_{\mathcal{T}}$.

Transcript: $(\text{"}\mathit{initialize\ user}\text{"}, i, ID_i)$

Initialize user instance, (i, j)**,** PID_{ij}

$\mathcal{T}$ asks to have a previously initialized user i initialize an instance j of that user. $\mathcal{T}$ assigns to (i, j) a partner identity PID_{ij}, previously used to initialize another user. The instance has access to $pk_{\mathcal{T}}$.

Transcript: $\big(\text{"}\mathit{initialize\ user\ instance}\text{"}, (i, j), PID_{ij}\big)$

Execute protocol, (i, j)**,** (i', j')**,** $MsgSeq$

For this operation to take place, user instances (i, j) and (i', j') must be initialized, $PID_{ij} = ID_{i'}$, $PID_{i'j'} = ID_i$, and neither instance should have been the argument of an "Execute Protocol" operation. The computations of an honest protocol execution between both instances and $\mathcal{T}$ take place, yielding in particular a sequence of messages $MsgSeq$ that $\mathcal{T}$ receives. At the end, both instances hold the same session key $SK_{ij} = SK_{i'j'}$, as computed by specification.

Transcript: $\big(\text{"}\mathit{implementation}\text{"}, \text{"}\mathit{execute\ protocol}\text{"}, (i, j), (i', j')\big)$ followed by
$\big(\text{"}\mathit{start\ session}\text{"}, (i, j), (i', j')\big)$

This operation allows $\mathcal{T}$ to observe as many honest protocol runs from the point of view of the server as it wishes.

Reveal session key, (i, j)**,** (i', j')

$\mathcal{T}$ specifies previously initialized user instances (i, j) and (i', j') that have computed a common session key together. It receives in return that session key.

$\big(\text{"}\mathit{reveal\ session\ key}\text{"}, (i, j), (i', j'), SK_{ij}\big)$

This operation captures potential leakage of session key information to the adversary via higher-level session key use. It is meaningful to have this because $\mathcal{T}$ is assumed honest-but-curious ***only within the context of the key exchange protocol****. No assumptions are made on its behavior if it interferes with* *other protocols. The operation is also technically useful to define security: after all, we need a mechanism to measure the server's ability to distinguish real keys from random strings.*

When $\mathcal{T}$ stops its interaction with $\mathcal{RM}$, the last entry in the transcript is
$(''implementation'', ''adversary\ coins'', coins)$
where *coins* holds all random values chosen by $\mathcal{T}$.

11 Conclusions and Future Work

We made a (theoretical and practical) case for considering 3-PAKE[spk]s. We proposed a hierarchy of security models to accommodate them, and exhibited several protocols that demonstrate the strictness of (parts of) this hierarchy. In the process, we were able to make what we believe to be a sensible definition of security against internal state revealing that does not trivially break password security.

Several directions should be explored from here. First, the models could be extended to accommodate arbitrary password distributions. It has already been argued for instance in [8] and [12] that simulation-based security models are well suited to do this.

Second, the models can be enhanced to capture server forward secrecy, in the sense that compromise of the server's long-term secret key should not damage past-established session keys. It is highly likely that **Prot1** and **Prot2** can be proven secure in this sense, at least using a weak definition of forward secrecy (i.e., one that assumes non-interference of the adversary in case of a corruption in the middle of a protocol run).

Third, it may be possible to consider queries that reveal *erased state* by demanding that a subset of random bits remains hidden. Admittedly, practical scenarios which would lead to this kind of attack may be somewhat rare (although one can imagine using an insecure pseudo-random generator for exponents and a secure one for encryption), but it is still worth exploring in an effort to continue refining provable security for key exchange. Of course, this result is of interest only if a provably secure protocol comes with it.

Finally, working in a simulation-style framework naturally leads to extending these results to the setting of universal composability [11].

Acknowledgments The author wishes to thank Dalia Khader, Peter Y. A. Ryan, Qiang Tang, and Jeroen van de Graaf for fruitful discussions on this topic.

References

[1] Abdalla, M., Chevalier, C., Pointcheval, D., *Smooth Projective Hashing for Conditionally Extractable Commitments* Crypto 2009, LNCS 5677, pp. 672-689, Springer, 2009

[2] Abdalla, M., Fouque, P.-A., Pointcheval, D., *Password-Based Authenticated Key Exchange in the Three-Party Setting*, PKC 2005, LNCS 3386, pp. 65-84, Springer, 2005

[3] Bellare, M., Canetti, R., Krawczyk, H., *A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols*, STOC 1998, pp. 419-428, 1998

[4] Bellare, M., Pointcheval, D., Rogaway, P., *Authenticated Key Exchange Secure Against Dictionary Attacks*, Adv. in Cryptology, Eurocrypt 2000, LNCS 1807, pp. 139-155, Springer, 2000

[5] Bellare, M., Rogaway, P., *Entity Authentication and Key Distribution*, Adv. in Cryptology, Crypto 1993, LNCS 773, Springer-Verlag, pp. 232-249, 1994

[6] Bellovin, S., Merrit, M., *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*, Proceedings of the IEEE Symposium on Research in Security and Privacy, 1992

[7] Boyarsky, M., *Public-Key Cryptography and Password Protocols: the Multi-User Case*, 6th ACM Conf. on Computer and Communications Security (CCS), pp. 63-72, 1999

[8] Boyko, V., MacKenzie, P., Patel, S., *Provably-Secure Password-Authenticated Key Exchange Using Diffie-Hellman*, Eurocrypt 2000, LNCS 1807, Springer-Verlag, pp. 156-171, 2000

[9] Byun, J., Jeong, I., Hoon Lee, D., Park, C., *Password-Authenticated Key Exchange Between Clients with Different Passwords*, ICICS 2002, LNCS 2513, pp. 134-146. Springer, 2002

[10] Camenisch, J., Casati, N., Gross, T., Shoup, V., *Credential Authenticated Identification and Key Exchange*, Crypto 2010, LNCS 6223, pp. 255-276, Springer, 2010

[11] Canetti, R., *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, 42nd FOCS, IEEE, pp. 136-145, 2001

[12] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P., *Universally Composable Password-Based Key Exchange*, Eurocrypt 2005, LNCS 3494, pp. 404-421, 2005

[13] Canetti, R., Krawczyk, H., *Analysis of Key Exchange Protocols and Their Use for Building Secure Channels*, Eurocrypt 2001, LNCS 2045, pp. 453-474, Springer, 2001

[14] Cramer, R., Shoup, V., *A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen-Ciphertext Attack*, Crypto 1998, LNCS 1462, pp. 13-25, Springer, 1998

[15] Cramer, R., Shoup, V., *Universal Hash Proofs and a Paradigm for Adaptive Chosen-Ciphertext Secure Public-key Encryption*, Eurocrypt 2002, LNCS 2332, pp. 45-64, Springer, 2001

[16] Cliff, Y., Tin, Y.S.T, Boyd, C., *Password Based Server Aided Key Exchange*, ACNS 2006, LNCS 3989, pp. 146-161, Springer, 2006

[17] Gennaro, R., *Faster and Shorter Password-Authenticated Key Exchange*, ACM Conference on Computer and Communications Security, 2008

[18] Gennaro, R., Lindell, Y., *A Framework for Password-Based Authenticated Key Exchange*, Eurocrypt 2003, pp. 524-543, 2003

[19] Goldreich, O., Lindell, Y., *Session-key Generation Using Human Passwords Only*, J. Cryptology, 19(3), pp. 241-340, 2006

[20] Goyal, V., Jain, A., Ostrovsky, R., *Password-Authenticated Session Key Generation on the Internet in the Plain Model*, Crypto 2010, LNCS 6223, pp. 277-294, Springer, 2010

[21] Groce, A., Katz, J., *A New Framework for Efficient Password-Based Authenticated Key Exchange*, 17th ACM Conf. on Computer and Communications Security (CCS), pp. 516-525, ACM Press, 2010

[22] Halevi, S., Krawczyk, H., *Public-Key Cryptography and Password Protocols*, 5th ACM Conf. on Computer and Communications Security (CCS), pp. 122-131, 1998

[23] Hao, F., Ryan, P. Y. A., *Password Authenticated Key Exchange by Juggling*, Security Protocols Workshop, 2008, pp. 159-171, 2008

[24] Hitchcock, Y., Tin, Y.S.T., Boyd, C., González Nieto, J.M., Montague, P., *A Password-Based Authenticator: Security Proofs and Applications*, Indocrypt 2003, LNCS 2904, Springer, 2003

[25] Jablon, D., *Strong Password-Only Authenticated Key Exchange*, ACM Computer Communications Review, Vol. 26, No. 5, pp. 5-26, 1996

[26] Jiang, S., Gong, G., *Password-Based Key Exchange with Mutual Authentication*, 11th Annual International Workshop on Selected Areas in Cryptography (SAC), LNCS 3357, pp. 267-279, Springer, 2004

[27] Katz, J., Ostrovsky, R., Yung, M., *Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords*, Eurocrypt 2001, LNCS 2045, Springer, pp. 475-494, 2001

[28] Katz, J., Vaikuntanathan, V., *Smooth Projective Hashing and Password-Based Authenticated Key Exchange from Lattices*, Asiacrypt 2009, LNCS 5912, pp. 636-652, Springer, 2009

[29] Katz, J., Vaikuntanathan, V., *Round-Optimal Password-Based Authenticated Key Exchange*, Theory of Cryptography, LNCS 6597, Springer, pp. 293-310, 2011

[30] Lancrenon, J., *What Public Keys Can Do for Three-Party, Password-Authenticated Key Exchange*, EuroPKI 2013, LNCS 8341, Springer, pp. 83-101, 2014

[31] Lucks, S., *Open Key Exchange: How to Defeat Dictionary Attacks without Encrypting Public Keys*, Security Protocols Workshop 1997, LNCS 1361, Springer, pp. 79-90, 1997

[32] LaMacchia, B., Lauter, K., Mityagin, A., *Stronger Security of Authenticated Key Exchange*, ProvSec, LNCS 4784, Springer, pp. 1-16, 2007

[33] Lin, C.-L., Sun, H.-M., Hwang, T., *Three-Party Encrypted Key Exchange: Attacks and a Solution*, ACM Operating Systems Review, vol. 34, no. 4, pp. 12-20, 2000

[34] Lin, C.-L., Sun, H.-M., Steiner, M., Hwang, T., *Three-party Encrypted Key Exchange Without Server Public-Keys*, IEEE Communications Letters, 5(12), pp. 497-499, 2001

[35] Shoup, V., *On Formal Models for Secure Key Exchange*, IBM Research Report RZ 3120, April 1999

[36] Steiner, M., Tsudik, G., Waidner, M., *Refinement and Extension of Encrypted Key Exchange*, ACM Operating Systems Review, 29(3), pp. 22-30, 1995

[37] Wang, F., Zhang, Y., *A New Security Model for Cross-Realm C2C-PAKE Protocol*, IACR e-print archive, 2007

[38] Wu, S., Zhu, Y., *Client-to-client Password-Based Authenticated Key Establishment in a Cross-Realm Setting*, Journal of Networks, vol. 4, n.7, 2009

[39] Yeh, H.-T., Sun, H.-M., Hwang, T., *Efficient Three-Party Authentication and Key Agreement Protocols Resistant to Password Guessing Attacks*, Journal of Information Science and Engineering, 2003, 19(6):1059-1070, 2003

[40] Yoneyama, K., *Efficient and Strongly Secure Password-Based Server Aided Key Exchange*, Indocrypt 2008, LNCS 5365, pp. 172-184, Springer, 2008

A Computational Assumptions and Properties

A.1 The Decisional Diffie-Hellman Assumption and Entropy Smoothing

The basic case Let G be a group of prime order q and g be a generator of G. The *Decisional Diffie-Hellman* (DDH) assumption states that

$$(g, g^x, g^y, g^{xy}) \text{ and } (g, g^x, g^y, g^z)$$

are computationally indistinguishable when x, y, and z are sampled independently and uniformly at random from $\mathbb{Z}_q^*$. Combining DDH with the *Entropy Smoothing* (ES) property of the hash family $\{H_n\}_n$ (the H_n mapping into $\{0,1\}^\ell$, for long enough ℓ), the DDH assumption becomes the statement that

$$\big(g, g^x, g^y, n, H_n(g^{xy})\big) \text{ and } (g, g^x, g^y, n, R)$$

are computationally indistinguishable when x, y, n, and R are sampled independently and uniformly at random. We shall simply call this the *Decisional Diffie-Hellman and Entropy Smoothing* (DDHES) property.

Remark: As is common in papers that rely on the DDH, we make the (outrageous) abuse of notation that consists of stating the assumption using a fixed group. In truth, we should be working with a *family of distributions of groups* indexed by the security parameter, since the assumption only makes sense asymptotically. We leave it up to the reader to collect the details. The same could also be said of the hash family.

Adding adaptive queries For the proofs of security of our protocols, we require an adaptive version of the DDHES property, best-described using algorithmic attack games $\mathbb{G}_b^{ad-DDHES}$ for $b \in \{0,1\}$. Let $\mathcal{B}$ be a probabilistic, polynomial-time algorithm. Game $\mathbb{G}_b^{ad-DDHES}$ is run between a challenger $\mathcal{CH}$ and $\mathcal{B}$ as follows.

On input security parameter 1^η, $\mathcal{CH}$ begins by constructing (q, G, g) and $\{H_n\}_n$, and gives this data to $\mathcal{B}$. It also initializes the sets $\mathcal{CE}$ and $\mathcal{CHI}$, which start out empty. Next, $\mathcal{B}$ may make a sequence of the following queries:

- *Left group element query:* $\mathcal{CH}$ selects $x \in \mathbb{Z}_q^* - \mathcal{CE}$ uniformly at random, computes $X \leftarrow g^x$, adds x to $\mathcal{CE}$, and gives (L, X) to $\mathcal{B}$;
- *Right group element query:* $\mathcal{CH}$ selects $y \in \mathbb{Z}_q^* - \mathcal{CE}$ uniformly at random, computes $Y \leftarrow g^y$, adds y to $\mathcal{CE}$, and gives (R, Y) to $\mathcal{B}$;
- *Hash index query:* $\mathcal{CH}$ selects $n \in \mathcal{K}_H - \mathcal{CHI}$ uniformly at random, adds n to $\mathcal{CHI}$, and gives n to $\mathcal{B}$;
- *Left exponent query:* $\mathcal{B}$ specifies (L, X) previously obtained through a "left group element" query and not previously input to a "challenge" query (see below), and $\mathcal{CH}$ gives x to $\mathcal{B}$;
- *Right exponent query:* $\mathcal{B}$ specifies (R, Y) previously obtained through a "right group element" query and not previously input to a "challenge" query, and $\mathcal{CH}$ gives y to $\mathcal{B}$;
- *Challenge query:* $\mathcal{B}$ specifies $\big((L, X), (R, Y), n\big)$ where (L, X) (resp., (R, Y), n) was obtained through a "left group element" (resp., "right group element", "hash index") query, neither X nor Y have had their exponents queried, and the pair $\big((L, X), (R, Y)\big)$ was not previously input to a "challenge" query. If $b = 0$, $\mathcal{CH}$ selects R uniformly at random from $\{0,1\}^\ell$ and gives R to $\mathcal{B}$. If $b = 1$, $\mathcal{CH}$ computes $H_n(g^{xy})$ and gives $H_n(g^{xy})$ to $\mathcal{B}$.

When $\mathcal{B}$ is done making queries, it outputs a single bit $\hat{b}$, and the game ends, the total output being $\hat{b}$. We write this as: $\mathcal{B}\big(\mathbb{G}_b^{ad-DDHES}(1^\eta)\big) = \hat{b}$.

A standard hybrid argument (slightly modified to accommodate the fact that we ask the challenger to choose pairwise-distinct exponents and hash indexes) shows that under the DDH assumption, for every probabilistic, polynomial-time algorithm $\mathcal{B}$, the expression

$$\left|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ad-DDHES}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ad-DDHES}(1^\eta)\big) = 1\Big]\right|$$

is negligible in η.

A.2 Hard-to-Compute Suffixes of Hashes of Diffie-Hellman Group Elements

The basic case Let us keep the notations of the previous section, except now we assume that the H_n functions map into $\{0,1\}^{\ell_1+\ell_2}$ for long enough ℓ_1 and ℓ_2. We are interested in the computability of the ℓ_2-bit-long suffix of $H_n(g^{xy})$, simply denoted $sf_{\ell_2}\big(H_n(g^{xy})\big)$. It can be proven that if the DDHES property holds, then for every probabilistic, polynomial-time algorithm $\mathcal{B}$ the expression

$$\mathbb{P}\Big[\mathcal{B}(g, g^x, g^y, n) = sf_{\ell_2}\big(H_n(g^{xy})\big)\Big]$$

is negligible in η. We call this the *Computational Diffie-Hellman and Entropy Smoothing* (CDHES) property.

Adding adaptive queries For the part of the proof of security of **Prot1** dealing with computationally lonely responder instances, we need an adaptive version of this basic case (again with pairwise-distinct exponents and hash indexes). Let $\mathcal{B}$ be a probabilistic, polynomial-time adversary, and consider the following game $\mathbb{G}^{ad-sf_{\ell_2}-CDHES}$ played by $\mathcal{B}$ against a challenger $\mathcal{CH}$.

On input security parameter 1^η, $\mathcal{CH}$ constructs (q, G, g) and $\{H_n\}_n$, and hands these to $\mathcal{B}$. It also initializes sets $\mathcal{CE}$, $\mathcal{CHI}$ and $\mathcal{OUT}$ which start out empty. $\mathcal{B}$ may now make a series of queries, as specified below:

- *Left/right group element query:* as in the previous section;
- *Hash index query:* as in the previous section;
- *Left exponent query:* $\mathcal{B}$ specifies (L, X) previously obtained through a "left group element" query and $\mathcal{CH}$ gives x to $\mathcal{B}$;
- *Right exponent query:* $\mathcal{B}$ specifies (R, Y) previously obtained through a "right group element" query and $\mathcal{CH}$ gives y to $\mathcal{B}$;
- *Hashed Diffie-Hellman query:* $\mathcal{B}$ specifies $\big((L, X), (R, Y), n\big)$ where (L, X) (resp., (R, Y), n) was obtained through a "left group element" (resp., "right group element", "hash index") query. $\mathcal{CH}$ responds with the full string $H_n(g^{xy})$;
- *Guess hash suffix query:* $\mathcal{B}$ specifies $\big((L, X), (R, Y), n, s\big)$ where (L, X) (resp., (R, Y), n) was obtained through a "left group element" (resp., "right group element", "hash index") query, $s \in \{0, 1\}^{\ell_2}$, neither X nor Y's exponent has been queried and the pair $\big((L, X), (R, Y)\big)$ has not been the input to a "hashed Diffie-Hellman" query. The tuple $\big((L, X), (R, Y), n, s\big)$ is added to the list $\mathcal{OUT}$.

Eventually, the game ends, and the list $\mathcal{OUT}$ is examined. If there is a tuple $\big((L, X), (R, Y), n, s\big)$ such that $s = sf_{\ell_2}\big(H_n(g^x y)\big)$, the game's output is 1. Otherwise, it is 0.

By using a tedious hybrid argument, it can be shown that assuming the DDHES property, we have that for every adversary $\mathcal{B}$, the expression

$$\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-sf_{\ell_2}-CDHES}(1^\eta)\big) = 1\Big]$$

is negligible in η.

A.3 Hard-to-Guess Random Group Elements and Hard-to-Guess Suffixes of Hashes of Hidden, Random Group Elements

The basic case Keeping the notations of the previous paragraph, we are now interested in the computability of the ℓ_2-long suffix of hashes of the form $H_n(U^z)$ for any non-trivial group element U, any uniformly random exponent z, and any uniformly random hash index n. Let $\mathcal{B}$ be a probabilistic, polynomial-time algorithm and consider the following game, denoted $\mathcal{G}^{sf_{\ell_2}-ES}$, played against a challenger $\mathcal{CH}$.

On input security parameter 1^η, $\mathcal{CH}$ constructs the (q, G, g) and the hash family $\{H_n\}_n$, and hands this over to $\mathcal{B}$. $\mathcal{CH}$ then chooses a hash index n and an exponent $z \in \mathbb{Z}_q^*$ uniformly at random. n is given to $\mathcal{B}$, and z is kept out of $\mathcal{B}$'s view. $\mathcal{B}$ then outputs a non-trivial group element U and a string $s \in \{0, 1\}^{\ell_2}$ to $\mathcal{CH}$ and halts. If $s = sf_{\ell_2}\big(H_n(U^z)\big)$, the total output of the game is 1, and otherwise it is 0. This is denoted $\mathcal{B}\big(\mathbb{G}^{sf_{\ell_2}-ES}(1^\eta)\big) \in \{0, 1\}$.

Using only the entropy smoothing property of the hash family (and appropriate group parameters), we can show that the quantity

$$\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{sf_{\ell_2}-ES}(1^\eta)\big) = 1\Big]$$

is negligible in η.

Adding adaptive queries In the part of the proof of security of **Prot1** dealing with statistically lonely originators and responders, we actually need an adaptive version of the above basic game that also integrates opportunities for the adversary to guess hidden group elements. We denote this new game $\mathbb{G}^{ad-GE-sf_{\ell_2}-ES}$. We keep the notations above. Let $\mathcal{B}$ be a probabilistic, polynomial-time adversary.

At the beginning of the game, $\mathcal{CH}$, on input 1^η, constructs (q, G, g) and $\{H_n\}_n$ and gives these to $\mathcal{B}$. It also initializes the sets $\mathcal{CE}$, $\mathcal{CHI}$, and $\mathcal{OUT}$. A counter $C \leftarrow 0$ is also initialized. $\mathcal{B}$ may then make any number of the following queries:

• *Prepare exponent query:* C is incremented, $\mathcal{CH}$ chooses z_C uniformly at random from $\mathbb{Z}_q^* - \mathcal{CE}$, and z_C is added to $\mathcal{CE}$;

• *Recover exponent query:* $\mathcal{B}$ submits $k \in \{0, ..., C\}$ to $\mathcal{CH}$ and gets z_k back in return;

• *Hash index query:* $\mathcal{CH}$ selects n uniformly at random from $\mathcal{K}_H - \mathcal{CHI}$, gives n to $\mathcal{B}$, and adds n to $\mathcal{CHI}$;

• *Guess group element query:* $\mathcal{B}$ submits (U, k) to $\mathcal{CH}$, where $k \in \{0, ..., C\}$ was not the input to a "recover exponent" query. If $U = g^{z_k}$, the game ends and the total output of the game is 1. Otherwise the game continues;

• *Guess hash suffix query:* $\mathcal{B}$ submits (U, n, k, s) to $\mathcal{CH}$, where U is a non-trivial group element, $k \in \{0, ..., C\}$ was not the input to a "recover exponent" query, n was obtained through a "hash index" query, and $s \in \{0,1\}^{\ell_2}$. The tuple (U, n, k, s) is added to the list $\mathcal{OUT}$.

Eventually, the game ends, and the list $\mathcal{OUT}$ is examined. If there is a tuple (U, n, k, s) such that $s = sf_{\ell_2}\big(H_n(U^z)\big)$, the game's output is 1. Otherwise, it is 0.

Using a hybrid argument with the non-adaptive basic game described above, we can show that the expression

$$\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{ad-sf_{\ell_2}-ES}(1^\eta)\big) = 1\Big]$$

is negligible in η.

B Security Notions for Primitives

B.1 IND-CCA-2-secure Encryption

Let $Enc := (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme.

The single-challenge case Consider the following two attack games $\mathbb{G}_b^{IND-CCA-2}$, where $b \in \{0,1\}$, played between a challenger $\mathcal{CH}$ and a probabilistic, polynomial-time adversary $\mathcal{B}$.

At the beginning of either game, on input 1^η challenger $\mathcal{CH}$ runs $\mathcal{K}(1^\eta)$ to get an encryption/decryption key pair (pk, sk) for Enc, and pk is given to $\mathcal{B}$. The games then proceed in four phases:

• *First query phase:* $\mathcal{B}$ may make "decryption" queries to $\mathcal{CH}$, wherein $\mathcal{B}$ submits a string $\mathfrak{c}$ to $\mathcal{CH}$ and $\mathcal{CH}$ returns $\mathcal{D}_{sk}(\mathfrak{c})$ back to $\mathcal{B}$;

• *Challenge phase:* $\mathcal{B}$ ***may*** submit a pair (m_0, m_1) of equal-length messages to $\mathcal{CH}$, and receives in return an encryption $c \leftarrow \mathcal{E}_{pk}(m_b)$;

• *Second query phase:* $\mathcal{B}$ may make more "decryption" queries to $\mathcal{CH}$, subject to the restriction that the challenge c itself may not be submitted;

• *Guess phase:* $\mathcal{B}$ outputs a bit $\hat{b}$. The game ends and the final output of the game is $\hat{b}$.

Enc is said to be an *adaptive, chosen-ciphertext-secure* (IND-CCA-2-secure) public-key encryption scheme if for every PPTA $\mathcal{B}$, the expression

$$\left|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{IND-CCA-2}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{IND-CCA-2}(1^\eta)\big) = 1\Big]\right|$$

is negligible in η.

Multiple adaptive challenges For the proofs of security of **Prot1** and **Prot2**, we need the following challenge-adaptive versions of the above game. Let $\mathcal{R}$ be the randomness space of the encryption algorithm $\mathcal{E}$. Below we make the randomness explicit in the notation, i.e. the encryption $\mathcal{E}_{pk}(m)$ of message m will be denoted $\mathcal{E}_{pk}(m; r)$ for $r \in \mathcal{R}$. The games $\mathbb{G}_b^{ad-IND-CCA-2}$ for $b \in \{0, 1\}$ are defined as follows:

At the beginning of either game, on input 1^η, $\mathcal{CH}$ runs $\mathcal{K}(1^\eta)$ to get (pk, sk) and outputs gives pk to $\mathcal{B}$. $\mathcal{CH}$ also initializes a set $\mathcal{CER}$ that starts out empty. $\mathcal{B}$ may then make any number of the following queries:

- *Challenge query:* $\mathcal{B}$ submits to $\mathcal{CH}$ a pair (m_0, m_1) of equal-length messages. $\mathcal{CH}$ samples r from $\mathcal{R} - \mathcal{CER}$, computes $c \leftarrow \mathcal{E}_{pk}(m_b; r)$, adds r to $\mathcal{CER}$, and gives c to $\mathcal{B}$;
- *Decryption query:* $\mathcal{B}$ submits to $\mathcal{CH}$ a string γ and $\mathcal{CH}$ responds with $\mathcal{D}_{sk}(\gamma)$. The string γ cannot have been previously obtained from $\mathcal{CH}$ as a "challenge" query.

Eventually, $\mathcal{B}$ stops the game and outputs a single bit $\hat{b}$. The total output of the game is $\hat{b}$.

Using a hybrid argument (slightly modified to accommodate the fact that the same encryption randomness cannot be used twice to answer challenge queries), it is possible to show that if *Enc* is IND-CCA-2-secure, then the expression

$$\left|\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_0^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big] - \mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}_1^{ch-ad-IND-CCA-2}(1^\eta)\big) = 1\Big]\right|$$

is negligible in η.

B.2 EU-ACMA-secure Signatures

Let $Sig := (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme. We consider the following attack game $\mathbb{G}^{EU-ACMA}$ played between challenger $\mathcal{CH}$ and a probabilistic, polynomial-time adversary $\mathcal{B}$.

At the beginning of the game, $\mathcal{CH}$ runs $\mathcal{K}(1^\eta)$ to get (pk, sk), and hands pk to $\mathcal{B}$. $\mathcal{B}$ may then make any number of the following queries:

- *Signature query:* $\mathcal{B}$ submits a message m to $\mathcal{CH}$, and $\mathcal{CH}$ returns to $\mathcal{B}$ a signature $\sigma \leftarrow \mathcal{S}_{sk}(m)$ on m;
- *Forgery query:* $\mathcal{B}$ submits to $\mathcal{CH}$ a pair (m, τ), where m was not previously submitted to $\mathcal{CH}$ in a "signature" query. If $\mathcal{V}_{pk}(m, \tau) = 1$, the game ends, with total output 1. Otherwise, the game continues.

Eventually, the game is halted. If it is halted without outputting 1, its total output is 0. We say that *Sig* is *Existentially Unforgeable Against Chosen-Message Attacks* (EU-ACMA-secure) if the expression

$$\mathbb{P}\Big[\mathcal{B}\big(\mathbb{G}^{EU-ACMA}(1^\eta)\big) = 1\Big]$$

is negligible in η.