Fuzzy Authenticated Key Exchange

Pierre-Alain Dupont[1,2,3], Julia Hesse[2,3], David Pointcheval[2,3], Leonid Reyzin[4], and Sophia Yakoubov[4]

[1] DGA
[2] DIENS, École Normale Supérieure, CNRS, PSL Research University, Paris, France
[3] INRIA
[4] Boston University

Abstract. Consider key agreement by two parties who start out knowing a common secret (which we refer to as "pass-string", a generalization of "password"), but face two complications: (1) the pass-string may come from a low-entropy distribution, and (2) the two parties' copies of the pass-string may have some noise, and thus not match exactly. We provide the first efficient and general solutions to this problem that enable, for example, key agreement based on commonly used biometrics such as iris scans.

The problem of key agreement with each of these complications individually has been well studied in literature. Key agreement from low-entropy shared pass-strings is achieved by *password-authenticated key exchange* (PAKE), and key agreement from noisy but high-entropy shared pass-strings is achieved by information-reconciliation protocols as long as the two secrets are "close enough." However, the problem of key agreement from noisy low-entropy pass-strings has never been studied.

We introduce (universally composable) *fuzzy password-authenticated key exchange* (fPAKE), which solves exactly this problem. fPAKE does not have any entropy requirements for the pass-strings, and enables secure key agreement as long as the two pass-strings are "close" for some notion of closeness. We also give two constructions. The first construction achieves our fPAKE definition for any (efficiently computable) notion of closeness, including those that could not be handled before even in the high-entropy setting. It uses Yao's Garbled Circuits in a way that is only two times more costly than their use against semi-honest adversaries, but that guarantees security against malicious adversaries. The second construction is more efficient, but achieves our fPAKE definition only for pass-strings with low Hamming distance. It builds on very simple primitives: robust secret sharing and PAKE.

Keywords: Authenticated Key Exchange, PAKE, Hamming Distance, Error Correcting Codes, Yao's Garbled Circuits

1 Introduction

Consider key agreement by two parties who start out knowing a common secret (which we refer to as "pass-string", a generalization of "password"). These parties may face several complications: (1) the pass-string may come from a non-uniform, low-entropy distribution, and (2) the two parties' copies of the pass-string may have some noise, and thus not match exactly. The use of such pass-strings for security has been extensively studied; examples include biometrics and other human-generated data [Dau04, ZH93, BS00, EHMS00, MG09, MRW02, KR08], physically unclonable functions (PUFs) [PRTG02, GCvD02, TSS+06, SD07, YD10], noisy channels [Wyn75], and quantum information [BBR88].

The Noiseless Case. When the starting secret is not noisy (i.e., the same for both parties), existing approaches work quite well. The case of low-entropy secrets is covered by *password-authenticated key exchange* (PAKE) (a long line of work starting with [BPR00, BMP00]). A PAKE protocol allows two parties to agree on a shared high-entropy key if and only if they hold the same short password. Even though the password may have low entropy, PAKE ensures that off-line dictionary attacks are impossible. Roughly speaking, an adversary has to participate in one on-line interaction for every attempted guess at the password. Because key agreement is not usually the final goal, PAKE protocols need to be composed with whatever protocols (such as authenticated encryption) use the output key. This composability has been achieved by universally composable (UC) PAKE defined by Canetti *et al.* [CHK+05] and implemented in several follow-up works.

In the case of high-entropy secrets, off-line dictionary attacks are not a concern, which enables more efficient protocols. If the adversary is passive, randomness extractors [NZ93] do the job. The case of active adversaries is covered by the literature on so-called robust extractors defined by Boyen *et al.* [BDK+05] and, more generally, by many papers on privacy amplification protocols secure against active adversaries, starting with the work of Maurer [Mau97]. Composability for these protocols is less studied; in particular, most protocols leak information about the pass-string itself, in which case reusing the pass-string over multiple protocol executions may present problems [Boy04] (with the exception of [CFP+16]).

The Noisy Case. When the pass-string is noisy (i.e., the two parties have slightly different versions of it), this problem has been studied only for the case of high-entropy pass-strings. A long series of works on information-reconciliation protocols started by Bennett *et al.* [BBR88] and their one-message variants called fuzzy extractors (defined by Dodis *et al.* [DORS08], further enhanced for active security starting by Renner *et al.* [RW04]) achieves key agreement when the pass-string has a lot of entropy and not too much noise. Unfortunately, these approaches do not extend to the low-entropy setting and are not designed to prevent off-line dictionary attacks.

Constructions for the noisy case depend on the specific noise model. The case of binary Hamming distance — when the n pass-string characters held by the two

parties are the same at all but δ locations — is the best studied. Most existing constructions require, at a minimum, that the pass-string should have at least δ bits of entropy. This requirement rules out using most kinds of biometric data as the pass-string— for example, estimates of entropy for iris scans (transformed into binary strings via wavelet transforms and projections) are considerably lower than the amount of errors that need to be tolerated [BH09, Section 5]. Even the PAKE-based construction of Boyen *et al.* [BDK+05] suffers from the same problem.

One notable exception is the construction of Canetti *et al.* [CFP+16], which does not have such a requirement, but places other stringent limitations on the probability distribution of pass-strings. In particular, because it is a one-message protocol, it cannot be secure against off-line dictionary attacks.

1.1 Our Contributions

We provide definitions and constant-round protocols for key agreement from noisy pass-strings that:

- Resist off-line dictionary attacks and thus can handle low-entropy pass-strings,
- Can handle a variety of noise types and have high error-tolerance, and
- Have well specified composition properties via the UC framework [Can01].

Instead of imposing entropy requirements or other requirements on the distribution of pass-strings, our protocols are secure as long as the adversary cannot guess a pass-string value that is sufficiently close. There is no requirement, for example, that the amount of pass-string entropy is greater than the number of errors; in fact, one of our protocols is suitable for iris scans. Moreover, our protocols prevent off-line attacks, so each adversarial attempt to get close to the correct pass-string requires an on-line interaction by the adversary. Thus, for example, our protocols can be meaningfully run with pass-strings whose entropy is only 30 bits—something not possible with any prior protocols for the noisy case.

New Models. Our security model is in the Universal Composability (UC) Framework of Canetti [Can01]. The advantage of this framework is that it comes with a composability theorem that ensures that the protocol stays secure even running in arbitrary environments, including arbitrary parallel executions. Composability is particularly important for key agreement protocols, because key agreement is rarely the ultimate goal. The agreed-upon key is typically used for some subsequent protocol—for example, a secure channel. Further, this framework allows to us to give a definition that is agnostic to how the initial pass-strings are generated. We have no entropy requirements or constraints on the pass-string distribution; rather, security is guaranteed as long as the adversary's input to the protocol is not close enough to the correct pass-string.

As a starting point, we use the definition of UC security for PAKE from Canetti *et al.* [CHK+05]. The PAKE ideal functionality is defined as follows: the

secret pass-strings (called "passwords" in PAKE) of the two parties are the inputs to the functionality, and two random keys, which are equal if and only if the two inputs are equal, are the outputs. The main change we make to PAKE is enhancing the functionality to give equal keys even if the two inputs are not equal, as long as they are close enough. We also relax the security requirement to allow one party to find out some information about the other party's input—perhaps even the entire input—if the two inputs are close. This relaxation makes sense in our application: if the two parties are honest, then the differences between their inputs are a problem rather than a feature, and we would not mind if the inputs were in fact the same. The benefit of this relaxation is that it permits us to construct more efficient protocols. (We also make a few other minor changes which will be described in Section 2.) We call our new UC functionality "Fuzzy Password-Authenticated Key Exchange" or fPAKE.

New Protocols. The only prior PAKE-based protocol for the noisy setting by Boyen *et al.* [BDK+05], although more efficient than ours, does not satisfy our goal. In particular, it is not composable, because it reveals information about the secret pass-strings (we actually demonstrate this formally in Appendix H). It also requires high-entropy pass-strings, because some information is unconditionally revealed in the protocol and reduces the secrecy of the inputs. Thus, we need to construct new protocols that realize our definition.

Realizing our fPAKE definition is easy using general two-party computation techniques for protocols with malicious adversaries and without authenticated channels [BCL+05]. However, we develop protocols that are considerably more efficient: our definitional relaxation allows us to build protocols that achieve security against malicious adversaries but cost just a little more than the protocols that achieve security only against honest-but-curious adversaries (i.e., adversaries who do not deviate from the protocol, but merely try to infer information they are not supposed to know).

Our first construction uses Yao's garbled circuits [Yao86, BHR12] and oblivious transfer (see [CO15] and references therein). The use of these techniques is standard in two-party computation. However, by themselves they give protocols secure only against honest-but-curious adversaries. In order to prevent malicious behavior of the players, one usually applies the cut-and-choose technique [LP11], which is quite costly: to achieve an error probability of $2^{-\lambda}$, the number of circuits that need to be garbled increases by a factor of λ, and the number of oblivious transfers that need to be performed increases by a factor of $\lambda/2$. We show that for our special case, to achieve malicious security, it suffices to repeat the honest-but-curious protocol twice (once in each direction), incurring only a factor of 2 overhead over the semi-honest case (plus small overhead due to the techniques of [BCL+05] for working with unauthenticated channels).[5] This construction works regardless of what it means for the two inputs to be "close," as long as the question of closeness can be evaluated by an efficient circuit.

[5] Gasti *et al.* [GSY+16] similarly use Yao's garbled circuits for continuous biometric user authentication on a smartphone. Our approach can eliminate the third party in their application, at the cost of requiring two garbled circuits instead of one. As far

Our second construction is for the Hamming case: the two n-character pass-strings have low Hamming distance if not too many characters of one party's pass-string are different from the corresponding characters of the other's pass-string. The two parties execute a PAKE protocol for each position in the string, obtaining n values each that agree or disagree depending on whether the characters of the pass-string agree or disagree in the corresponding positions. It is important that at this stage, agreement or disagreement at individual positions remains unknown to everyone; we therefore make use of a special variant of PAKE which we call *implicit-only PAKE* (we give a formal UC security definition of implicit-only PAKE and show that it is realized by the PAKE protocol from [BM92,ACCP08]). This first step upgrades Hamming distance over a potentially small alphabet to Hamming distance over an exponentially large alphabet. We then secret-share the ultimate output key into n shares using a robust secret sharing scheme, and encrypt each share using the output of the corresponding PAKE protocol.

The second construction is more efficient than the first in the number of rounds, communication, and computation. However, it works only for Hamming distance. Moreover, it has an intrinsic gap between functionality and security: if the honest parties need to be within distance δ to agree, then the adversary may break security by guessing a secret within distance 2δ. See Figure 10 for a comparison between the two constructions.

The advantages of our protocols are similar to the advantages of UC PAKE: They provide composability, protection against off-line attacks, the ability to use low-entropy inputs, and handle any distribution of secrets. And, of course, because we construct *fuzzy* PAKE, our protocols can handle noisy inputs—including many types of noisy inputs that could not be handled before. Our first protocol can handle any type of noise as long as the notion of "closeness" can be efficiently computed, whereas most prior work was for Hamming distance. However, these advantages come at the price of efficiency. Our protocols require 2-6 rounds of interaction, as opposed to many single-message protocols in the literature [DKK+12, CFP+16, WCD+17]. They are also more computationally demanding than most existing protocols for the noisy case, requiring one public-key operation per input character. We emphasize, however, that our protocols are much less computationally demanding than the protocols based on general two-party computation, as already discussed above, or general-purpose obfuscation, as discussed in [BCKP14, Section 4.3.4].

2 Security Model

We now present a security definition for fuzzy password-authenticated key exchange (fPAKE). We adapt the definition of PAKE from Canetti *et al.* [CHK+05] to work for pass-strings (a generalization of "passwords") that are similar, but not necessarily equal. Our definition uses measures of the distance $d(\mathsf{pw}, \mathsf{pw}')$

as we know, ours is the first use of garbled circuits in the two-party fully malicious setting without calling on an expensive transformation.

between pass-strings $\mathsf{pw}, \mathsf{pw}' \in \mathbb{F}_p^n$. In Section 3.3 and Section 4, Hamming distance is used, but in the generic construction of Section 3, any other notion of distance can be used instead. We say that pw and pw' are "similar enough" if $d(\mathsf{pw}, \mathsf{pw}') \leq \delta$ for a distance notion d and a threshold δ that is hard-coded into the functionality.

To model the possibility of dictionary attacks, the functionality allows the adversary to make one pass-string guess against each player ($\mathcal{P}_0$ or $\mathcal{P}_1$). In the real world, if the adversary succeeds in guessing (a pass-string similar enough to) party $\mathcal{P}_i$'s pass-string, it can often choose (or at least bias) the session key computed by $\mathcal{P}_i$. To model this, the functionality then allows the adversary to set the session key for $\mathcal{P}_i$.

As usual in security notions for key exchange, the adversary also sets the session keys for corrupted players. In the definition of Canetti *et al.* [CHK+05], the adversary additionally sets $\mathcal{P}_i$'s key if $\mathcal{P}_{1-i}$ is corrupted. However, contrarily to the original definition, we do not allow the adversary to set $\mathcal{P}_i$'s key if $\mathcal{P}_{1-i}$ is corrupted but did not guess $\mathcal{P}_i$'s pass-string. We make this change in order to protect an honest $\mathcal{P}_i$ from, for instance, revealing sensitive information to an adversary who did not successfully guess her pass-string, but did corrupt her partner.

Another minor change we make is considering only two parties — $\mathcal{P}_0$ and $\mathcal{P}_1$ — in the functionality, instead of considering arbitrarily many parties and enforcing that only two of them engage the functionality. This is because universal composability takes care of ensuring that a two-party functionality remains secure in a multi-party world.

As in the definition of Canetti *et al.* [CHK+05], we consider only static corruptions in the standard corruption model of Canetti [Can01]. Also as in their definition, we chose not to provide the players with confirmation that key agreement was successful. The players might obtain such confirmation from subsequent use of the key.

By default, in the fPAKE functionality the `TestPwd` interface provides the adversary with one bit of information — whether the pass-string guess was correct or not. This definition can be strengthened by providing the adversary with no information at all, as in implicit-only PAKE ($\mathcal{F}_{\mathsf{iPAKE}}$, Figure 7), or weakened by providing the adversary with extra information when the adversary's guess is close enough.

To capture the diversity of possibilities, we introduce a more general `TestPwd` interface, described in Figure 2. It includes three leakage functions that we will instantiate in different ways below—L_c if the guess is close-enough to succeed, L_f if it is too far. Moreover, a third leakage function—L_m for medium distance—allows the adversary to get some information even if the adversary's guess is only somewhat close (closer than some parameter $\gamma \geq \delta$), but not close enough for successful key agreement. We thus decouple the distance needed for functionality from the (possibly larger) distance needed to guarantee security; the smaller the gap between these two distances, the better, of course.

The functionality fPAKE is parameterized by a security parameter λ and tolerances $\delta \leq \gamma$. It interacts with an adversary $\mathcal{S}$ and two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i)$ from party $\mathcal{P}_i$:**
  - Send $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$;
  - If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$, then record $(\mathcal{P}_i, \mathsf{pw}_i)$ and mark this record `fresh`.
- **Upon receiving a query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}'_i)$ from the adversary $\mathcal{S}$:** If there is a `fresh` record $(\mathcal{P}_i, \mathsf{pw}_i)$, then set $d \leftarrow d(\mathsf{pw}_i, \mathsf{pw}'_i)$ and do:
  - If $d \leq \delta$, mark the record `compromised` and reply to $\mathcal{S}$ with "correct guess";
  - If $d > \delta$, mark the record `interrupted` and reply to $\mathcal{S}$ with "wrong guess".
- **Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{sk})$ from the adversary $\mathcal{S}$:** If there is no record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$, or if this is not the first `NewKey` query for $\mathcal{P}_i$, then ignore this query. Otherwise:
  - If at least one of the following is true, then output $(\mathsf{sid}, \mathsf{sk})$ to player $\mathcal{P}_i$:
    - The record is `compromised`
    - $\mathcal{P}_i$ is corrupted
    - The record is `fresh`, $\mathcal{P}_{1-i}$ is corrupted, and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) \leq \delta$
  - If this record is `fresh`, both parties are honest, there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) \leq \delta$, a key sk' was sent to $\mathcal{P}_{1-i}$, and $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was `fresh` at the time, then output $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$;
  - In any other case, pick a new random key sk' of length λ and send $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$.
  - Mark the record $(\mathcal{P}_i, \mathsf{pw}_i)$ as `completed`.

Fig. 1. Ideal Functionality fPAKE

Below, we list the specific leakage functions L_c, L_m and L_f that we consider in this work, in order of decreasing strength (or increasing leakage):

1. The strongest option is to provide no feedback at all to the adversary. We define fPAKE^N to be the functionality described in Figure 1, except that `TestPwd` is from Figure 2 with
$$L_c^N(\mathsf{pw}_i, \mathsf{pw}'_i) = L_m^N(\mathsf{pw}_i, \mathsf{pw}'_i) = L_f^N(\mathsf{pw}_i, \mathsf{pw}'_i) = \perp .$$
2. The basic functionality fPAKE, described in Figure 1, leaks the correctness of the adversary's guess. That is, in the language of Figure 2,
$$L_c(\mathsf{pw}_i, \mathsf{pw}'_i) = \text{"correct guess"},$$
$$\text{and} \qquad L_m(\mathsf{pw}_i, \mathsf{pw}'_i) = L_f(\mathsf{pw}_i, \mathsf{pw}'_i) = \text{"wrong guess"}.$$
The classical PAKE functionality from [CHK+05] has such a leakage.
3. Assume the two pass-strings are strings of length n over some finite alphabet, with the jth character of the string pw denoted by $\mathsf{pw}[j]$. We define fPAKE^M to be the functionality described in Figure 1, except that `TestPwd` is from

– **Upon receiving a query $(\texttt{TestPwd}, \mathbf{sid}, \mathcal{P}_i, \mathsf{pw}'_i)$ from the adversary $\mathcal{S}$:** If there is a `fresh` record $(\mathcal{P}_i, \mathsf{pw}_i)$, then set $d \leftarrow d(\mathsf{pw}_i, \mathsf{pw}'_i)$ and do:
  - If $d \leq \delta$, mark the record `compromised` and reply to $\mathcal{S}$ with $L_c(\mathsf{pw}_i, \mathsf{pw}'_i)$;
  - If $\delta < d \leq \gamma$, mark the record `compromised` and reply to $\mathcal{S}$ with $L_m(\mathsf{pw}_i, \mathsf{pw}'_i)$;
  - If $\gamma < d$, mark the record `interrupted` and reply to $\mathcal{S}$ with $L_f(\mathsf{pw}_i, \mathsf{pw}'_i)$.

Fig. 2. A Modified `TestPwd` Interface to Allow for Different Leakage

Figure 2, with L_c and L_m that leaks the indices at which the guessed pass-string differs from the actual one when the guess is close enough (we will call this leakage the *mask* of the pass-strings). That is,

$$L_c^M(\mathsf{pw}_i, \mathsf{pw}'_i) = (\{j \text{ s.t. } \mathsf{pw}_i[j] = \mathsf{pw}'_i[j]\}, \text{“correct guess”}),$$
$$L_m^M(\mathsf{pw}_i, \mathsf{pw}'_i) = (\{j \text{ s.t. } \mathsf{pw}_i[j] = \mathsf{pw}'_i[j]\}, \text{“wrong guess”})$$
$$\text{and} \quad L_f^M(\mathsf{pw}_i, \mathsf{pw}'_i) = \text{“wrong guess”}.$$

4. The weakest definition — or the strongest leakage — reveals the entire actual pass-string to the adversary if the pass-string guess is close enough. We define fPAKE^P to be the functionality described in Figure 1, except that `TestPwd` is from Figure 2, with

$$L_c^P(\mathsf{pw}_i, \mathsf{pw}'_i) = L_m^P(\mathsf{pw}_i, \mathsf{pw}'_i) = \mathsf{pw}_i \quad \text{and} \quad L_f^P(\mathsf{pw}_i, \mathsf{pw}'_i) = \text{“wrong guess”}.$$

   Here, L_c^P and L_m^P do not need to include “correct guess” and “wrong guess”, respectively, because this is information that can be easily derived from pw_i itself.

The first two functionalities are the strongest, but there are no known constructions that realize them, other than through generic two-party computation secure against malicious adversaries, which is an inefficient solution. In Section 3, we present a construction satisfying fPAKE^P for any efficiently computable notion of distance, with $\gamma = \delta$ (which is the best possible). We present a construction for Hamming distance satisfying fPAKE^M in Section 4, with $\gamma = 2\delta$.

3 General Construction Using Garbled Circuits

In this section, we describe a protocol realizing fPAKE^P that uses Yao's Garbled Circuits [Yao86]. We briefly introduce this primitive in Section 3.1 and refer to Yakoubov [Yak17] for a more thorough introduction.

This Yao's Garbled Circuit fPAKE construction has two advantages:

1. It is more flexible than other approaches; any notion of distance that can be efficiently computed by a circuit can be used. In Section 3.3, we describe a suitable circuit for Hamming distance. The total size of this circuit is $O(n)$, where n is the length of the pass-strings used. Edit distance is slightly less efficient, and uses a circuit whose total size is $O(n^2)$.

2. There is no gap between the distances required for functionality and security — that is, there is no leakage about the pass-strings used unless they are similar enough to agree on a key. In other words, $\delta = \gamma$.

Informally, the construction involves the garbled evaluation of a circuit that takes in two pass-strings as input, and computes whether their distance is less than δ. Because Yao's Garbled Circuits are only secure against semi-honest garblers, we cannot simply have one party do the garbling and the other party do the evaluation. A malicious garbler could provide a garbling of the wrong function — maybe even a constant function — which would result in successful key agreement even if the two pass-strings are very different. However, since a malicious evaluator (unlike a malicious garbler) cannot compromise the computation, if we perform the protocol twice with each party playing each role once, we can protect against cheating.

3.1 Preliminaries

3.1.1 Oblivious Transfer (OT) Informally, 1-out-of-2 Oblivious Transfer (see [CO15] and citations therein) enables one party (the sender) to transfer exactly one of two secrets to another party (the receiver). The receiver chooses (by index 0 or 1) which secret she wants. The security of the OT protocol guarantees that the sender does not learn this choice bit, and the receiver does not learn anything about the other secret.

3.1.2 Yao's Garbled Circuits (YGC) Next, we give a brief introduction to Yao's Garbled Circuits [Yao86]. Informally, Yao's Garbled Circuits (YGC) are an asymmetric secure two-party computation scheme. It enables two parties with sensitive inputs (in our case, pass-strings) to compute a joint function of their inputs (in our case, an augmented version of similarity) without revealing any additional information about their inputs. One party "garbles" the function they wish to evaluate, and the other evaluates it in its garbled form.

There have been many optimizations to YGC over the years [BMR90, KS08, PSSW09, KMR14, ZRE15, BMR16]. While YGC is naturally secure against a malicious evaluator, all known constructions share the drawback of being insecure against a malicious garbler without expensive transformations. A garbler can "mis-garble" the function, either replacing it with a different function entirely or causing an error to occur in an informative way (this is known as "selective failure").

Typically, malicious security is introduced to YGC by using the cut-and-choose transformation [LP15, Lin13, HKE13]. To achieve a $2^{-\lambda}$ probability of cheating without detection, the parties need to exchange λ garbled circuits [Lin13].[6] Some of the garbled circuits are "checked", and the rest of them are evaluated, their outputs checked against one another for consistency. Because of the factor

[6] There are techniques [LR14] that improve this number in the amortized case when many computations are done — however, this does not fit our setting.

of λ computational overhead, though, cut-and-choose is expensive, and too heavy a tool for fPAKE. In Section 3.2.2, we describe how we achieve security against malicious adversaries for a lower price, without this generic transformation.

Below, we summarize the garbling scheme formalization of Bellare *et al.* [BHR12], which is a generalization of YGC.

Functionality. A garbling scheme $\mathcal{G}$ consists of four polynomial-time algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$:

1. $\mathsf{Gb}(1^\lambda, f) \to (F, e, d)$. The garbling algorithm Gb takes in the security parameter λ and a circuit f, and returns a garbled circuit F, encoding information e, and decoding information d.
2. $\mathsf{En}(e, x) \to X$. The encoding algorithm En takes in the encoding information e and an input x, and returns a garbled input X.
3. $\mathsf{Ev}(F, X) \to Y$. The evaluation algorithm Ev takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.
4. $\mathsf{De}(d, Y) \to y$. The decoding algorithm De takes in the decoding information d and the garbled output Y, and returns the plaintext output y.

A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is *projective* if encoding information e consists of $2n$ wire labels, where n is the number of input bits. Two wire labels are associated with each bit of the input; one wire label corresponds to the event of that bit being 0, and the other corresponds to the event of that bit being 1. Garbled input includes only the wire labels corresponding to the actual values of the input bits. In projective schemes, in order to give the evaluator the garbled input she needs for evaluation, the garbler can send her all of the wire labels corresponding to the garbler's input. The evaluator can the use OT to retrieve the wire labels corresponding to her own input.

Similarly, we call a garbling scheme *output-projective* if decoding information d consists of two labels for each output bit, one corresponding to each possible value of that bit. The garbling schemes used in this paper are both projective and output-projective.

Correctness. Informally, a garbling scheme $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is *correct* if it always holds that $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)$.

Security. Bellare *et al.* [BHR12] describe three security notions for garbling schemes: *obliviousness*, *privacy* and *authenticity*. Informally, a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is *oblivious* if a garbled function F and a garbled input X do not reveal anything about the input x. It is *private* if additionally knowing the decoding information d reveals the output y, but does not reveal anything more about the input x. It is *authentic* if an adversary, given F and X, cannot find a garbled output $Y' \neq \mathsf{Ev}(F, X)$ which decodes without error.

In Appendix B, we define a new property of output-projective garbling schemes called *garbled output randomness*. Informally, it states that even given one of the output labels, the other should be indistinguishable from random.

3.2 Construction

Building a fPAKE from YGC and OT is not straightforward, since all constructions of OT assume authenticated channels, and fPAKE (or PAKE) is designed with unauthenticated channel in mind. We therefore follow the framework of Canetti *et al.* [CDVW12], who build a UC secure PAKE protocol using OT. We first build our protocol assuming authenticated channels, and then apply the generic transformation of Barak *et al.* [BCL+05] to adapt it to the unauthenticated channel setting. More formally, we proceed in three steps:

1. First, in Section 3.2.1, we define a randomized fuzzy equality-testing functionality $\mathcal{F}_{\mathsf{RFE}}$, which is analogous to the randomized equality-testing functionality of Canetti *et al.*
2. In Section 3.2.2, we build a protocol that securely realizes $\mathcal{F}_{\mathsf{RFE}}$ in the OT-hybrid model, assuming authenticated channels.
3. In Section 3.2.3, we apply the transformation of Barak *et al.* to our protocol. This results in a protocol that realizes the "split" version of functionality $\mathcal{F}_{\mathsf{RFE}}^{P}$, which we show to be enough to implement to fPAKE^{P}. Split functionalities, which are introduced by Barak *et al.*, adapt functionalities which assume authenticated channels to an unauthenticated channels setting. The only additional ability an adversary has in a split functionality is the ability to execute the protocol separately with the participating parties.

3.2.1 The Randomized Fuzzy Equality Functionality Figure 3 shows the randomized fuzzy equality functionality $\mathcal{F}_{\mathsf{RFE}}^{P}$, which is essentially what $\mathcal{F}_{\mathsf{fPAKE}}^{P}$ would look like assuming authenticated channels. The primary difference between $\mathcal{F}_{\mathsf{RFE}}^{P}$ and $\mathcal{F}_{\mathsf{fPAKE}}^{P}$ is that the only pass-string guesses allowed by $\mathcal{F}_{\mathsf{RFE}}^{P}$ are the ones actually used as protocol inputs; this limits the adversary to guessing by corrupting one of the participating parties, not man in the middle attacks. Like $\mathcal{F}_{\mathsf{fPAKE}}^{P}$, if a pass-string guess is "similar enough", the entire pass-string is leaked. This leakage could be replaced with any other leakage from Section 2; $\mathcal{F}_{\mathsf{RFE}}$ would leak the correctness of the guess, $\mathcal{F}_{\mathsf{RFE}}^{M}$ would leak which characters are the same between the two pass-strings, etc.

Note that, unlike the randomized equality functionality in the work of Canetti *et al.* [CDVW12], $\mathcal{F}_{\mathsf{fPAKE}}^{P}$ has a `TestPwd` interface. This is because `NewKey` does not return the necessary leakage to an honest user. So, an interface enabling the adversary to retrieve additional information is necessary.

3.2.2 A Randomized Fuzzy Equality Protocol In Figure 4 we introduce a protocol Π_{RFE} that securely realizes $\mathcal{F}_{\mathsf{RFE}}^{P}$ using Yao's Garbled Circuits. Garbled circuits are secure against a malicious evaluator, but only a semi-honest garbler; however, we obtain security against malicious adversaries by having each party play each role once. In more detail, both parties $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$ proceed as follows:

1. $\mathcal{P}_i$ garbles the circuit f that takes in two pass-strings pw_0 and pw_1, and returns '1' if $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$ and '0' otherwise. Section 3.3 describes how

The functionality $\mathcal{F}_{\mathsf{RFE}}$ is parameterized by a security parameter λ and a tolerance δ. It interacts with an adversary $\mathcal{S}$ and two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from party $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$:**
  - Send $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$;
  - If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$, then record $(\mathcal{P}_i, \mathsf{pw}_i)$.
- **Upon receiving a query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i)$ from the adversary $\mathcal{S}$, $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$:**
  If records of the form $(\mathcal{P}_0, \mathsf{pw}_0)$ and $(\mathcal{P}_1, \mathsf{pw}_1)$ do not exist, if $\mathcal{P}_{1-i}$ is not corrupted, or this is not the first `TestPwd` query for $\mathcal{P}_i$, ignore this query. Otherwise, if $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, send pw_i to the adversary $\mathcal{S}$.
- **Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{sk})$ from the adversary $\mathcal{S}$, $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$:**
  If there are no record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ and $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$, or if this is not the first `NewKey` query for $\mathcal{P}_i$, then ignore this query. Otherwise:
  - If at least one of the following is true, then output $(\mathsf{sid}, \mathsf{sk})$ to party $\mathcal{P}_i$.
    - $\mathcal{P}_i$ is corrupted
    - $\mathcal{P}_{1-i}$ is corrupted and $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$
  - If both parties are honest, $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, and a key k_{1-i} was sent to $\mathcal{P}_{1-i}$, then output $(\mathsf{sid}, \mathsf{k}_{1-i})$ to $\mathcal{P}_i$.
  - In any other case, pick a new random key k_i of length λ and send $(\mathsf{sid}, \mathsf{k}_i)$ to $\mathcal{P}_i$.

Fig. 3. Ideal Functionality $\mathcal{F}^P_{\mathsf{RFE}}$ for Randomized Fuzzy Equality

f can be designed efficiently for Hamming distance. Instead of using the output of f ('0' or '1'), we will use the garbled output, also referred to as an *output label* in an output-projective garbling scheme. The possible output labels are two random strings — one corresponding to a '1' output (we call this label $\mathsf{k}_{i,correct}$, and one corresponding to a '0' output (we call this label $\mathsf{k}_{i,wrong}$).

2. $\mathcal{P}_i$ uses OT to retrieve the input labels from $\mathcal{P}_{1-i}$'s garbling that correspond to $\mathcal{P}_i$'s pass-string.
3. $\mathcal{P}_i$ sends $\mathcal{P}_{1-i}$ her garbled circuit, together with the input labels from her garbling that correspond to her own pass-string. After this step, $\mathcal{P}_i$ should have $\mathcal{P}_{1-i}$'s garbled circuit and a garbled input consisting of input labels corresponding to the bits of the two pass-strings.
4. $\mathcal{P}_i$ evaluates $\mathcal{P}_{1-i}$'s garbled circuit, and obtains an output label Y_{1-i}.
5. $\mathcal{P}_i$ outputs $\mathsf{k}_i = \mathsf{k}_{i,correct} \oplus Y_{1-i}$.

The natural question to ask is why can we only realize $\mathcal{F}^P_{\mathsf{RFE}}$ in this way, and not a stronger functionality with less leakage. Say $\mathcal{P}_1$ is corrupted. Π_{RFE} cannot realize a functionality that leaks less than the full pass-string pw_0 to $\mathcal{P}_1$ if $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$; intuitively, this is because if $\mathcal{P}_1$ knows a pass-string pw_1 such that $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, $\mathcal{P}_1$ can extract the actual pass-string pw_0, as follows. If $\mathcal{P}_1$

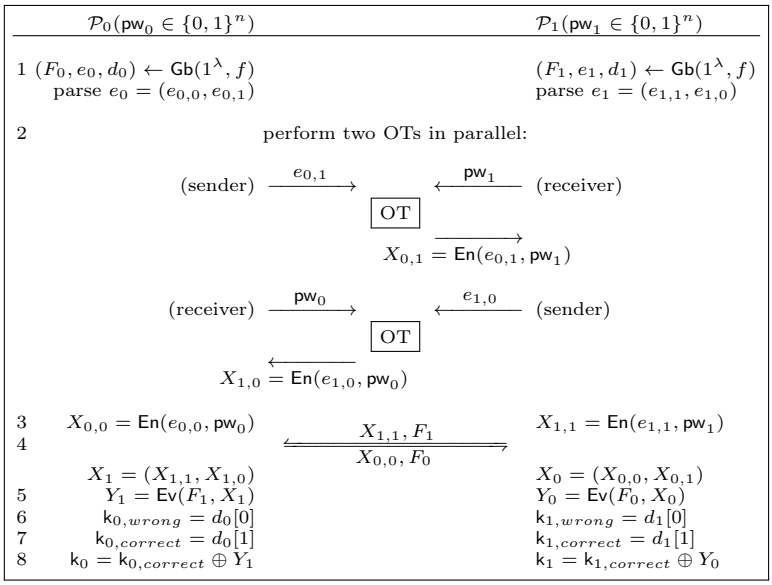

Fig. 4. A Protocol Π_{RFE} Realizing $\mathcal{F}^P_{\mathsf{RFE}}$ using Yao's Garbled Circuits and an Ideal OT Functionality. If at any point an expected message fails to arrive (or arrives malformed), the parties output a random key. Subscripts are used to indicate who produced the object in question. If a double subscript is present, the second subscript indicates whose data the object is meant for use with. For instance, a double subscript $0, 1$ denotes that the object was produced by party $\mathcal{P}_0$ for use with $\mathcal{P}_1$'s data; $e_{0,1}$ is encoding information produced by $\mathcal{P}_0$ to encode $\mathcal{P}_1$'s pass-string. Note that we abuse notation by encoding inputs to a single circuit separately; the input to $\mathcal{P}_0$'s circuit corresponding to pw_0 is encoded by $\mathcal{P}_0$ locally, and the input corresponding to pw_1 is encoded via OT. For any projective garbling scheme, this is is not a problem.

plays the role of OT receiver and garbled circuit evaluator honestly, $\mathcal{P}_0$ and $\mathcal{P}_1$ will agree on $\mathsf{k}_{0,correct}$. $\mathcal{P}_1$ can then mis-garble a circuit that returns $\mathsf{k}_{1,correct}$ if the first bit of pw_0 is 0, and $\mathsf{k}_{1,wrong}$ if the first bit of pw_0 is 1. By testing whether the resulting keys k_0 and k_1 match (which $\mathcal{P}_1$ can do in subsequent protocols where the key is used), $\mathcal{P}_1$ will be able to determine the actual first bit of pw_0. $\mathcal{P}_1$ can then repeat this for the second bit, and so on, extracting the entire pass-string pw_0. Of course, if $\mathcal{P}_1$ does *not* know a sufficiently close pw_1, $\mathcal{P}_1$ will not be able to perform these tests, because the keys will not match no matter what circuit $\mathcal{P}_1$ garbles.

More formally, if $\mathcal{P}_1$ knows a pass-string pw_1 such that $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$ and carries out the mis-garbling attack described above, then in the real world, the keys produced by $\mathcal{P}_0$ and $\mathcal{P}_1$ either will or will not match based on a bit of $\mathcal{P}_0$'s pass-string pw_0. Therefore, in the ideal world, the keys should also match or not match based on that bit; otherwise, the environment will be able to distinguish between the two worlds. In order to make that happen, the simulator must be

able to recover the entire pass-string pw_0 (given a sufficiently close pw_1) through the `TestPwd` interface.

Theorem 1. *If* $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is a projective, output-projective and garbled-output random secure garbling scheme, then protocol* Π_{RFE} *with authenticated channels in the* $\mathcal{F}_{\mathsf{OT}}$*-hybrid model securely realizes* $\mathcal{F}_{\mathsf{RFE}}^{P}$ *with respect to static corruptions for any threshold* δ*, as long as the pass-string space and notion of distance are such that for any pass-string* pw*, it is easy to compute another pass-string* pw' *such that* $d(\mathsf{pw}, \mathsf{pw}') > \delta$.[7]

Proof (Sketch). For every efficient adversary $\mathcal{A}$, we describe a simulator $\mathcal{S}_{\mathsf{RFE}}$ such that no efficient environment can distinguish an execution with the real protocol Π_{RFE} and $\mathcal{A}$ from an execution with the ideal functionality $\mathcal{F}_{\mathsf{RFE}}^{P}$ and $\mathcal{S}_{\mathsf{RFE}}$. $\mathcal{S}_{\mathsf{RFE}}$ is described in Figure 20. We prove indistinguishability in a series of hybrid steps. First, we introduce the ideal functionality as a dummy node. Next, we allow the functionality to choose the parties' keys, and we prove the indistinguishability of this step from the previous using the garbled output randomness property of our garbling scheme (Definition 7, Theorem 8). Next, we simulate an honest party's interaction with another honest party without using their pass-string, and prove the indistinguishability of this step from the previous using the obliviousness property of our garbling scheme. Finally, we simulate an honest party's interaction with a corrupted party without using the honest party's pass-string, and prove the indistinguishability of this step from the previous using the privacy property of our garbling scheme.

We give a more formal proof of Theorem 1 in Appendix C.

3.2.3 From Split Randomized Fuzzy Equality to fPAKE

The Randomized Fuzzy Equality (RFE) functionality $\mathcal{F}_{\mathsf{RFE}}^{P}$ assumes authenticated channels, which an fPAKE protocol cannot do. In order to adapt RFE to our setting, we use the split functionality transformation defined by Barak *et al.* [BCL+05]. Barak *et al.* provide a generic transformation from protocols which require authenticated channels to protocols which do not. In the "transformed" protocol, an adversary can engage in two separate instances of the protocol with the sender and receiver, and they will not realize that they are not talking to one another. However, it does guarantee that the adversary cannot do anything beyond this attack. In other words, it provides "session authentication", meaning that each party is guaranteed to carry out the entire protocol with the same partner, but not "entity authentication", meaning that the identity of the partner is not guaranteed.

Barak *et al.* achieve this transformation in three steps. First, the parties generate signing and verification keys, and send one another their verification keys. Next, the parties sign the list of all keys they have received (which, in a two-party protocol, consists of only one key), sign that list, and send both

[7] This is used in the argument of indistinguishability of Games $\mathbf{G}_2$ and $\mathbf{G}_3$ in Appendix C.

The functionality $s\mathcal{F}^P_{\mathsf{RFE}}$ is parameterized by a security parameter λ. It interacts with an adversary $\mathcal{S}$ and two parties $\mathcal{P}_0$ and $\mathcal{P}_1$ via the following queries:

- **Initialization**
  - **Upon receiving a query** $(\mathtt{Init}, \mathsf{sid})$ **from a party** $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$, send $(\mathtt{Init}, \mathsf{sid}, \mathcal{P}_i)$ to the adversary $\mathcal{S}$.
  - **Upon receiving a query** $(\mathtt{Init}, \mathsf{sid}, \mathcal{P}_i, H, \mathsf{sid}_H)$ **from the adversary** $\mathcal{S}$:
    - Verify that $H \subseteq \{\mathcal{P}_0, \mathcal{P}_1\}$, that $\mathcal{P}_i \in H$, and that if a previous set H' was recorded, either (1) $H \cap H'$ contains only corrupted parties and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or (2) $H = H'$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$.
    - If verification fails, do nothing.
    - Otherwise, record the pair (H, sid_H) (if it was not already recorded), output $(\mathtt{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to $\mathcal{P}_i$, and locally initialize a new instance of the original RFE functionality $\mathcal{F}_{\mathsf{RFE}}$ denoted $H\mathcal{F}^P_{\mathsf{RFE}}$, letting the adversary play the role of $\{\mathcal{P}_0, \mathcal{P}_1\} - H$ in $H\mathcal{F}^P_{\mathsf{RFE}}$.
- **RFE**
  - **Upon receiving a query from a party** $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$**,** find the set H such that $\mathcal{P}_i \in H$, and forward the query to $H\mathcal{F}^P_{\mathsf{RFE}}$. Otherwise, ignore the query.
  - **Upon receiving a query from the adversary** $\mathcal{S}$ **on behalf of** $\mathcal{P}_i$ **corresponding to set** H**,** if $H\mathcal{F}^P_{\mathsf{RFE}}$ is initialized and $\mathcal{P}_i \notin H$, then forward the query to $H\mathcal{F}^P_{\mathsf{RFE}}$. Otherwise, ignore the query.

Fig. 5. Functionality $s\mathcal{F}^P_{\mathsf{RFE}}$

list and signature to all other parties. Finally, they verify all of the signatures they have received. After this process — called "link initialization" — has been completed, the parties use those public keys they have exchanged to authenticate subsequent communication.

We describe the Randomized Fuzzy Equality Split Functionality in Figure 5. It is simplified from Figure 1 in Barak *et al.* [BCL+05] because we only need to consider two parties and static corruptions.

It turns out that $s\mathcal{F}^P_{\mathsf{RFE}}$ is enough to realize $\mathcal{F}^P_{\mathsf{fPAKE}}$. In fact, the protocol Π_{RFE} with the split functionality transformation directly realizes $\mathcal{F}^P_{\mathsf{fPAKE}}$. In Appendix D, we prove that this is the case.

3.3 An Efficient Circuit f for Hamming Distance

The Hamming distance of two pass-strings $\mathsf{pw}, \mathsf{pw}' \in \mathbb{F}_p^n$ is equal to the number of locations at which the two pass-strings have the same character. More formally,

$$d(\mathsf{pw}, \mathsf{pw}') := |\{j \mid \mathsf{pw}[j] \neq \mathsf{pw}'[j], j \in [n]\}|.$$

We design f for Hamming distance as follows:

1. First, f XORs corresponding (binary) pass-string characters, resulting in a list of bits indicating the (in)equality of those characters.

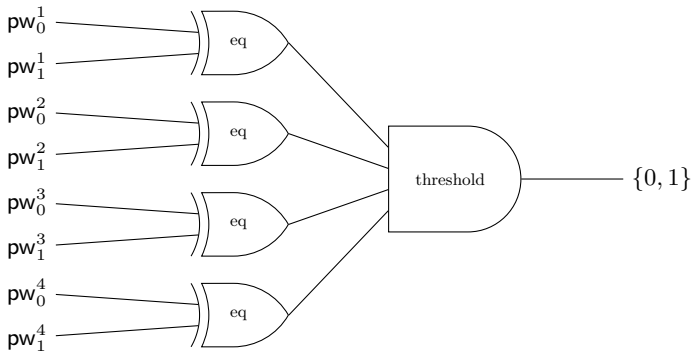

Fig. 6. The f circuit

2. Then, f feeds those bits into a threshold gate, which returns 1 if at least $n-\delta$ of its inputs are 0, and returns 0 otherwise. f returns the output of that threshold gate, which is 1 if and only if at least $n-\delta$ pass-string characters match.

This circuit, illustrated in Figure 6, is very efficient to garble; it only requires n ciphertexts. Below, we briefly explain this garbling. Our explanation assumes familiarity with YGC literature [Yak17, and references therein]. Briefly, garbled gadget labels [BMR16] enable the evaluation of modular addition gates for free (there is no need to include any information in the garbled circuit to enable this addition). However, for a small modulus m, converting the output of that addition to a binary decision requires $m-1$ ciphertexts. We utilize garbled gadgets with a modulus of $n+1$ in our efficient garbling as follows:

1. The input wire labels encode 0 or 1 modulo $n+1$. However, instead of having those input wire labels encode the characters of the two pass-strings directly, they encode the outputs of the comparisons of corresponding characters. If the jth character of $\mathcal{P}_i$'s pass-string is 0, then $\mathcal{P}_i$ puts the 0 label first; however, if the jth character of $\mathcal{P}_i$'s pass-string is 1, then $\mathcal{P}_i$ flips the labels. Then, when $\mathcal{P}_{1-i}$ is using oblivious transfer to retrieve the label corresponding to her jth pass-string character, she will retrieve the 0 label if the two characters are equal, and the 1 label otherwise. (Note that this preprocessing on the garbler's side eliminates the need to send $X_{0,0}$ and $X_{1,1}$ in Figure 4.)
2. Compute a n-input threshold gate, as illustrated in Figure 6 of Yakoubov [Yak17]. This gate returns 0 if the sum of the inputs is above a certain threshold (that is, if at least $n-\delta$ pass-string characters differ), and 1 otherwise. This will require n ciphertexts.

Thus, a garbling of f consists of n ciphertexts. Since fPAKE requires two such garbled circuits (Figure 4), $2n$ ciphertexts will be exchanged.

Larger Pass-string Characters. If larger pass-string characters are used, then Step 1 above needs to change to check (in)equality of the larger characters instead of bits. Step 2 will remain the same. There are several ways to perform an (in)equality check on characters in $\mathbb{F}_p$ for $p \geq 2$:

1. Represent each character in terms of bits. Step 1 will then consist of XORing corresponding bits, and taking an OR or the resulting XORs of each character to get negated equality. This will take an additional $n \log(p)$ ciphertexts for every pass-string character.
2. Use garbled gadget labels from the outset. We will require a larger OT (1-out-of-p instead of 1-out-of-2), but nothing else will change.

4 Specialized Construction For Hamming Distance

In Appendix H, we show that it is not straightforward to build a secure fPAKE from primitives that are, by design, well-suited for correcting errors. However, PAKE protocols are appealingly efficient compared to the garbled circuits used in the prior construction. So in this section, we will see whether the failed approach can be rescued in an efficient way, and we answer this question in the affirmative.

4.1 Preliminaries

4.1.1 Robust Secret Sharing We recall the definition of a robust secret sharing scheme, slightly simplified for our purposes from [CDD+15]. For a vector $c \in \mathbb{F}_q^n$ and a set $A \subseteq [n]$, we denote with c_A the projection $\mathbb{F}_q^n \to \mathbb{F}_q^{|A|}$, i.e., the sub-vector $(c_i)_{i \in A}$ and, similarly, $c_{\bar{A}} := (c_i)_{i \in \bar{A}}$ for the complement $\bar{A} := [n] \setminus A$.

Definition 2. *Let $\mathbb{F}_q$ be a finite field and $n, t, r \in \mathbb{N}$ with $t < r \leq n$. An (n, t, r) robust secret sharing scheme (RSS) consists of two probabilistic algorithms* $\mathsf{Share} : \mathbb{F}_q \to \mathbb{F}_q^n$ *and* $\mathsf{Reconstruct} : \mathbb{F}_q^n \to \mathbb{F}_q$ *with the following properties:*

- *t-privacy: for any $s, s' \in \mathbb{F}_q, A \subset [n]$ with $|A| \leq t$, the projections c_A of $c \xleftarrow{\$} \mathsf{Share}(s)$ and c'_A of $c' \xleftarrow{\$} \mathsf{Share}(s')$ are identically distributed.*
- *r-robustness: for any $s \in \mathbb{F}_q, A \subset [n]$ with $|A| \geq r$, any c output by $\mathsf{Share}(s)$, and any $\tilde{c}$ such that $c_A = \tilde{c}_A$, $\mathsf{Reconstruct}(\tilde{c}) = s$.*

In other words, a (n, t, r)-RSS is able to reconstruct the shared secret even if the adversary tampered with up to $n - r$ shares, while each set of t shares is distributed independently of the shared secret s and thus reveals nothing about it. We note that we allow for a gap, i.e., $r \geq t + 1$. Schemes with $r > t + 1$ are called *ramp* RSS.

4.1.2 Linear Codes A linear q-ary code of length n and rank k is a subspace C with dimension k of the vector space $\mathbb{F}_q^n$. The vectors in C are called codewords. The size of a code is the number of codewords, and is thus equal to q^k. The weight of a word $w \in \mathbb{F}_q^n$ is the number of non-zero components and the distance between two words is the Hamming distance between them (equivalently, the weight of their difference). The minimal distance d of a linear code C is the minimum weight of its non-zero codewords, or equivalently, the minimum distance between any two distinct codewords.

A code for an alphabet of size q, of length n, rank k, and minimal distance d is called an $(n, k, d)_q$-code. Such a code can be used to detect up to $d-1$ errors (because if a codeword is sent and fewer than $d-1$ errors occur, it will not get transformed to another codeword), and correct up to $\lfloor (d-1)/2 \rfloor$ errors (because for any received word, there is a unique codeword within distance $\lfloor (d-1)/2 \rfloor$). For linear codes, encoding of a (row vector) word $W \in \mathbb{F}_q^k$ is performed by an algorithm $C.\mathsf{Encode} : \mathbb{F}_q^k \to \mathbb{F}_q^n$, which is the multiplication of W by a so-called "generating matrix" $G \in \mathbb{F}_q^{k \times n}$ (which defines an injective linear map). This leads to a row-vector codeword $c \in C \subset \mathbb{F}_q^n$.

The Singleton bound states that for any linear code, $k + d \leq n + 1$, and a *maximum distance separable* (or MDS) code satisfies $k + d = n + 1$. Hence, $d = n - k + 1$ and MDS codes are fully described by the parameters (q, n, k). Such an $(n, k)_q$-MDS code can correct up to $\lfloor (n-k)/2 \rfloor$ errors; it can detect if there are errors, whenever there are no more than $n - k$ of them.

For a thorough introduction to linear codes and proof of all statements in this short overview we refer the reader to [Rot06].

Observe that a linear code, due to the linearity of its encoding algorithm, is not a primitive designed to hide anything about the encoded message. However, we show in the following lemma how to turn an MDS code into a RSS scheme.

Lemma 3. *Let C be a $(n+1, k)_q$-MDS code. We set L to be the last column of the generating matrix G of the code C and we denote by C' the $(n, k)_q$-MDS code whose generating matrix G' is G without the last column. Let* Share *and* Reconstruct *work as follows:*

- $\mathsf{Share}(s)$ *for $s \in \mathbb{F}_q$ first chooses a random row vector $W \in \mathbb{F}_q^k$ such that $W \cdot L = s$, and outputs $c \leftarrow C'.\mathsf{Encode}(W)$ (equivalently, we can say that $\mathsf{Share}(s)$ chooses a uniformly random codeword of C whose last coordinate is s, and outputs the first n coordinates as c).*
- $\mathsf{Reconstruct}(w)$ *for $w \in \mathbb{F}_q^n$ first runs $C'.\mathsf{Decode}(w)$. If it gets a vector W', then output $s = W' \cdot L$, otherwise output $s \xleftarrow{\$} \mathbb{F}_q$.*

Then Share *and* Reconstruct *form a (n, t, r)-RSS for $t = k-1$ and $r = \lceil (n+k)/2 \rceil$.*

Proof. Let us consider the two properties from Definition 2.

- t-privacy: Assume $|A| = t$ (privacy for smaller A will follow immediately by adding arbitrary coordinates to it to get to size t). Let $J = A \cup \{n+1\}$; note that $|J| = t + 1 = k$. Note that for the code C, any k coordinates of a

codeword determine uniquely the input to Encode that produces this codeword (otherwise, there would be two codewords that agreed on k elements and thus had distance $n-k+1$, which is less than the minimum distance of C). Therefore, the mapping given by $\mathsf{Encode}_J : \mathbb{F}_q^k \to \mathbb{F}_q^{|J|}$ is bijective; thus coordinates in J are uniform when the input to Encode is uniform. The algorithm Share chooses the input to Encode uniformly subject to fixing the coordinate $n+1$ of the output. Therefore, the remaining coordinates (i.e., the coordinates in A) are uniform.

– r-robustness: Note that C has minimum distance $n-k+2$, and therefore C' has minimum distance $n-k+1$ (because dropping one coordinate reduces the distance by at most 1). Therefore, C' can correct $\lfloor (n-k)/2 \rfloor = n-r$ errors. Since $c_A = \tilde{c}_A$ and $|A| \geq r$, there are at most $n-r$ errors in $\tilde{c}$, so the call to $C'.\mathsf{Decode}(c')$ made by $\mathsf{Reconstruct}(c')$ will output $W' = W$. Then $\mathsf{Reconstruct}(c')$ will output $s = W' \cdot L = W \cdot L$.

Note that the Shamir's secret sharing scheme is exactly the above construction with Reed-Solomon codes [MS81].

4.1.3 Implicit-Only PAKE

PAKE protocols can have two types of authentication: implicit authentication, where at the end of the protocol the two parties share the same key if they used the same pass-string and random independent keys otherwise; or explicit authentication where, in addition, they actually know which of the two situations they are in. A PAKE protocol that only achieves implicit authentication can provide explicit authentication by adding key-confirmation flows [BPR00].

The standard PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ from [CHK+05] (see Figure 16) is designed with explicit authentication in mind, or at least considers that success or failure will later be detected by the adversary when he will try to use the key. Thus, it reveals to the adversary whether a pass-string guess attempt was successful or not. However, some applications could require a PAKE that does not provide any feedback, and so does not reveal the situation before the keys are actually used. Observe that, regarding honest players, already $\mathcal{F}_{\mathsf{pwKE}}$ features implicit authentication since the players do not learn anything but their own session key.

Definition of implicit-only PAKE. Hence, we introduce a new notion, called implicit-only PAKE or iPAKE (see Figure 7). This ideal functionality is designed to implement implicit authentication also with respect to an adversary, namely by not providing him with any feedback upon a dictionary attack. Of course, in many cases, the parties as well as the adversary can later check whether their session keys match or not, and so whether the pass-strings were the same or not. We stress that this is not a leakage from the PAKE protocol itself, but from the global system.

In terms of functionalities, there are two differences from $\mathcal{F}_{\mathsf{pwKE}}$ to $\mathcal{F}_{\mathsf{iPAKE}}$. First, the `TestPwd` query only silently updates the internal state of the record (from `fresh` to either `compromised` or `interrupted`), meaning that its outcome

The functionality $\mathcal{F}_{\mathsf{iPAKE}}$ is parameterized by a security parameter λ. It interacts with an adversary $\mathcal{S}$ and the (dummy) parties $\mathcal{P}_0$ and $\mathcal{P}_{1-i}$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i)$ from party $\mathcal{P}_i$:**
  - Send $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$;
  - If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$, then record $(\mathcal{P}_i, \mathsf{pw}_i)$ and mark this record `fresh`.
- **Upon receiving a query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}'_i)$ from $\mathcal{S}$:**
  If there is a `fresh` record $(\mathcal{P}_i, \mathsf{pw}_i)$, then:
  - If $\mathsf{pw}_i = \mathsf{pw}'_i$, mark the record `compromised`;
  - If $\mathsf{pw}_i \neq \mathsf{pw}'_i$, mark the record `interrupted`.
- **Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{sk})$ from $\mathcal{S}$, where $|\mathsf{sk}| = \lambda$:**
  If there is no record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$, or if this is not the first `NewKey` query for $\mathcal{P}_i$, then ignore this query. Otherwise:
  - If at least one of the following is true, then output $(\mathsf{sid}, \mathsf{sk})$ to player $\mathcal{P}_i$:
    - The record is `compromised`
    - $\mathcal{P}_i$ is corrupted
    - The record is `fresh`, $\mathcal{P}_{1-i}$ is corrupted, and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $\mathsf{pw}_i = \mathsf{pw}_{1-i}$
  - If this record is `fresh`, both parties are honest, there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $\mathsf{pw}_i = \mathsf{pw}_{1-i}$, a key sk' was sent to $\mathcal{P}_{1-i}$, and $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was `fresh` at the time, then output $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$;
  - In any other case, pick a new random key sk' of length λ and send $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$.
  - Mark the record $(\mathcal{P}_i, \mathsf{pw}_i)$ as `completed`.

Fig. 7. Functionality $\mathcal{F}_{\mathsf{iPAKE}}$

is not given to the adversary $\mathcal{S}$. Second, the `NewKey` query is modified so that the adversary gets to choose the key for a non-corrupted party only if it uses the correct pass-string (corruption of the other party is no longer enough), as already discussed earlier. Without going too much into the details, it is intuitively clear that simulation of an honest party is hard if the simulator does not know whether it should proceed the simulation with a pass-string extracted from a dictionary attack or not. Regarding the output, i.e., the question whether the session keys computed by both parties should match or look random, the simulator thus gets help from our functionality by modifying the `NewKey` queries.

We further alter this functionality to allow for public labels, see in figure 25 in Appendix F. The resulting functionality $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$ idealizes what we call *labeled implicit-only PAKE* (or ℓ-iPAKE for short), resembling the notion of labeled public key encryption as formalized in [Sho01]. In a nutshell, labels are public authenticated strings that are chosen by each user individually for each execution of the protocol. Authenticated here means that tampering with the label can be efficiently detected. Such labels can be used to, e.g., distribute public information such as public keys reliably over unauthenticated channels.

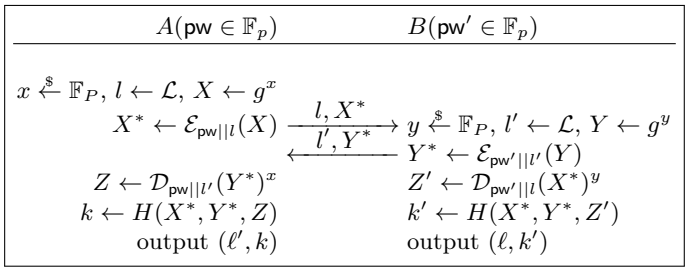

Fig. 8. Protocol EKE2, in a group $\mathbb{G} = \langle g \rangle$ of prime order P, with a hash function $H : \mathbb{G}^3 \to \{0,1\}^k$ and a symmetric cipher $\mathcal{E}, \mathcal{D}$ onto $\mathbb{G}$ for keys in $\mathbb{F}_p \times \mathcal{L}$.

A UC-Secure ℓ-iPAKE Protocol. In the seminal paper by Bellovin and Merritt [BM92] that introduced PAKE, the Encrypted Key Exchange protocol (EKE) is proposed, which is essentially a Diffie-Hellman [DH76] key exchange. The two flows of the protocol are encrypted using the pass-string as key with an appropriate symmetric encryption scheme. The EKE protocol has been further formalized by Bellare *et al.* [BPR00] under the name EKE2. We present its labeled variant in Figure 8. The idea of appending the label to the symmetric key is taken from [ACCP08].

Theorem 4. *If the CDH assumption holds in $\mathbb{G}$, the protocol EKE2 depicted in Figure 8 securely realizes $\mathcal{F}_{\ell\text{-iPAKE}}$ in the $\mathcal{F}_{RO}, \mathcal{F}_{IC}, \mathcal{F}_{CRS}$-hybrid model with respect to static corruptions.*

We note that this result is not surprising, given that other variants of EKE2 have already been proven to UC-emulate $\mathcal{F}_{\mathsf{pwKE}}$. Intuitively, a protocol with only two flows not depending on each other does not leak the outcome to the adversary via the transcript, which explains why EKE2 is implicit-only. Hashing of the transcript keeps the adversary from biasing the key unless he knows the correct pass-string or breaks the ideal cipher. For completeness, we include the full proof in Appendix F.

4.2 Construction

We show how to combine an RSS with a signature scheme and an ℓ-iPAKE to obtain an fPAKE. The high-level idea is to fix the issue that arose in the protocol from Appendix H due to pass-strings being used as one-time pads. Instead, we first expand the pass-string characters to session keys with large entropy using ℓ-iPAKE. The resulting session keys are then used as a one-time pad on the entirety of shares of a nonce. We also apply known techniques from the literature, such as executing the protocol twice with reversed roles to protect against malicious parties, and adding signatures and labels to prevent man-in-the-middle attacks. Our full protocol is depicted in Figure 9. It works as follows:

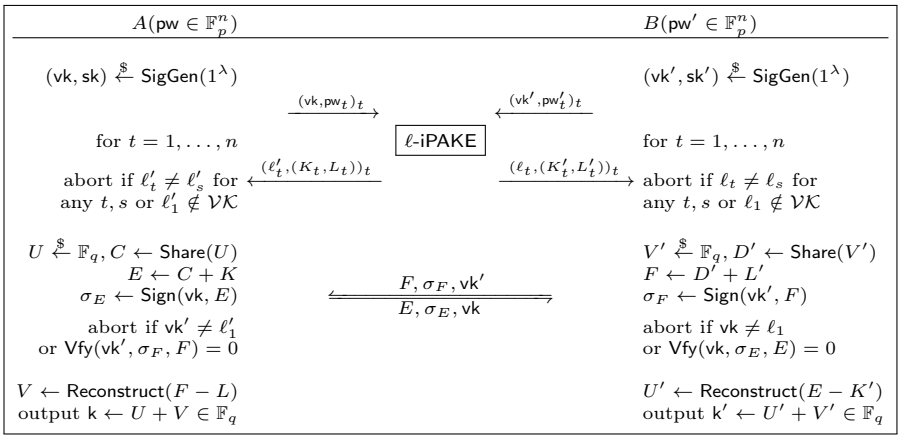

Fig. 9. Protocol $\mathsf{fPAKE}_{\mathsf{RSS}}$ where $q \approx 2^\lambda$ is a prime number and $+$ denotes the group operation in $\mathbb{F}_q^n$. $(\mathsf{Share}, \mathsf{Reconstruct})$ is a Robust Secret Sharing scheme with $\mathsf{Share} : \mathbb{F}_q \to \mathbb{F}_q^n$, and $(\mathsf{SigGen} \to \mathcal{VK} \times \mathcal{SK}, \mathsf{Sign}, \mathsf{Vfy})$ is a signature scheme. The parties repeatedly execute a labeled implicit-only PAKE protocol with label space $\mathcal{VK}$ and key space $\mathbb{F}_q^2$, which takes inputs from $\mathbb{F}_p$. If at any point an expected message fails to arrive (or arrives malformed), the parties output a random key.

1. In the first phase, the two parties aim at enhancing their pass-strings to a vector of session keys with good entropy. For this, pass-strings are viewed as vectors of characters. The parties repeatedly execute a PAKE on each of these characters separately. The PAKE will ensure that the key vectors held by the two parties match in all positions where their pass-strings matched, and are uniformly random in all other positions.
2. In the second phase, the two parties exchange nonces of their choice, in such a way that the nonce reaches the other party only if enough of the key vector matches. This is done by applying an RSS to the nonce, and sending it to the other party using the key vector as a one time pad. Both parties do this symmetrically, each using half of the bits of the key vector. The robustness property of the RSS ensures that a few non-matching pass-string characters do not prevent both parties from recovering the other party's nonce. The final key is then obtained by adding the nonces (again, as a one-time pad): this is a scalar in $\mathbb{F}_q$.

When using the RSS from MDS codes described in Lemma 3, the one-time pad encryption of the shares (which form a codeword) can be viewed as the code-offset construction for information reconciliation (aka secure sketch) [JW99, DRS04] applied to the key vectors. While our presentation goes through RSS as a separate object, we could instead present this construction using information reconciliation. The syndrome construction of secure sketches 3 can also be used here instead of the code-offset construction.

4.3 Security of $\mathsf{fPAKE}_{\mathsf{RSS}}$

We show that our protocol realizes functionality $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ in the $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$-hybrid model. In a nutshell, the idea is to simulate without the pass-strings by adjusting the keys outputted by $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$ to the mask of the pass-strings, which is leaked by $\mathcal{F}_{\mathsf{fPAKE}}^{M}$.

Theorem 5. *If* $(\mathsf{Share} : \mathbb{F}_q \to \mathbb{F}_q^n, \mathsf{Reconstruct} : \mathbb{F}_q^n \to \mathbb{F}_q)$ *is an* (n, t, r) *RSS and* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vfy})$ *is an EUF-CMA secure one-time signature scheme, protocol* fPAKE_{RSS} *securely realizes* $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ *with* $\gamma = n - t - 1$ *and* $\delta = n - r$ *in the* $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$*-hybrid model with respect to static corruptions.*

In particular, if we wish key agreement to succeed as long as there are fewer than δ errors, we instantiate RSS using the construction of Lemma 3 based on a $(n+1, k)_q$ MDS code, with $k = n - 2\delta$. This will give $r = \lceil (n+k)/2 \rceil = n - \delta$, so δ will be equal to $n - r$, as required. It will also give $\gamma = n - t - 1 = 2\delta$.

We thus obtain the following corollary:

Corollary 6. *For any* δ *and* $\gamma = 2\delta$*, given an* $(n+1, k)_q$*-MDS code for* $k = n - 2\delta$ *(with minimal distance* $d = n - k + 2$*) and an EUF-CMA secure one-time signature scheme, protocol* fPAKE_{RSS} *securely realizes* $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ *in the* $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$*-hybrid model with respect to static corruptions.*

Proof sketch of Theorem 5. We start with the real execution of the protocol and indistinguishably switch to an ideal execution with dummy parties relaying their inputs to and obtaining their outputs from $\mathcal{F}_{\mathsf{fPAKE}}^{M}$. To preserve the view of the distinguisher, the environment $\mathcal{Z}$, a simulator $\mathcal{S}$ plays the role of the real world adversary by controlling the communication between $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ and $\mathcal{Z}$. During the proof, we built $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ and $\mathcal{S}$ by subsequently randomizing pass-strings (since the final simulation has to work without them) and session keys (since $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ hands out random session keys in certain cases). We have to tackle the following difficulties, which we will describe in terms of attacks.

- Passive attack: in this attack, $\mathcal{Z}$ picks two pass-strings and then observes the transcript and outputs of the protocol, without having access to any internal state of the parties. We show that $\mathcal{Z}$ cannot distinguish between transcript and outputs that were either produced using $\mathcal{Z}$'s pass-strings or random pass-strings. Regarding the outputs, we argue that even in the real execution the session keys were chosen uniformly at random (with $\mathcal{Z}$ not knowing the coins consumed by this choice) as long as the distance check is reliable. Using properties of the RSS, we show that this is the case with overwhelming probability. Regarding the transcript, randomization is straightforward using properties of the one-time pad.
- Man-in-the-middle attack: in this attack, $\mathcal{Z}$ injects a malicious message into a session of two honest parties. There are several ways to secure protocols that have to run in unauthenticated channels and are prone to this attack. Basically, all of them introduce methods to bind messages together to prevent

the adversary from injecting malicious messages. To do this, we need the *labeled* version of our iPAKE and a one-time signature scheme[8]. Unless $\mathcal{Z}$ is able to break a one-time-signature scheme, this attack always results in an abort.

– Active attack: in this attack, $\mathcal{Z}$ injects a malicious message into a session with one corrupted party, thereby knowing the internal state of this party. We show how to produce transcript and outputs looking like in a real execution, but without using the pass-strings of the honest party. Since $\mathcal{Z}$ can now actually decrypt the one-time pad and therefore the transcript reveals the positions of the errors in the pass-strings, $\mathcal{S}$ has to rely on $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ revealing the mask of the pass-strings used in the real execution. If, on the other hand, the pass-strings are too far away from each other, we show that the privacy property of the RSS actually hides the number and positions of the errors. This way, $\mathcal{S}$ can use a random pass-string to produce the transcript in that case.

One interesting subtlety that arises is the usage of the iPAKE. Observe that the UC security notion for a regular PAKE as defined in [CHK+05] and recalled in Appendix A provides an interface to the adversary to test a pass-string once and learn whether it is right or wrong. Using this notion, our simulator would have to answer to such queries from $\mathcal{Z}$. Since this is not possible without $\mathcal{F}_{\mathsf{fPAKE}}^{M}$ leaking the mask all the time, it is crucial to use the iPAKE variant that we introduced in section 4.1.3. Using this stronger notion, the adversary is still allowed one pass-string guess which may affect the output, but the adversary learns nothing more about the outcome of his guess than he can infer from whatever access he has to the outputs alone. Since our protocol uses the outputs of the PAKE as one-time pad keys, it is intuitively clear that by preventing $\mathcal{Z}$ from getting additional leakage about these keys, we protect the secrets of honest parties.

4.4 Further Discussion

4.4.1 Adaptive Corruptions Adaptive security of our protocol is not achievable without relying on additional assumptions. To see this, consider the following attack: $\mathcal{Z}$ starts the protocol with two equal pass-strings and, without corrupting anyone, silently observes the transcript produced by $\mathcal{S}$ using random pass-strings. Afterwards, $\mathcal{Z}$ corrupts both players to learn their internal state. $\mathcal{S}$ may now choose a value K. This also fixes $L' = K$ since the pass-strings were equal. Now note that $\mathcal{S}$ is committed to E, F since signatures are not equivocable. Since perfect shares are sparse in $\mathbb{F}_q^n$, the probability that there exists a K

[8] Instead of labels and one-time signature, one could just sign all the messages, as would be done using the split-functionality [BCL+05], but this would be less efficient. This trade-off, with labels, is especially useful when we use a PAKE that admits adding labels basically for free, as it is the case with the special PAKE protocol we use.

such that $E - K$ and $F - K$ are both perfect shares is negligible. Thus, there do not exist plausible values U, V' that explain the transcript[9].

4.4.2 Removing Modeling Assumptions All modeling assumptions of our protocol come from the realization of the ideal $\mathcal{F}_{\ell\text{-iPAKE}}$ functionality. E.g., the ℓ-iPAKE protocol from section 4.1.3 requires a random oracle, an ideal cipher and a CRS. We note that we can remove everything up to the CRS by, e.g., taking the PAKE protocol introduced in [KV11]. This protocol also securely realizes our $\mathcal{F}_{\ell\text{-iPAKE}}$ functionality[10]. However, it is more costly than our ℓ-iPAKE protocol since both messages each contain one non-interactive zero knowledge proof.

Since fPAKE implies a regular PAKE (simply set $\delta = 0$), [CHK+05] gives strong evidence that we cannot hope to realize $\mathcal{F}_{\text{fPAKE}}$ without a CRS.

5 Comparison of fPAKE Protocols

In this section, we give a brief comparison of our fPAKE protocols. First, in Figure 10, we describe the assumptions necessary for the two constructions, and the security parameters that they can achieve.

	Assumptions	Threshold δ	Gap $\gamma - \delta$
$\text{fPAKE}_{\text{RSS}}$	UC-secure ℓ-iPAKE	$< n/2$	δ
$\text{fPAKE}_{\text{YGC}}$	(1) UC-secure OT (2) projective, output-projective and garbled-output random secure garbling scheme	Any	None

Fig. 10. Assumptions, Distance Thresholds and Functionality/Security Gaps achieved by the two schemes. $\text{fPAKE}_{\text{RSS}}$ is the construction in Figure 9. $\text{fPAKE}_{\text{YGC}}$ is the construction in Figure 4 with the split functionality transformation of Barak *et al.* [BCL+05].

Then, in Figure 11, we describe the efficiency of the constructions when concrete primitives (OT / ℓ-iPAKE) are used to instantiate them. $\text{fPAKE}_{\text{RSS}}$ is

[9] We note that additional assumptions like assuming erasures can enable an adaptive security proof.

[10] In a nutshell, their protocol is implicit-only for the same reason as the ℓ-iPAKE protocol we use here: there are only two flows that do not depend on each other, so the transcript cannot reveal the outcome of a guess unless it reveals the pass-string to anyone. Regarding the session keys, usage of a hash function takes care of randomizing the session key in case of a failed dictionary attack. Furthermore, the protocol already implements labels. A little more detailed, looking at the proof in [KV11], the simulator does not make use of the answer of `TestPwd` to simulate any messages. Regarding the session key that an honest player receives in an corrupted session, they are chosen to be random in the simulation (in Expt_3). Letting this happen already in the functionality makes the simulation independent of the answer of `TestPwd` also regarding the computation of the session keys.

instantiated as the construction in Figure 9 with the ℓ-iPAKE in Figure 8 and a RSS. $\mathsf{fPAKE}_{\mathsf{YGC}}$ is instantiated as the construction in Figure 4 with the UC-secure oblivious transfer protocol of Chou and Orlandi [CO15] described in Figure 24, with the garbling scheme of Bal *et al.* [BMR16], and with the split functionality transformation of Barak *et al.* [BCL+05]. We describe the efficiency in terms of sub-operations (per-party, not in aggregate).

Note that these concrete primitives each have their own set of required assumptions. Specifically, the ℓ-iPAKE in Figure 8 requires a random oracle (RO, described in Figure 13), ideal cipher (IC, described in Figure 14) and common reference string (CRS, described in Figure 12). The oblivious transfer protocol in Figure 24 requires a random oracle. The garbling scheme of Bal *et al.* [BMR16] requires a Mixed Modulus Circular Correlation Robust hash function, which is a weakening of the random oracle assumption.

For $\mathsf{fPAKE}_{\mathsf{RSS}}$, the factor of n arises from the n times EKE2 is executed. For $\mathsf{fPAKE}_{\mathsf{YGC}}$, the factor of n comes from the garbled circuit. Additionally, in $\mathsf{fPAKE}_{\mathsf{YGC}}$, three rounds of communication come from OT, one is required to send the garbled circuits, and two additional rounds of communication come from the split functionality transformation. The need for signatures also arises from the split functionality transformation.

	Output Key Format	*# (Bidirectional) Communication Flows*	*# Exponentiations*	*# Hashes*	*# Encryptions*	*# Decryptions*	*# Share*	*# Reconstruct*	*# SigKeyGens*	*# Signs*	*# SigVerifies*
$\mathsf{fPAKE}_{\mathsf{RSS}}$	$\mathbb{F}_q$	2	$2n$	n	n	n	1	1	1	1	1
$\mathsf{fPAKE}_{\mathsf{YGC}}$	$\{0,1\}^\lambda$	6	$3n+2$	$4n+7$	$2n$	n	–	–	1	5	5

Fig. 11. Efficiency (in Terms of Sub-Operations) of the Two Constructions. $\mathsf{fPAKE}_{\mathsf{RSS}}$ is the construction in Figure 9 instantiated with the ℓ-iPAKE in Figure 8. $\mathsf{fPAKE}_{\mathsf{YGC}}$ is the construction in Figure 4 instantiated with the UC-secure oblivious transfer protocol of Chou and Orlandi [CO15] described in Figure 24, the garbling scheme of Bal *et al.* [BMR16], and with the split functionality transformation of Barak *et al.* [BCL+05].

Efficiency Optimizations to $\mathsf{fPAKE}_{\mathsf{YGC}}$ We can make two small efficiency improvements to the $\mathsf{fPAKE}_{\mathsf{YGC}}$ construction. (They are not reflected in Figure 11, because they are not a part of the scheme the security of which we prove.) First, the last flow of the OT protocol can be combined with the sending of the garbled circuits, reducing the number of bidirectional communication flows from 6 to 5. Second, instead of using the split functionality transformation of Barak *et al.* [BCL+05], we can use the split split functionality of Camenisch *et*

al. [CCGS10]. It uses a split key exchange functionality to establish symmetric keys, and then uses those to symmetrically encrypt and authenticate each flow. While this does not save any rounds, it does reduce the number of public key operations needed.

Acknowledgments

We thank Ran Canetti for guidance on the details of UC key agreement definitions, and Adam Smith for discussions on coding and information reconciliation.

This work was supported in part by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud). Leonid Reyzin gratefully acknowledges the hospitality of École Normale Supérieure, where some of this work was performed. He was supported, in part, by US NSF grants 1012910, 1012798, and 1422965.

References

ACCP08. Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In Tal Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 335–351. Springer, Heidelberg, April 2008.

BBR88. Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM Journal on Computing*, 17(2):210–229, 1988.

BCKP14. Nir Bitansky, Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On virtual grey box obfuscation for general circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2014.

BCL+05. Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Heidelberg, August 2005.

BDK+05. Xavier Boyen, Yevgeniy Dodis, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Secure remote authentication using biometric data. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 147–163. Springer, Heidelberg, May 2005.

BH09. Marina Blanton and William MP Hudelson. Biometric-based non-transferable anonymous credentials. In *Information and Communications Security*, pages 165–180. Springer, 2009.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, 2012. http://eprint.iacr.org/2012/265.

BM92. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

BMP00. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

BMR16. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 565–577. ACM Press, October 2016.

Boy04. Xavier Boyen. Reusable cryptographic fuzzy extractors. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04*, pages 82–91. ACM Press, October 2004.

BPR00. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

BS00. Sacha Brostoff and M.Angela Sasse. Are passfaces more usable than passwords?: A field trial investigation. *People and Computers*, pages 405–424, 2000.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

Can07. Ran Canetti. Obtaining universally compoable security: Towards the bare bones of trust (invited talk). In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 88–112. Springer, Heidelberg, December 2007.

CCGS10. Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential authenticated identification and key exchange. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 255–276. Springer, Heidelberg, August 2010.

CDD+15. Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling, Serge Fehr, and Gabriele Spini. Linear secret sharing schemes from error correcting codes and universal hash functions. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 313–336. Springer, Heidelberg, April 2015.

CDVW12. Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012.

CFP+16. Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam D. Smith. Reusable fuzzy extractors for low-entropy distributions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 117–146. Springer, Heidelberg, May 2016.

CHK+05. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

CKKZ12. Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

CO15. Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015.

Dau04. John Daugman. How iris recognition works. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):21 – 30, January 2004.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DKK+12. Yevgeniy Dodis, Bhavana Kanukurthi, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. *IEEE Transactions on Information Theory*, 58(9):6207–6222, 2012.

DORS08. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

DRS04. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

EHMS00. Carl Ellison, Chris Hall, Randy Milbert, and Bruce Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4):311–318, 2000.

FHH14. Eduarda S. V. Freire, Julia Hesse, and Dennis Hofheinz. Universally composable non-interactive key exchange. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 1–20. Springer, Heidelberg, September 2014.

GCvD02. Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In Vijayalakshmi Atluri, editor, *ACM CCS 02*, pages 148–160. ACM Press, November 2002.

GSY+16. Paolo Gasti, Jaroslav Sedenka, Qing Yang, Gang Zhou, and Kiran S. Balagani. Secure, fast, and energy-efficient outsourced authentication for smartphones. *Trans. Info. For. Sec.*, 11(11):2556–2571, November 2016.

HKE13. Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, Heidelberg, August 2013.

HMQ04. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Heidelberg, February 2004.

JW99. Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *ACM CCS 99*, pages 28–36. ACM Press, November 1999.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and

Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

KR08. Vladimir Kolesnikov and Charles Rackoff. Password mistyping in two-factor-authenticated key exchange. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 702–714. Springer, Heidelberg, July 2008.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

KV11. Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, March 2011.

Lin13. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2013.

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Heidelberg, March 2011.

LP15. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Journal of Cryptology*, 28(2):312–350, April 2015.

LR14. Yehuda Lindell and Ben Riva. Cut-and-choose based two-party computation in the online/offline and batch settings. Cryptology ePrint Archive, Report 2014/667, 2014. http://eprint.iacr.org/2014/667.

Mau97. Ueli M. Maurer. Information-theoretically secure secret-key agreement by NOT authenticated public discussion. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 209–225. Springer, Heidelberg, May 1997.

MG09. Rene Mayrhofer and Hans Gellersen. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing*, 8(6):792–806, 2009.

MRW02. Fabian Monrose, Michael K Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security*, 1(2):69–83, 2002.

MS81. Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, 1981.

NZ93. Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In *25th ACM STOC*, pages 235–244. ACM Press, May 1993.

PRTG02. Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

Rot06. Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, New York, NY, USA, 2006.

RW04. Renato Renner and Stefan Wolf. The exact price for unconditionally secure asymmetric cryptography. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 109–125. Springer, Heidelberg, May 2004.

SD07. G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

Sho01. Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. http://eprint.iacr.org/2001/112.

TSS+06. Pim Tuyls, Geert Jan Schrijen, Boris Skoric, Jan van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In Louis Goubin and Mitsuru Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 369–383. Springer, Heidelberg, October 2006.

WCD+17. Joanne Woodage, Rahul Chatterjee, Yevgeniy Dodis, Ari Juels, and Thomas Ristenpart. A new distribution-sensitive secure sketch and popularity-proportional hashing. In *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 682–710. Springer, 2017.

Wyn75. Aaron D. Wyner. The wire-tap channel. *The Bell System Technical Journali*, 54, October 1975.

Yak17. Sophia Yakoubov. A gentle introduction to yao's garbled circuits, 2017. http://web.mit.edu/sonka89/www/papers/2017ygc.pdf.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

YD10. Meng-Day Mandel Yu and Srinivas Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Design & Test*, 27(1):48–65, 2010.

ZH93. Moshe Zviran and William J. Haga. A comparison of password techniques for multilevel authentication mechanisms. *The Computer Journal*, 36(3):227–237, 1993.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

Supplementary Material

A Ideal UC Functionalities

Common Reference String. The Common Reference String (CRS) functionality was already defined in [Can07]. We recall it in Figure 12 for completeness. Note that we do not let $\mathcal{F}_{\mathsf{CRS}}$ check whether a party is allowed to obtain the CRS — it is assumed public.

The functionality $\mathcal{F}_{\mathsf{CRS}}^{\mathcal{D}}$ is parameterized with a distribution $\mathcal{D}$ and proceeds as follows:

- Upon receiving $(\mathsf{sid}, \mathtt{crs})$:
  - If there is no value r recorded, then choose and record a value $r \xleftarrow{\$} \mathcal{D}$.
  - Reply with (sid, r).

Fig. 12. Functionality $\mathcal{F}_{\mathsf{CRS}}$

Random Oracles. The Random Oracle (RO) functionality was already defined by Hofheinz and Müller-Quade in [HMQ04]. We recall it in Figure 13 for completeness. It is clear that the random oracle model UC-emulates this functionality.

The functionality $\mathcal{F}_{\mathsf{RO}}$ proceeds as follows, running on security parameter k, with a set of (dummy) parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and an adversary $\mathcal{S}$:

- $\mathcal{F}_{\mathsf{RO}}$ keeps a list L (which is initially empty) of pairs of bit strings.
- Upon receiving a value (sid, m) (with $m \in \{0,1\}^*$) from some party $\mathcal{P}_i$ or from $\mathcal{S}$, do:
  - If there is a pair $(m, \tilde{h})$ for some $\tilde{h} \in \{0,1\}^k$ in the list L, set $h := \tilde{h}$.
  - If there is no such pair, choose uniformly $h \in \{0,1\}^k$ and store the pair $(m, h) \in L$.

  Once h is set, reply to the activating machine (i.e., either $\mathcal{P}_i$ or $\mathcal{S}$) with (sid, h).

Fig. 13. Functionality $\mathcal{F}_{\mathsf{RO}}$

Ideal Cipher. An ideal cipher [BPR00] is a block cipher that takes a plaintext or a ciphertext as input. We describe the ideal cipher functionality $\mathcal{F}_{\mathsf{IC}}$ in Figure 14, in the same vein as the above random oracle functionality. It is clear

that the ideal cipher model UC-emulates this functionality. Note that this functionality characterizes a perfectly random permutation for each key by ensuring injectivity for each query simulation: to this aim, it uses a list L and projections M_{sk} and C_{sk}, that are global, independently of the sid.

The functionality $\mathcal{F}_{\mathsf{IC}}$ takes as input the security parameter k, and interacts with an adversary $\mathcal{S}$ and with a set of (dummy) parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ by means of these queries:

- $\mathcal{F}_{\mathsf{IC}}$ keeps a (initially empty) list L containing 3−tuples of bit strings and two (initially empty) sets C_{sk} and M_{sk} for every sk. (The sets are not created until sk is first used, thus avoiding the need to instantiate exponentially many sets.)
- **Upon receiving a query $(\mathsf{sid}, \mathcal{E}, \mathsf{sk}, m)$ (with $m \in \{0,1\}^k$) from some party $\mathcal{P}_i$ or $\mathcal{S}$, do:**
  - If there is a 3−tuple $(\mathsf{sk}, m, \tilde{c})$ for some $\tilde{c} \in \{0,1\}^k$ in the list L, set $c := \tilde{c}$.
  - If there is no such record, choose uniformly $c \in \{0,1\}^k \backslash C_{\mathsf{sk}}$ which is the set consisting of ciphertexts not already used with sk. Next, it stores the 3−tuple $(\mathsf{sk}, m, c) \in L$ and sets both $M_{\mathsf{sk}} \leftarrow M_{\mathsf{sk}} \cup \{m\}$ and $C_{\mathsf{sk}} \leftarrow C_{\mathsf{sk}} \cup \{c\}$.

  Once c is set, reply to the activating machine with (sid, c).
- **Upon receiving a query $(\mathsf{sid}, \mathcal{D}, \mathsf{sk}, c)$ (with $c \in \{0,1\}^k$) from some party $\mathcal{P}_i$ or $\mathcal{S}$, do:**
  - If there is a 3−tuple $(\mathsf{sk}, \tilde{m}, c)$ for some $\tilde{m} \in \{0,1\}^k$ in L, set $m := \tilde{m}$.
  - If there is no such record, choose uniformly $m \in \{0,1\}^k \backslash M_{\mathsf{sk}}$ which is the set consisting of plaintexts not already used with sk. Next, it stores the 3−tuple $(\mathsf{sk}, m, c) \in L$ and sets both $M_{\mathsf{sk}} \leftarrow M_{\mathsf{sk}} \cup \{m\}$ and $C_{\mathsf{sk}} \leftarrow C_{\mathsf{sk}} \cup \{c\}$.

  Once m is set, reply to the activating machine with (sid, m).

Fig. 14. Functionality $\mathcal{F}_{\mathsf{IC}}$

Oblivious Transfer. The Oblivious Transfer (OT) functionality was defined by Canetti *et al.* [CLOS02]. We recall it in Figure 15.

The functionality $\mathcal{F}_{\mathsf{OT}}$ is parameterized by a security parameter λ. It interacts with an adversary $\mathcal{S}$ and the players $\mathcal{S}$ (the sender) and $\mathcal{R}$ (the receiver) via the following queries:

- **Upon receiving a query $(\mathtt{Send}, \mathsf{sid}, x_0, x_1)$ from $\mathcal{S}$**, where $x_0, x_1 \in \{0,1\}^\lambda$, record the tuple (x_0, x_1).
- **Upon receiving a query $(\mathtt{Receive}, \mathsf{sid}, i)$ from $\mathcal{R}$:**
  If there is a record (x_0, x_1), then send (sid, x_i) to $\mathcal{R}$ and sid to $\mathcal{S}$, and halt. Otherwise, ignore the query.

Fig. 15. Functionality $\mathcal{F}_{\mathsf{OT}}$

Password-Authenticated Key Exchange. The initial PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ has been defined by Canetti *et al.* [CHK+05]. We recall it in Figure 16. We

stress that this functionality immediately leaks the result of the `TestPwd`-query, which models explicit authentication; when the adversary tries a password, it learns whether the guess was correct or not.

The functionality $\mathcal{F}_{\mathsf{pwKE}}$ is parameterized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of (dummy) parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \texttt{role})$ from party $\mathcal{P}_i$:**
  - Send $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \texttt{role})$ to $\mathcal{S}$;
  - If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}')$, then record $(\mathcal{P}_i, \mathcal{P}_j, \mathsf{pw})$ and mark this record `fresh`.
- **Upon receiving a query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}')$ from $\mathcal{S}$:**
  If there is a `fresh` record $(\mathcal{P}_i, \mathcal{P}_j, \mathsf{pw})$, then do:
  - If $\mathsf{pw} = \mathsf{pw}'$, mark the record `compromised`, and reply to $\mathcal{S}$ with *correct guess*;
  - If $\mathsf{pw} \neq \mathsf{pw}'$, mark the record `interrupted`, and reply to $\mathcal{S}$ with *wrong guess*.
- **Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{sk})$ from the $\mathcal{S}$, where $|\mathsf{sk}| = k$:**
  If there is a record of the form $(\mathcal{P}_i, \mathcal{P}_j, \mathsf{pw})$, and this is the first `NewKey` query for $\mathcal{P}_i$, then:
  - If this record is `compromised`, or either $\mathcal{P}_i$ or $\mathcal{P}_j$ is corrupted, then output $(\mathsf{sid}, \mathsf{sk})$ to player $\mathcal{P}_i$;
  - If this record is `fresh`, there is a record $(\mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}')$ with $\mathsf{pw} = \mathsf{pw}'$, a key sk' was sent to $\mathcal{P}_j$, and $(\mathcal{P}_j, \mathcal{P}_i, \mathsf{pw})$ was `fresh` at the time, then output $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$;
  - In any other case, pick a new random key sk' of length k and send $(\mathsf{sid}, \mathsf{sk}')$ to $\mathcal{P}_i$.

  Either way, mark the record $(\mathcal{P}_i, \mathcal{P}_j, \mathsf{pw})$ as `completed`.

Fig. 16. Functionality $\mathcal{F}_{\mathsf{pwKE}}$

In our paper, we start from this functionality to derive the basic functionality fPAKE, in Section 2, after a few changes:

- we consider only two parties — $\mathcal{P}_0$ and $\mathcal{P}_1$ —, which is enough since universal composability takes care of ensuring that a two-party functionality remains secure in a multi-party world;
- we do not allow the adversary to set $\mathcal{P}_i$'s key if $\mathcal{P}_{1-i}$ is corrupted but did not guess $\mathcal{P}_i$'s password. We make this change in order to protect an honest $\mathcal{P}_i$ from, for instance, revealing sensitive information to an adversary who did not successfully guess her password, but did corrupt her partner.

B Garbled Output Randomness: A New Yao's Garbled Circuit Definition

We refer to Yakoubov [Yak17] for a gentle introduction to Yao's Garbled Circuits. Note that the authenticity property implies that, in an output-projective garbling scheme, if the output is a single bit, the second output label and the second token of d are hard for the evaluator to guess (no probabilistic polynomial-time adversary can guess it with non-negligible probability). However, for our fPAKE construction (Section 3), we require a stronger property: not only should the second output label be hard to guess, but it should be *indistinguishable from random*. We call this *garbled-output randomness*.

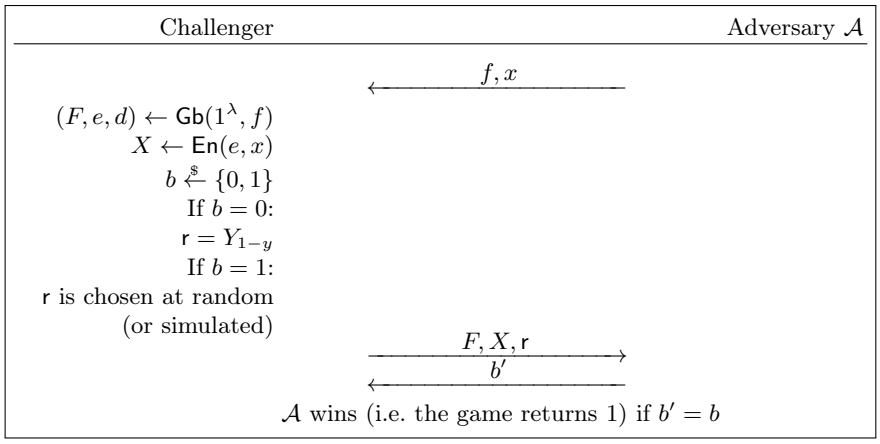

Fig. 17. The $\mathsf{GOutRand}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda)$ Game, where $y = f(x) \in \{0,1\}$, Y_y is the corresponding garbled output (or output label), and Y_{1-y} is the other output label.

Define the adversary $\mathcal{A}$'s advantage in the the garbled-output randomness game (Figure 17) as

$$\mathsf{GOutRandAdv}_{\mathcal{G}}(1^\lambda, \mathcal{A}) = \left| \Pr[\mathsf{GOutRand}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda) = 1] - \frac{1}{2} \right|.$$

Definition 7. *An output-projective binary output garbling scheme* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is* garbled-output random *if for all sufficiently large security parameters* λ*, for any polynomial time adversary* $\mathcal{A}$*,*

$$\mathsf{GOutRandAdv}_{\mathcal{G}}(1^\lambda, \mathcal{A}) = \mathsf{negl}\,.$$

In order to achieve this, we modify the scheme of Bal *et al.* [BMR16] to put the output wire label through the hash function H one more time; the two labels will thus be $Y_0 = H(\mathsf{finaloutput}, W^0_{output})$ and $Y_1 = H(\mathsf{finaloutput}, W^0_{output} \oplus R)$, where W^0_{output} and $W^0_{output} \oplus R$ were the labels in the scheme of Bal *et al.*

Theorem 8. *This modified scheme is garbled-output random (Definition 7) when the key derivation function H is mixed-modulus circular correlation robust (Definition 1 of Bal* et al. *[BMR16]).*

Proof (Proof Sketch).

Definition 7 requires the indistinguishability of (real garbled circuit and inputs, real second output label) and (real garbled circuit and inputs, random second output label); in shorthand, we want to show that $(real, real) \sim (real, random)$, where $\sim$ denotes computational indistinguishability.

We use (simulated garbled circuit and inputs, random second output label) — $(simulated, random)$ for short — as a hybrid step. We show that $(real, real) \sim (simulated, random)$. We do this by having the adversary $\mathcal{A}$ (modeled after the one in Choi *et al.* [CKKZ12]) compute the second output label $\mathcal{O}(\mathsf{finaloutput}, 2, 2, W^b_{output}, 1, 0)$ (using the mixed-modulus circular correlation robustness oracle) and send it to the obliviousness adversary $\mathcal{B}$ along with the garbled circuit and garbled inputs. If the oracle is random, $\mathcal{B}$ will see $(simulated, random)$. Otherwise, $\mathcal{B}$ will see $(real, real)$. If $\mathcal{B}$ can distinguish between those two, then $\mathcal{A}$ can use that to break mixed-modulus circular correlation robustness. Hence, $(real, real) \sim (simulated, random)$.

Because the garbling scheme is oblivious, we know that $(simulated, random) \sim (real, random)$, since we can always add a random value to the adversary's view in the obliviousness game.

Now that we have $(real, real) \sim (simulated, random)$ and $(simulated, random) \sim (real, random)$, we can conclude that $(real, real) \sim (real, random)$.

C Proof of Theorem 1

We proceed in a series of games, where no probabilistic polynomial-time environment can distinguish the view of the adversary $\mathcal{A}$ in each game from that in the previous game. We start with the real execution of the protocol and end with the ideal execution. Figure 18 summarizes the changes made in each game.

Game $\mathbf{G}_0$: Real
This is the real execution of Π_{RFE} where the environment $\mathcal{Z}$ runs the protocol (described in Figure 4) with parties $\mathcal{P}_0$ and $\mathcal{P}_1$, both having access to an ideal OT functionality $\mathcal{F}_{\mathsf{OT}}$, and an adversary $\mathcal{A}$ that, w.l.o.g., can be assumed to be the dummy adversary as shown in [Can01, section 4.4.1].

Game $\mathbf{G}_1$: Adding Ideal Layout
This is the real game, but with dummy party and ideal functionality nodes thrown in and all previously existing nodes (except the environment) grouped into one machine, called the simulator ($\mathcal{S}_{\mathsf{RFE}}$, or $\mathcal{S}$ for short). Please refer to Figure 19 for the differences between $\mathbf{G}_0$ and $\mathbf{G}_1$.

Game $\mathbf{G}_2$: Adding $\mathcal{F}$'s Record-Keeping and `TestPwd` Interface
Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to do all of the record-keeping described in Figure 3.

	Game	Functionality $\mathcal{F}$			$\mathcal{S}_{\mathsf{RFE}}$	Property Used
		`NewSession`	`TestPwd`	`NewKey`		
	Game $\mathbf{G}_0$	N/A	N/A	N/A	N/A	
	Game $\mathbf{G}_1$	forwards inputs to $\mathcal{S}_{\mathsf{RFE}}$		forwards outputs to dummy parties	runs protocol for honest parties	
	Game $\mathbf{G}_2$	records inputs			creates `NewKey` queries from party outputs	
both parties honest	Game $\mathbf{G}_3$			chooses keys for both parties when $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$		garbled output randomness
	Game $\mathbf{G}_4$			chooses key for $\mathcal{P}_0$ when $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$		garbled output randomness
	Game $\mathbf{G}_5$			chooses keys for both parties when $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$		garbled output randomness
	Game $\mathbf{G}_6$				simulates F_0, X_0	obliviousness
	Game $\mathbf{G}_7$	does not forward $\mathsf{pw}_0, \mathsf{pw}_1$			simulates F_1, X_1	obliviousness
$\mathcal{P}_i$ honest, $\mathcal{P}_{1-i}$ corrupt	Game $\mathbf{G}_8$	replaces the malicious `NewSession` input with the one given by $\mathcal{S}_{\mathsf{RFE}}$			extracts malicious pw'_{1-i} from OT, and tells $\mathcal{F}$ to replace the malicious `NewSession` input with pw'_{1-i}	
	Game $\mathbf{G}_9$			chooses key for $\mathcal{P}_i$ when $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$ (now fully implemented)		garbled output randomness
	Game $\mathbf{G}_{10}$				simulates F_i, X_i using $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i})$	privacy
	Game $\mathbf{G}_{11}$	does not forward pw_i	fully implemented		makes `TestPwd` query to set pw'_i	

Fig. 18. A Summary of the Sequence of Games in the Proof of Theorem 1

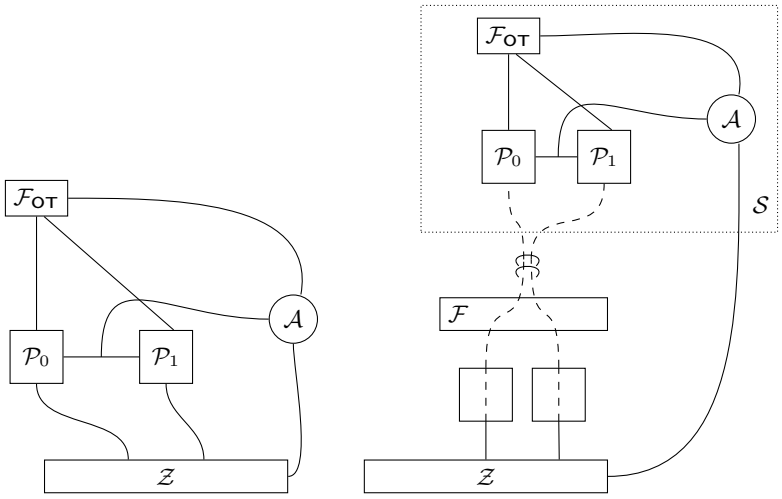

Fig. 19. Transition from game $\mathbf{G}_0$ (left) to game $\mathbf{G}_1$ (right), showing a setting where both parties are honest.

$\mathcal{F}$ still forwards `NewSession` queries from the dummy parties in their entirety (including the pass-string) to $\mathcal{S}_{\mathsf{RFE}}$, but also records them. Since this is a matter of internal record-keeping only, this does not affect $\mathcal{A}$'s view.
Modifications to $\mathcal{S}_{RFE}$: $\mathcal{S}_{\mathsf{RFE}}$ creates `NewKey` queries for $\mathcal{F}$ from whatever output the simulated parties produce. In this game, $\mathcal{F}$ still simply forwards the keys it is given to the dummy parties without modifying them, so this does not affect $\mathcal{A}$'s view.

Game $\mathbf{G}_3$: Allowing $\mathcal{F}$ to Choose Keys For Two Honest Parties With Close Pass-strings
Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Figure 3 to choose the key when $\mathcal{P}_0$ and $\mathcal{P}_1$ are both honest, and $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$.
We can use any environment who can distinguish this game from Game $\mathbf{G}_2$ to build an adversary $\mathcal{B}$ that can break the garbled output randomness property (Definition 7, Theorem 8) of our garbling scheme.
Since both parties are honest, in order to use the environment as a distinguisher, the adversary needs to give the environment a transcript of the parties' interactions $(F_0, X_{0,0}, F_1, X_{1,1})$ as well as the parties' output keys. Because we are in the OT hybrid model, the environment sees neither inputs to the OT nor its outputs.
Our adversary $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_0$ with the $\mathcal{F}$ of Game $\mathbf{G}_2$ with some modifications. First, it finds a pass-string pw such that $d(\mathsf{pw}_0, \mathsf{pw}) > \delta$. (Note that in order for this reduction to work, such a pass-string must be efficiently computable, which it is by the assumption in the statement of Theorem 1.) Instead of running Gb, $\mathcal{B}$ queries the garbled output randomness challenger on $(f, (\mathsf{pw}_0, \mathsf{pw}))$ to obtain (F_0, X_0, r). Let $X_0 = (X_{0,0}, X_{0,1})$. Note that F_0 and $X_{0,0}$ are generated by the challenger exactly as they would be by

$\mathcal{S}_{\mathsf{RFE}}$, so those values do not change. ($X_{0,1}$ is different, since pw is different from pw_1, but $X_{0,1}$ is not visible to the environment.) Since the adversary used a pass-string pw that is dissimilar to pw_1, it uses the value r — corresponding to the output label *not* returned by $\mathsf{Ev}(F_0, X_0)$ — as $\mathsf{k}_{0,correct}$. ($\mathsf{Ev}(F_0, X_0)$ would give $\mathsf{k}_{0,wrong}$.) If $b = 0$, the challenger will return the actual $\mathsf{k}_{0,correct}$ as r, and the environment's view will be that of Game $\mathbf{G}_2$. If $b = 1$, the challenger will give a random value as r. If r is truly random, then so is $\mathsf{r} \oplus Y_1$; so, the environment's view will be that of Game $\mathbf{G}_3$. (Note that we do not change the way in which $\mathcal{P}_1$ generates F_1, X_1, but we do set $\mathcal{P}_1$'s output key to be the same as $\mathcal{P}_0$'s. This will be the key that an honest execution of the protocol would produce if $b = 0$, and random otherwise.) The adversary $\mathcal{B}$ then returns the environment's guess as b'. The advantage of $\mathcal{B}$ in the garbled output randomness game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_2$ and Game $\mathbf{G}_3$.

Game $\mathbf{G}_4$: Allowing $\mathcal{F}$ to Choose Keys For One of Two Honest Parties With Dissimilar Pass-strings

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Figure 3 to choose the key for $\mathcal{P}_0$ when $\mathcal{P}_0$ and $\mathcal{P}_1$ are both honest, and $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$.

We can use any environment who can distinguish this game from Game $\mathbf{G}_3$ to build an adversary $\mathcal{B}$ that can break the garbled output randomness property (Definition 7, Theorem 8) of our garbling scheme.

Our adversary $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_0$ with the $\mathcal{F}$ of Game $\mathbf{G}_3$ with some modifications. Instead of running Gb, $\mathcal{B}$ queries the garbled output randomness challenger on $(f, (\mathsf{pw}_0, \mathsf{pw}_1))$ to obtain (F_0, X_0, r). Note that F_0 and X_0 are generated by the challenger exactly as they would be by $\mathcal{S}_{\mathsf{RFE}}$, so these values do not change. If $b = 0$, the challenger will return the actual $\mathsf{k}_{0,correct}$ as r, and the environment's view will be that of Game $\mathbf{G}_3$. If $b = 1$, the challenger will give a random value as r. If r is truly random, then so is $\mathsf{r} \oplus Y_1$; so, the environment's view will be that of Game $\mathbf{G}_4$. The adversary $\mathcal{B}$ then returns the environment's guess as b'. The advantage of $\mathcal{B}$ in the garbled output randomness game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_3$ and Game $\mathbf{G}_4$.

Game $\mathbf{G}_5$: Allowing $\mathcal{F}$ to Choose Keys For Both Honest Parties With Dissimilar Pass-strings

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Figure 3 to choose the key for $\mathcal{P}_1$ as well as for $\mathcal{P}_0$ when $\mathcal{P}_0$ and $\mathcal{P}_1$ are both honest, and $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$.

We can use any environment who can distinguish this game from Game $\mathbf{G}_4$ to build an adversary that can break the garbled output randomness property (Definition 7, Theorem 8) of our garbling scheme, exactly as we did in the reduction above.

Game $\mathbf{G}_6$: Simulating F, X for One of Two Honest Parties

Modifications to $\mathcal{S}_{RFE}$: Consider the case when both $\mathcal{P}_0$ and $\mathcal{P}_1$ are honest. In this game, the simulator replaces $\mathcal{P}_0$'s garbled circuit and input with simulated ones. $\mathcal{S}_{\mathsf{RFE}}$ does not need to simulate anything relating to the OT, since the environment cannot observe OT functionality inputs or outputs if

both participating parties are honest. $\mathcal{S}_{\mathsf{RFE}}$ uses the obliviousness simulator to generate F_0, X_0 (while continuing to generate F_1, X_1 honestly), and sends the garbled circuits and the appropriate parts of the garbled inputs between the parties. $\mathcal{S}_{\mathsf{RFE}}$ outputs $\perp$ bot as both parties' keys, since the outputs don't matter - $\mathcal{F}$ takes care of outputting appropriate keys as of a few games ago (Game $\mathbf{G}_3$ if $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, and Games $\mathbf{G}_4$,$\mathbf{G}_5$ otherwise), so this change is not observable by the environment.

We can use any environment who can distinguish this game from Game $\mathbf{G}_5$ to build an adversary $\mathcal{B}$ that can break the obliviousness property of our garbling scheme. $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_0$ as in Game $\mathbf{G}_5$, but instead of generating (F_0, X_0) and (F_1, X_1) according to the protocol, it queries the obliviousness challenger on $(f, (\mathsf{pw}_0, \mathsf{pw}_1))$ to obtain (F_0, X_0). If $b = 0$, the challenger will return actual (F_i, X_i) values, and the environment's view we will be that of Game $\mathbf{G}_5$. If $b = 1$, the challenger will return simulated values, and the environment's view will be that of this game. The adversary $\mathcal{B}$ then returns the environment's guess as b'. The advantage of $\mathcal{B}$ in the obliviousness game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_5$ and Game $\mathbf{G}_6$.

Game $\mathbf{G}_7$: Removing Pass-string Forwarding Always

Modifications to $\mathcal{F}$: We now modify $\mathcal{F}$ to forward only $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}_{\mathsf{RFE}}$ (omitting the pass-string pw_i) for $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$ when $\mathcal{P}_0$ and $\mathcal{P}_1$ are both honest.

Modifications to $\mathcal{S}_{RFE}$: In Game $\mathbf{G}_6$, $\mathcal{S}_{\mathsf{RFE}}$ started simulating $\mathcal{P}_0$'s messages without using knowledge of pw_0. $\mathcal{S}_{\mathsf{RFE}}$ now also simulates $\mathcal{P}_1$'s messages by using the obliviousness simulator to generate the garbled circuit and input F_1, X_1.

We can use any environment who can distinguish this game from Game $\mathbf{G}_6$ to build an adversary $\mathcal{B}$ that can break the obliviousness property of our garbling scheme. $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_1$ as in Game $\mathbf{G}_6$, but instead of generating (F_1, X_1) according to the protocol, it queries the obliviousness challenger on $(f, (\mathsf{pw}_1, \mathsf{pw}_0))$ to obtain (F_1, X_1). If $b = 0$, the challenger will return actual (F_1, X_1) values, and the environment's view we will be that of Game $\mathbf{G}_6$. If $b = 1$, the challenger will return simulated values, and the environment's view will be that of this game. The adversary $\mathcal{B}$ then returns the environment's guess as b'. The advantage of $\mathcal{B}$ in the obliviousness game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_6$ and Game $\mathbf{G}_7$.

Game $\mathbf{G}_8$: Setting the Malicious Input as in the "Standard Corruption Model"

Modifications to $\mathcal{S}_{RFE}$: In this game, $\mathcal{S}_{\mathsf{RFE}}$ sets a corrupt party $\mathcal{P}_{1-i}$'s `NewSession` input according to the standard corruption model [Can01]. It does so as soon as it sees pw'_{1-i}, when it is given as an input to the ideal OT functionality. Since $\mathcal{F}$ does not currently use pw_{1-i}, this does not affect the environment's view.

Game $\mathbf{G}_9$: Allowing $\mathcal{F}$ to Choose the Key For An Honest Party With a Pass-string Dissimilar to Its Corrupt Partners'

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Figure 3 to choose all keys. Note that the only remaining scenario this affects is the one where only one party $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$ is honest, and $d(\mathsf{pw}_0, \mathsf{pw}_1) > \delta$. If both parties are corrupt, or if only one party is corrupt and $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, $\mathcal{F}$ still simply forwards the output key.

We can use any environment who can distinguish this game from Game $\mathbf{G}_8$ to build an adversary $\mathcal{B}$ that can break the garbled output randomness property (Definition 7, Theorem 8) of our garbling scheme. Our adversary $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_0$ with the $\mathcal{F}$ of Game $\mathbf{G}_8$ with some modifications. Instead of running Gb first, it waits to see $\mathcal{P}_{1-i}$'s input pw'_{1-i} to the ideal OT functionality, and then queries the garbled output randomness challenger on $(f, (\mathsf{pw}_i, \mathsf{pw}'_{1-i}))$ to obtain (F_i, X_i, r). Note that F_i and X_i are generated by the challenger exactly as they would be by $\mathcal{S}_{\mathsf{RFE}}$, so those values do not change. The adversary then uses r as $\mathsf{k}_{i,correct}$. If $b = 0$, the challenger will return the actual $\mathsf{k}_{i,correct}$ as r, and the environment's view will be that of Game $\mathbf{G}_8$. If $b = 1$, the challenger will give a random value as r. If r is truly random, then so is $\mathsf{r} \oplus Y_{1-i}$; so, the environment's view will be that of Game $\mathbf{G}_9$. The adversary $\mathcal{B}$ then returns the environment's guess as b'. The advantage of $\mathcal{B}$ in the garbled output randomness game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_8$ and Game $\mathbf{G}_9$.

Game $\mathbf{G}_{10}$: Simulating Garbled Circuit and Inputs For An Honest Party With a Corrupt Partner

Modifications to $\mathcal{S}_{RFE}$: In this game, $\mathcal{S}_{\mathsf{RFE}}$ simulates F_i and X_i when $\mathcal{P}_i$ is honest and $\mathcal{P}_{1-i}$ is corrupt.

In more detail, $\mathcal{S}_{\mathsf{RFE}}$ proceeds as follows on behalf of $\mathcal{P}_i$:

- $\mathcal{S}_{\mathsf{RFE}}$ postpones step 1.
- In step 2, $\mathcal{S}_{\mathsf{RFE}}$:
  - Plays an OT sender as follows:
    - Waits for $\mathcal{P}_{1-i}$ to provide their select bits to the OT. As a result, $\mathcal{S}_{\mathsf{RFE}}$ learns the pass-string used by $\mathcal{P}_{1-i}$, pw'_{1-i}.
    - If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$ sets $y = 1$, and sets $y = 0$ otherwise.
    - Uses the privacy simulator for the garbling scheme to generate $(F_i, X_i, d_i) \leftarrow \mathcal{S}(1^\lambda, f, y)$.
    - Parses $(X_{i,i}, X_{i,1-i}) \leftarrow X_i$.
    - Sends $X_{i,1-i}$ to $\mathcal{P}_{1-i}$ as the OT output.
  - Plays an OT receiver honestly.
- $\mathcal{S}_{\mathsf{RFE}}$ follows the instructions in Figure 4 for steps 3-8.

We can use any environment who can distinguish this game from Game $\mathbf{G}_9$ to build an adversary $\mathcal{B}$ that can break the privacy property of our garbling scheme. Our adversary $\mathcal{B}$ executes $\mathcal{S}_{\mathsf{RFE}}$'s simulation of $\mathcal{P}_0$ with some modifications. Instead of running the privacy simulator $\mathcal{S}$, it queries the privacy challenger on $(f, (\mathsf{pw}_i, \mathsf{pw}'_{1-i}))$ to obtain (F_i, X_i, d_i). If $b = 0$, the challenger will return actual (F_i, X_i, d_i) values, and the environment's view will be that of Game $\mathbf{G}_9$; if $b = 1$, the challenger return simulated values, and the environment's view will be that of this game. The adversary $\mathcal{B}$ then returns

the environment's guess as b'. The advantage of $\mathcal{B}$ in the privacy game will be exactly the same as that of the environment in distinguishing between Game $\mathbf{G}_9$ and Game $\mathbf{G}_{10}$.

Game $\mathbf{G}_{11}$: Removing Pass-string Forwarding To An Honest Party With a Corrupt Partner

Modifications to $\mathcal{F}$:

- If $\mathcal{P}_i$ is honest and $\mathcal{P}_{1-i}$ is corrupt, then, upon receiving a `NewSession` query, $\mathcal{F}$ forwards only $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}_{\mathsf{RFE}}$ (omitting the pass-string pw_i).
- $\mathcal{F}$ now processes `TestPwd` queries (which were not issued in any prior game) according to the instructions in Figure 3. Given a $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i)$ query ($\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$), if $d(\mathsf{pw}_0, \mathsf{pw}_1) \leq \delta$, $\mathcal{F}$ sends pw_i to $\mathcal{S}_{\mathsf{RFE}}$.

Modifications to $\mathcal{S}_{\mathsf{RFE}}$: Now that $\mathcal{S}_{\mathsf{RFE}}$ does not know $\mathcal{P}_i$'s pass-string, it must simulate the honest party's messages without that knowledge.

In more detail, $\mathcal{S}_{\mathsf{RFE}}$ proceeds as follows on behalf of $\mathcal{P}_i$:

- $\mathcal{S}_{\mathsf{RFE}}$ postpones step 1.
- In step 2, $\mathcal{S}_{\mathsf{RFE}}$:
  - Plays an OT sender as follows:
    - As in Game $\mathbf{G}_{10}$, waits for $\mathcal{P}_{1-i}$ to provide their select bits to the OT. As a result, $\mathcal{S}_{\mathsf{RFE}}$ learns the pass-string used by $\mathcal{P}_{1-i}$, pw'_{1-i}.
    - Makes a $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i)$ query to $\mathcal{F}$. If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$ (that is, the adversary has approximately guessed $\mathcal{P}_i$'s pass-string), $\mathcal{S}_{\mathsf{RFE}}$ learns pw_i, and sets $\mathsf{pw}'_i = \mathsf{pw}_i$. Otherwise, it sets pw'_i at random such that $d(\mathsf{pw}'_i, \mathsf{pw}'_{1-i}) > \delta$, and uses pw'_i in place of pw_i in the rest of the simulation.
    - As in Game $\mathbf{G}_{10}$, If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$ sets $y = 1$, and sets $y = 0$ otherwise.
    - As in Game $\mathbf{G}_{10}$, uses the privacy simulator for the garbling scheme to generate $(F_i, X_i, d_i) \leftarrow \mathcal{S}(1^\lambda, f, y)$.
    - As in Game $\mathbf{G}_{10}$, parses $(X_{i,i}, X_{i,1-i}) \leftarrow X_i$.
    - As in Game $\mathbf{G}_{10}$, sends $X_{i,1-i}$ to $\mathcal{P}_{1-i}$ as the OT output.
  - Plays an OT receiver honestly with pw'_i.
- $\mathcal{S}_{\mathsf{RFE}}$ follows the instructions in Figure 4 for steps 3-8.

Nothing could have changed from the point of view of the environment. If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$, this game is identical to Game $\mathbf{G}_{10}$. If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) > \delta$, a random pw'_i is used instead of pw_i. However, pw'_i only affects the OT execution with $\mathcal{P}_i$ as receiver, where $\mathcal{P}_{1-i}$ does not receive any outputs anyway. In this case, $\mathcal{P}_i$'s output key gets set randomly by $\mathcal{F}$ as of Game $\mathbf{G}_9$, so that does not change.

In Figure 20, we show the simulator $\mathcal{S}_{\mathsf{RFE}}$ for Π_{RFE}.

D Proof that $s\mathcal{F}^P_{\mathsf{RFE}}$ is Enough to Realize $\mathcal{F}^P_{\mathsf{fPAKE}}$

In Figure 21, we describe a protocol $\mathsf{fPAKE}_{\mathsf{YGC}}$ which trivially realizes $\mathcal{F}^P_{\mathsf{fPAKE}}$ in the $s\mathcal{F}^P_{\mathsf{RFE}}$-hybrid model.

- Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{F}^P_{\mathbf{RFE}}$ for honest party $\mathcal{P}_i$ when $\mathcal{P}_{1-i}$ is also honest, $\mathcal{S}_{\mathbf{RFE}}$ generates F_i, X_{1-i} using the obliviousness simulator for the garbling scheme, and sends those to $\mathcal{P}_{1-i}$.
- Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{F}^P_{\mathbf{RFE}}$ for honest party $\mathcal{P}_i$ when $\mathcal{P}_{1-i}$ is corrupt, $\mathcal{S}_{\mathbf{RFE}}$ does the following:
  - $\mathcal{S}_{\mathbf{RFE}}$ postpones step 1.
  - In step 2, $\mathcal{S}_{\mathbf{RFE}}$:
    - Plays an OT sender as follows:
      - Waits for $\mathcal{P}_{1-i}$ to provide their select bits to the OT. As a result, $\mathcal{S}_{\mathbf{RFE}}$ learns the pass-string used by $\mathcal{P}_{1-i}$, pw'_{1-i}.
      - Makes a $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i)$ query to $\mathcal{F}$. If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$ (that is, the adversary has approximately guessed $\mathcal{P}_i$'s pass-string), $\mathcal{S}_{\mathbf{RFE}}$ learns pw_i, and sets $\mathsf{pw}'_i = \mathsf{pw}_i$. Otherwise, it sets pw'_i at random such that $d(\mathsf{pw}'_i, \mathsf{pw}'_{1-i}) > \delta$, and uses pw'_i in place of pw_i in the rest of the simulation.
      - If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$ sets $y = 1$, and sets $y = 0$ otherwise.
      - Uses the privacy simulator for the garbling scheme to generate $(F_i, X_i, d_i) \leftarrow \mathcal{S}(1^\lambda, f, y)$.
      - Parses $(X_{i,i}, X_{i,1-i}) \leftarrow X_i$.
      - Sends $X_{i,1-i}$ to $\mathcal{P}_{1-i}$ as the OT output.
    - Plays an OT receiver honestly with pw'_i.
  - $\mathcal{S}_{\mathbf{RFE}}$ follows the instructions in Figure 4 for steps 3-8 with pw'_i.

Additionally, $\mathcal{S}_{\mathbf{RFE}}$ forwards all other instructions from $\mathcal{Z}$ to $\mathcal{A}$ and reports all output of $\mathcal{A}$ towards $\mathcal{Z}$. Instructions of corrupting a player are only obeyed if they are received before the protocol started, i.e., before $\mathcal{S}$ received any $\texttt{NewSession}$ query from $\mathcal{F}^P_{\mathbf{RFE}}$.

Fig. 20. Simulator $\mathcal{S}_{\mathsf{RFE}}$ for Π_{RFE}

Upon receiving the input $(\mathsf{sid}, \mathsf{pw}_i)$, $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$ does the following:

- Sends $(\texttt{Init}, \mathsf{sid})$ to $s\mathcal{F}^P_{\mathsf{RFE}}$;
- Sends $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ to $s\mathcal{F}^P_{\mathsf{RFE}}$;
- Waits for k from $s\mathcal{F}^P_{\mathsf{RFE}}$, and
- Outputs k.

Fig. 21. Protocol $\mathsf{fPAKE}_{\mathsf{YGC}}$ realizing $\mathcal{F}^P_{\mathsf{fPAKE}}$ in the $s\mathcal{F}^P_{\mathsf{RFE}}$-hybrid model.

Theorem 9. *Protocol $\mathit{fPAKE}_{\mathit{YGC}}$ realizes $\mathcal{F}^P_{\mathit{fPAKE}}$ in the $s\mathcal{F}^P_{\mathit{RFE}}$-hybrid model.*

Proof. For every efficient adversary $\mathcal{A}$, we describe a simulator $\mathcal{S}_{\mathsf{fPAKE}}$ in Figure 22 such that no efficient environment can distinguish an execution with the real protocol $\mathsf{fPAKE}_{\mathsf{YGC}}$ and $\mathcal{A}$ from an execution with the ideal functionality $\mathcal{F}^P_{\mathsf{fPAKE}}$ and $\mathcal{S}_{\mathsf{fPAKE}}$. Since the environment does not get any information about the honest parties except their output, all the simulator needs to do is respond to queries to $s\mathcal{F}^P_{\mathsf{RFE}}$. Since the honest party does nothing except query the ideal functionality $s\mathcal{F}^P_{\mathsf{RFE}}$, and its output gets replaced by values chosen by $\mathcal{F}^P_{\mathsf{fPAKE}}$, there is nothing to simulate.

All that remains to show is that the values produced by $\mathcal{F}^P_{\mathsf{fPAKE}}$ and by $s\mathcal{F}^P_{\mathsf{RFE}}$ are identically distributed. We describe the outputs of $\mathcal{F}^P_{\mathsf{fPAKE}}$ and $s\mathcal{F}^P_{\mathsf{RFE}}$ in Figure 23. The table enumerates all possible cases in the functionalities. Cases in $s\mathcal{F}^P_{\mathsf{RFE}}$ are described in terms of the distances between pass-strings: between the honest parties' pass-strings if no man in the middle attack occurred, and between adversarial and honest pass-strings if it did. (If the adversary engaged

$\mathcal{S}_{\mathsf{fPAKE}}$ responds to queries to $s\mathcal{F}^P_{\mathsf{RFE}}$ as follows:

- Upon getting $(\mathtt{Init}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of corrupt party $\mathcal{P}_{1-i} \in \{\mathcal{P}_0, \mathcal{P}_1\}$, $\mathcal{S}_{\mathsf{fPAKE}}$ does nothing.
- Upon getting $(\mathtt{Init}, \mathsf{sid}, \mathcal{P}_i, H, \mathsf{sid}_H)$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{fPAKE}}$ does nothing.
- Upon getting $(\mathtt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from $\mathcal{A}$ on behalf of honest party $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$, $\mathcal{S}_{\mathsf{fPAKE}}$ does nothing.
- Upon getting $(\mathtt{NewSession}, \mathsf{sid}, \mathsf{pw}'_{1-i})$ from $\mathcal{A}$ on behalf of corrupt party $\mathcal{P}_{1-i} \in \{\mathcal{P}_0, \mathcal{P}_1\}$, $\mathcal{S}_{\mathsf{fPAKE}}$:
  - Records pw'_{1-i};
  - Sends $(\mathtt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}'_{1-i})$ to $\mathcal{F}^P_{\mathsf{fPAKE}}$;
  - If $d(\mathsf{pw}_i, \mathsf{pw}'_{1-i}) \leq \delta$, $\mathcal{S}_{\mathsf{fPAKE}}$ learns pw_i.
- Upon getting a $(\mathtt{TestPwd}, \mathsf{sid}, \mathcal{P}_i)$ query from $\mathcal{A}$, $\mathcal{S}_{\mathsf{fPAKE}}$ responds with the output of the $\mathtt{TestPwd}$ query above.
- Upon getting a $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ query from $\mathcal{A}$, if $\mathcal{P}_i$ is corrupt, $\mathcal{S}_{\mathsf{fPAKE}}$ outputs k_i to $\mathcal{P}_i$. In any case, $\mathcal{S}_{\mathsf{fPAKE}}$ forwards $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ to $\mathcal{F}^P_{\mathsf{fPAKE}}$.

Additionally, $\mathcal{S}_{\mathsf{fPAKE}}$ forwards all other instructions from $\mathcal{Z}$ to $\mathcal{A}$ and reports all output of $\mathcal{A}$ towards $\mathcal{Z}$. Instructions of corrupting a player are only obeyed if they are received before the protocol started, i.e., before $\mathcal{S}$ received any $\mathtt{NewKey}$ query from $\mathcal{F}^P_{\mathsf{fPAKE}}$.

Fig. 22. Simulator $\mathcal{S}_{\mathsf{fPAKE}}$ for $\mathsf{fPAKE}_{\mathsf{YGC}}$.

one party but not the other, only one of the distances is filled in.) Cases in $\mathcal{F}^P_{\mathsf{fPAKE}}$ are described in terms of record markings ("fresh", "compromised" or "interrupted"). There is a one-to-one mapping between the cases in $s\mathcal{F}^P_{\mathsf{RFE}}$ and $\mathcal{F}^P_{\mathsf{fPAKE}}$ such that the outputs for the parties, whether they are honest or corrupt, are identically distributed. Those outputs are described in the last three columns of the table, as tuples of values the first of which is output to $\mathcal{P}_0$, and the second of which is output to $\mathcal{P}_1$. a, b are adversarially chosen values (which may or may not be distinct). r, s are independent, uniformly random values.

The transformation of Barak *et al.* proceeds in two steps. First, links are initialized:

1. Each party generates a signing and verification key pair, and sends the verification key to its partner.
2. Each party then signs the key it receives and sends the signature back.
3. Each party verifies the signature it receives on its own verification key using the verification key it received; if the signature does not verify, it aborts.

Second, the parties run the protocol exactly as they would over authenticated channels, signing each message with their signing key, and verifying each signature they receive.

Applying this transformation adds (1) two rounds of communication, and (2) a hash operation and a signature operation for each message, assuming the hash-and-sign paradigm is used.

$s\mathcal{F}^P_{\mathsf{RFE}}$			$\mathcal{F}^P_{\mathsf{fPAKE}}$		outputs in both $s\mathcal{F}^P_{\mathsf{RFE}}$ and $\mathcal{F}^P_{\mathsf{fPAKE}}$		
distance between $\mathcal{P}_0$'s pass-string and $\mathcal{P}_1$'s pass-string $d(\mathsf{pw}_0, \mathsf{pw}_1)$, if MITM didn't happen	distance between $\mathcal{P}_0$'s pass-string and the adversary's $d(\mathsf{pw}_0, \mathsf{pw}'_1)$, if MITM happened	distance between $\mathcal{P}_1$'s pass-string and the adversary's $d(\mathsf{pw}'_0, \mathsf{pw}_1)$, if MITM happened	$\mathcal{P}_0$'s record	$\mathcal{P}_1$'s record	$\mathcal{P}_0$ and $\mathcal{P}_1$ honest	$\mathcal{P}_0$ honest, $\mathcal{P}_1$ corrupt	$\mathcal{P}_0$ and $\mathcal{P}_1$ corrupt
close ($\leq \delta$)			fresh	fresh	r, r	a, b	a, b
far ($> \delta$)			fresh	fresh	r, s	r, b	a, b
	close ($\leq \delta$)	close ($\leq \delta$)	compro-mised	compro-mised	a, b	a, b	a, b
	close ($\leq \delta$)	far ($> \delta$)	compro-mised	interrupted	a, s	a, b	a, b
	close ($\leq \delta$)		compro-mised	fresh	a, s	a, b	a, b
	far ($> \delta$)	close ($\leq \delta$)	interrupted	compro-mised	r, b	r, b	a, b
	far ($> \delta$)	far ($> \delta$)	interrupted	interrupted	r, s	r, b	a, b
	far ($> \delta$)		interrupted	fresh	r, s	r, b	a, b
		close ($\leq \delta$)	fresh	compro-mised	r, b	r, b	a, b
		far ($> \delta$)	fresh	interrupted	r, s	r, b	a, b

Fig. 23. Output tables for $\mathcal{F}^P_{\mathsf{fPAKE}}$ and $s\mathcal{F}^P_{\mathsf{RFE}}$. r, s represent random outputs, while a, b represent adversarially chosen outputs.

E A Concrete OT

In this section, we recall a concrete UC-secure oblivious transfer protocol due to Chou and Orlandi [CO15]. While they consider the general case of 1-out-of-n transfer, we only consider $n = 2$. Say m 1-out-of-2 OTs are performed. The protocol requires the sender to compute $m + 2$ exponentiations, and the receiver to compute $2m$ exponentiations, for a total of $3m + 2$ exponentiations. Figure 24 shows a summary of the protocol. Note that this construction does require a random oracle.

F Proof of Theorem 4

We proceed in a series of games, starting with the real execution of the protocol and ending up with the ideal execution, with a simulator. For convenience, we refer to a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ from the adversary $\mathcal{S}$ as *due* when:

- $\mathcal{P}_i$ is honest
- there is a `fresh` record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$
- this is the first `NewKey` query for $\mathcal{P}_i$
- there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ in $\Lambda_{\mathcal{P}}$ with $\mathsf{pw}_i = \mathsf{pw}_{1-i}$ and $\mathcal{P}_{1-i}$ is honest
- a key k_{1-i} was sent to the other party, and $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was `fresh` at the time.

Game $\mathbf{G}_0$: The real protocol execution. This is the real execution where the environment $\mathcal{Z}$ runs the EKE2 protocol (see Figure 26) with parties $\mathcal{P}_0$

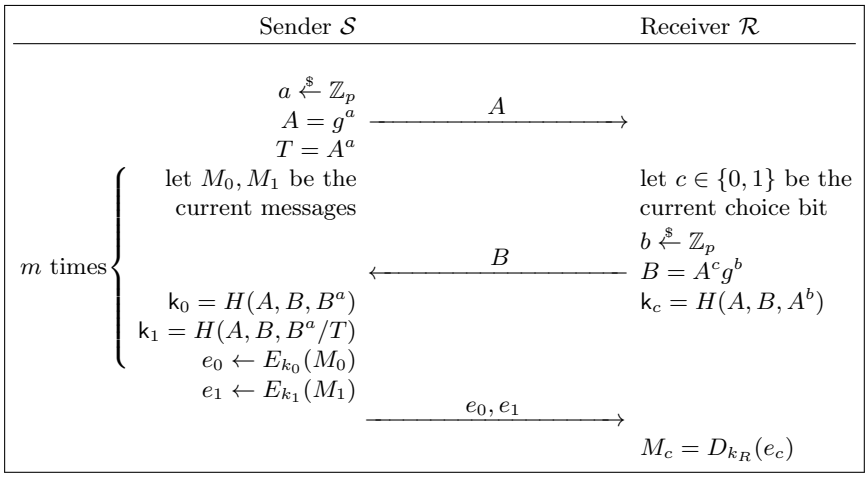

Fig. 24. A Concrete OT [CO15] to be used in Π_{RFE}

and $\mathcal{P}_1$, both having access to ideal CRS, RO, and IC functionalities, and an adversary $\mathcal{A}$ that, w.l.o.g., is assumed to be the dummy adversary as shown in [Can01, section 4.4.1].

Game $\mathbf{G}_1$: Modeling the ideal layout. We first regroup and create new machines, similar to Game 1 in the proof of Theorem 5. The new machine $\mathcal{S}$ executes the code of the CRS, RO and IC functionalities as depicted in Figures 12, 13, and 14.

Game $\mathbf{G}_2$: Simulating the ideal functionalities. We modify simulation of $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{IC}}$ as follows. We let $\mathcal{S}$ implement Figure 14 by maintaining a list Λ_{IC} with entries of the form $(k, m, \alpha, \mathcal{E}|\mathcal{D}, c)$. $\mathcal{S}$ handles encryption and decryption queries as follows:

- Upon receiving $(\mathsf{sid}, \mathcal{E}, \mathsf{k}, m)$ (for shortness of notation, we will also write $\mathcal{E}_{\mathsf{k}}(m)$ for this query), if $\mathsf{k} \notin \mathbb{F}_p$ or $m \notin \mathbb{G}$ then abort. Else, if there is an entry $(\mathsf{k}, m, *, *, c)$ in Λ_{IC}, $\mathcal{S}$ replies with (sid, c). Else, $\mathcal{S}$ chooses $c \xleftarrow{\$} \mathbb{G} \setminus \{1\}$. If there already is a record $(*, *, *, *, *, c)$ in Λ_{IC}, $\mathcal{S}$ aborts. Else, $\mathcal{S}$ adds $(\mathsf{k}, m, \perp, \mathcal{E}, c)$ to Λ_{IC} and replies with (sid, c).
- Upon receiving $(\mathsf{sid}, \mathcal{D}, \mathsf{k}, c)$ (or $\mathcal{D}_{\mathsf{k}}(c)$, for short), if $\mathsf{k} \notin \mathbb{F}_p$ or $c \notin \mathbb{G}$ then abort. Else, if there is an entry $(\mathsf{k}, m, *, *, c)$ in Λ_{IC}, $\mathcal{S}$ replies with (sid, m). Else, $\mathcal{S}$ chooses $\alpha \leftarrow \mathbb{F}_q^*$. If there already is a record $(*, *, g^\alpha, *, *, *)$ in Λ_{IC}, $\mathcal{S}$ aborts. Else, $\mathcal{S}$ adds $(\mathsf{k}, g^\alpha, \alpha, \mathcal{D}, c)$ to Λ_{IC} and replies with (sid, g^α).

Similarly, let Λ_{RO} denote the list that $\mathcal{S}$ maintains upon implementing Figure 13, containing entries of the form (m, h). We let $\mathcal{S}$ handle queries to $\mathcal{F}_{\mathsf{RO}}$ as follows:

- Upon receiving $H(m)$, if $m \notin \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}^3$, then abort. Else, if there is an entry (m, h) in Λ_{RO}, $\mathcal{S}$ replies with (sid, h). Else, $\mathcal{S}$ chooses $h \xleftarrow{\$} \{0,1\}^k$. If there already is a record $(*, *, h)$ in Λ_{RO}, $\mathcal{S}$ aborts. Else, $\mathcal{S}$ adds (m, h) to Λ_{RO} and replies with (sid, h).

The functionality $\mathcal{F}_{\ell\text{-iPAKE}}$ is parameterized by a security parameter λ and makes use of two initially empty lists $\Lambda_{\mathcal{P}}$ and $\Lambda_{\mathcal{L}}$, storing pass-strings and labels, respectively. It interacts with an adversary $\mathcal{S}$ and the (dummy) parties $\mathcal{P}_0$ and $\mathcal{P}_1$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i, \ell)$ from party $\mathcal{P}_i$:**
  - Send $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \ell)$ to $\mathcal{S}$;
  - If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ in $\Lambda_{\mathcal{P}}$:
    - Record $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$ and mark this record `fresh`.
    - Unless there exists a record $(\mathcal{P}_i, \cdot)$, record $(\mathcal{P}_i, \ell)$ in $\Lambda_{\mathcal{L}}$.

- **Upon receiving a query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}'_i, \ell')$ from $\mathcal{S}$:**
  If there is a `fresh` record $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$, then do:
  - If $\mathsf{pw}_i = \mathsf{pw}'_i$, mark the record `compromised`; else mark it `interrupted`;
  - Record $(\mathcal{P}_{1-i}, \ell')$ in $\Lambda_{\mathcal{L}}$, possibly overwriting any existing record $(\mathcal{P}_{1-i}, \cdot)$.

- **Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{sk})$ from $\mathcal{S}$, where $|\mathsf{sk}| = \lambda$:**
  If there is a record $(\mathcal{P}_{1-i}, \ell)$ in $\Lambda_{\mathcal{L}}$, extract ℓ from it; otherwise set $\ell \leftarrow \perp$.
  If there is a record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$, and this is the first `NewKey` query for $\mathcal{P}_i$, then:
  - If at least one of the following is true, then output $(\mathsf{sid}, \ell, \mathsf{sk})$ to player $\mathcal{P}_i$:
    - The record is `compromised`
    - $\mathcal{P}_i$ is corrupted
    - The record is `fresh`, $\mathcal{P}_{1-i}$ is corrupted, and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $\mathsf{pw}_{1-i} = \mathsf{pw}_i$
  - If this record is `fresh`, both parties are honest, there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $\mathsf{pw}_{1-i} = \mathsf{pw}_i$, a key sk' was sent to $\mathcal{P}_{1-i}$, and $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was `fresh` at the time, then output $(\mathsf{sid}, \ell, \mathsf{sk}')$ to $\mathcal{P}_i$;
  - In any other case, pick a new random key sk' of length λ and send $(\mathsf{sid}, \ell, \mathsf{sk}')$ to $\mathcal{P}_i$.

  No matter what, mark the record $(\mathcal{P}_i, \mathsf{pw}_i)$ as `completed`.

Fig. 25. Functionality $\mathcal{F}_{\ell\text{-iPAKE}}$

The parties $\mathcal{P}_0$ and $\mathcal{P}_1$ are running with $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}_{\mathsf{IC}}$.
Protocol Steps:

1. When a party $\mathcal{P}_i$, $i \in \{0, 1\}$, receives an input $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i, \ell)$ from $\mathcal{Z}$, it does the following:
   - chooses $x \xleftarrow{\$} \mathbb{F}_q$
   - sends $(\mathsf{sid}, \mathtt{crs})$ to $\mathcal{F}_{\mathsf{CRS}}$ and receives $(\mathsf{sid}, (g, q))$ back
   - sends $(\mathsf{sid}, \mathcal{E}, \mathsf{pw}_i || \ell, g^x)$ to $\mathcal{F}_{\mathsf{IC}}$ and receives (sid, X^*) back
   - sends $(\mathsf{sid}, \ell, X^*)$ to $\mathcal{P}_{1-i}$ and waits for an answer
2. When $\mathcal{P}_i$, who already obtained an input $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i, \ell)$ and thus holds $(x, (g, q), X^*)$, receives a message $(\mathsf{sid}, \ell', Y^*)$ from $\mathcal{P}_{1-i}$, it
   - sends $(\mathsf{sid}, \mathcal{D}, \mathsf{pw}_i || \ell, Y^*)$ to $\mathcal{F}_{\mathsf{IC}}$ and receives (sid, X') back
   - sends $(\mathsf{sid}, X^*, Y^*, X'^y)$ to $\mathcal{F}_{\mathsf{RO}}$ and receives $(\mathsf{sid}, \mathsf{k}_i)$ back
   - outputs $(\mathsf{sid}, \ell', \mathsf{k}_i)$ towards $\mathcal{Z}$ and terminates the session.

Fig. 26. A UC Execution of EKE2

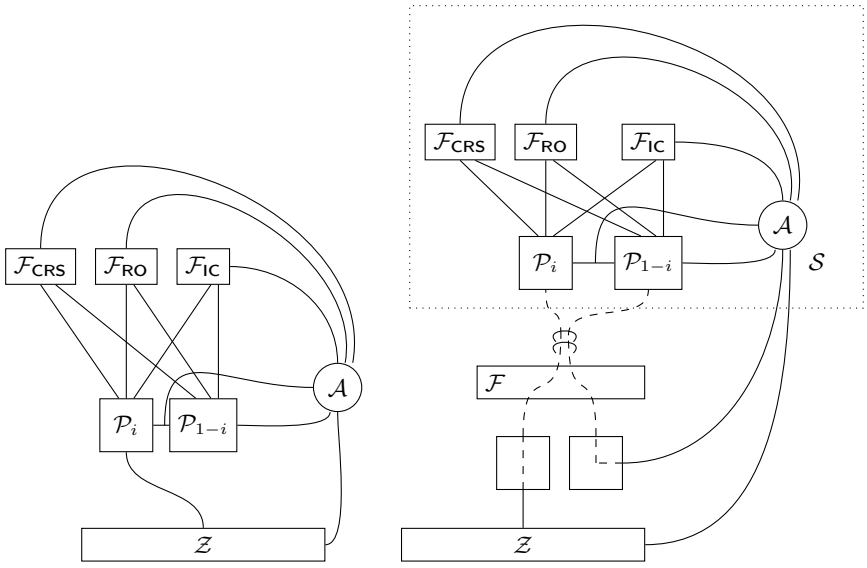

Fig. 27. Transition from game $\mathbf{G}_0$ (left) to game $\mathbf{G}_1$ (right), showing a setting where $\mathcal{P}_{1-i}$ is corrupted.

These modifications later allow $\mathcal{S}$ to extract unique inputs from values obtained from the two functionalities. Especially, note that Λ_{IC} will never contain $(*, \mathsf{k}_i, *, *, \mathcal{E}, c), (*, \mathsf{k}_{1-i}, *, *, \mathcal{E}, c)$ with $\mathsf{k}_i \neq \mathsf{k}_{1-i}$. The entry α serves $\mathcal{S}$ as a trapdoor for solving discrete-log type problems.

Since q is greater than 2^λ, if the oracles are only queried a polynomial number of times, the birthday problem states that game $\mathbf{G}_1$ and game $\mathbf{G}_2$ are indistinguishable with probability overwhelming in λ.

Game $\mathbf{G}_3$: Building $\mathcal{F}_{\mathsf{iPAKE}}$. In this game, we start modeling $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$. First, we let $\mathcal{F}$ maintain two initially empty lists: $\Lambda_{\mathcal{P}}$, a list of tuples of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ and $\Lambda_{\mathcal{L}}$, a list of tuples of the form $(\mathcal{P}_i, \ell)$. Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i, \ell)$ from (dummy) party $\mathcal{P}_i$, if this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i}, \ell')$, then $\mathcal{F}$ records $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$ and marks this record as `fresh`. If $\Lambda_{\mathcal{L}}$ does not contain any record $(\mathcal{P}_i, \cdot)$ so far, $\mathcal{F}$ also records $(\mathcal{P}_i, \ell)$ in $\Lambda_{\mathcal{L}}$. Then, $\mathcal{F}$ relays the query $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i, \ell)$ to $\mathcal{S}$. Now that $\mathcal{F}$ knows about pass-strings and labels, we can add a `TestPwd` interface to $\mathcal{F}$ as described in Figure 7. We let $\mathcal{S}$ parse outputs $(\mathsf{sid}, \ell', \mathsf{k}_i)$ towards $\mathcal{F}$ to be of the form $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ by adding the `NewKey` tag and the name of the party who produced the output. Additionally, we let $\mathcal{F}$ translate this back to $(\mathsf{sid}, \ell', \mathsf{k}_i)$ and send it to $\mathcal{Z}$ via the dummy party $\mathcal{P}_i$, marking the corresponding record as `completed`.

None of these modifications changes the output towards $\mathcal{Z}$ compared to the previous game $\mathbf{G}_2$.

Game $\mathbf{G}_4$: $\mathcal{F}$ generates a random session key for an honest, interrupted session. Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from $\mathcal{S}$, if $\mathcal{P}_i$ is not corrupted and there is a record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ that is marked as `interrupted`, and this is the first `NewKey` query for $\mathcal{P}_i$, we let $\mathcal{F}$ choose a random session key k^* of length λ. Additionally, $\mathcal{F}$ derives the label as follows: if there is a record $(\mathcal{P}_{1-i}, \ell^*)$ in $\Lambda_{\mathcal{L}}$, extract ℓ^* from it; otherwise, set $\ell^* \leftarrow \perp$. Then, $\mathcal{F}$ outputs $(\mathsf{sid}, \ell^*, \mathsf{k}^*)$ to P.

If there is no such interrupted record, $\mathcal{F}$ continues to relay k_i and ℓ'.

Since the simulators described in game $\mathbf{G}_3$ and game $\mathbf{G}_4$ do not make use of the `TestPwd` interface, none of the records of $\mathcal{F}$ are marked as `interrupted` and thus the output towards $\mathcal{Z}$ is equally distributed in both games.

Game $\mathbf{G}_5$: $\mathcal{S}$ handles dictionary attacks against the client $\mathcal{P}_0$ using the `TestPwd` interface. In this game, we will only change the simulation. First note that the client, the initiator of the protocol, is intended to send the first message, and we call him $\mathcal{P}_0$. If both $\mathcal{P}_0$ and $\mathcal{P}_1$ are honest, $\mathcal{P}_0$ obtained input and $\mathcal{Z}$ advises $\mathcal{A}$ to substitute $(\mathsf{sid}, \ell', Y^*)$ with $(\mathsf{sid}, \ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}})$, or if $\mathcal{P}_1$ is corrupted and produces $(\mathsf{sid}, \ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}})$ as first flow, then $\mathcal{S}$ will proceed simulation of $\mathcal{P}_0$ using $\ell_{\mathcal{Z}}$ and $Y^*_{\mathcal{Z}}$. In this situation, we modify $\mathcal{S}$ as follows: upon receiving $(\mathsf{sid}, \ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}})$, if there is an entry $(\mathsf{pw}_{\mathcal{Z}} || \hat{\ell}, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$ for any $\hat{\ell} \in \mathcal{L}$ in Λ_{IC}[11], $\mathcal{S}$ asks a `TestPwd` query $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_0, \mathsf{pw}_{\mathcal{Z}}, \ell_{\mathcal{Z}})$ to $\mathcal{F}$. $\mathcal{S}$ then proceeds the simulation using $\mathsf{pw}_{\mathcal{Z}}$ and $\ell_{\mathcal{Z}}$ instead of pw_0 and ℓ'[12]. If there is no entry $(\mathsf{pw}_{\mathcal{Z}} || \hat{\ell}, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$ in Λ_{IC}, $\mathcal{S}$ sends $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_0, \mathsf{pw}_0, \ell_{\mathcal{Z}})$ to $\mathcal{F}$[13].

Regarding the label, observe that $\mathcal{S}$'s `NewKey` query will contain $\ell_{\mathcal{Z}}$ which was contained in the output of the honest $\mathcal{P}_0$ (cf. Figure 26). Since `TestPwd` queries overwrite any existing labels, there will be an entry $(\mathcal{P}_1, \ell_{\mathcal{Z}})$ in $\Lambda_{\mathcal{L}}$ and thus, regarding the label, the output towards $\mathcal{Z}$ does not change compared to the previous game. Regarding the session key, we have to analyze different cases depending on whether $Y^*_{\mathcal{Z}}$ was generated using $\mathcal{F}_{\mathsf{IC}}$ or not. However, observe that the only changes of session keys between this and the previous game occur whenever a `TestPwd` query of $\mathcal{S}$ causes a record to be marked as `interrupted`.

- There is an entry $(\mathsf{pw}_{\mathcal{Z}} || \hat{\ell}, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$ in Λ_{IC}: if $\mathsf{pw}_{\mathcal{Z}} = \mathsf{pw}_0$, the record is marked `compromised` and the session key is not changed by $\mathcal{F}$. If $\mathsf{pw}_{\mathcal{Z}} \neq \mathsf{pw}_0$, on the other hand, the record is marked `interrupted` and $\mathcal{F}$ hands out a random session key, as opposed to game $\mathbf{G}_4$. However, since the session key is distributed as before, $\mathcal{Z}$ can only detect this by

[11] This entry is unique due to simulation of $\mathcal{F}_{\mathsf{IC}}$ as described in game $\mathbf{G}_2$.

[12] Note that, since $\mathcal{F}$ does not leak any information at this point, $\mathcal{S}$ cannot depend on the outcome of a `TestPwd` query.

[13] Letting $\mathcal{S}$ guess a pass-string that he actually knows seems a little artificial. Indeed, the simulation in this case will be changed in game $\mathbf{G}_{12}$ when $\mathcal{S}$ becomes oblivious of $\mathcal{P}_0$'s pass-string.

reproducing $\mathcal{P}_0$'s input $(\mathsf{sid}, X^*, Y^*_{\mathcal{Z}}, \mathsf{CDH}(\mathcal{D}_{\mathsf{pw}_0||\ell}(X^*), \mathcal{D}_{\mathsf{pw}_0||\ell_{\mathcal{Z}}}(Y^*_{\mathcal{Z}}))$ to $\mathcal{F}_{\mathsf{RO}}$. Lemma 10 (see below) shows indistinguishability of game $\mathbf{G}_4$ and game $\mathbf{G}_5$.

– There is no entry $(*, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$ in Λ_{IC}: since the `TestPwd` query will result in a compromised record, the modified simulation has no impact on the output towards $\mathcal{Z}$ in this case.

The following lemma bounds the probability that an unsuccessful dictionary attack leads to a non-random looking session key. Since in this case the labels do not play any role (the encryption keys of the form $pass-string||label$ will not match regardless of the labels), we ignore them for the sake of simplicity.

Lemma 10. *If CDH holds in $\mathbb{G}$, then $\forall \mathsf{pw}_0, Y^*_{\mathcal{Z}} \leftarrow \mathcal{Z}$, where $Y^*_{\mathcal{Z}}$ is a ciphertext generated through $\mathcal{F}_{IC}$ with some key $\mathsf{pw}_{\mathcal{Z}} \neq \mathsf{pw}_0$, $\mathsf{pw}_0 \in \mathbb{F}_p$, it holds that*

$$\Pr_{\boldsymbol{G}_5}[\mathit{CDH}(\mathcal{D}_{\mathsf{pw}_0}(X^*), \mathcal{D}_{\mathsf{pw}_0}(Y^*_{\mathcal{Z}})) \leftarrow \mathcal{Z}(X^*)] = negl(\lambda).$$

Proof. We create an attacker $\mathcal{B}_{\mathsf{CDH}}$ given a CDH instance $(g, A = g^a, B = g^b)$. $\mathcal{B}_{\mathsf{CDH}}$ runs $\mathcal{Z}$ simulating game $\mathbf{G}_5$ as follows: $\mathcal{B}_{\mathsf{CDH}}$ internally runs all of the participating machines, i.e. $\mathcal{S}$, $\mathcal{F}$ and the dummy parties as in game $\mathbf{G}_5$, but with some modifications. First, $\mathcal{B}_{\mathsf{CDH}}$ computes $X^* \leftarrow \mathcal{E}_{\mathsf{pw}_0}(A)$ and updates Λ_{IC} accordingly, aborting if there already was an entry $(\mathsf{pw}_0, A, *, \mathcal{E}, *)$. Upon receiving a query $\mathcal{D}_{\mathsf{pw}_0}(Y^*_{\mathcal{Z}})$, $\mathcal{B}_{\mathsf{CDH}}$ again aborts if there already is an entry $(\mathsf{pw}_0, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$. Otherwise, it draws $\beta \xleftarrow{\$} \mathbb{F}_q$ and sets the answer to this query to be Bg^β. This can happen multiple times (for different $Y^*_{\mathcal{Z}}$), and $\mathcal{B}_{\mathsf{CDH}}$ keeps track of the pairs $(\beta, Y^*_{\mathcal{Z}})$ in a list Λ_{CDH}. The last modification concerns the part of the simulator's code of game $\mathbf{G}_5$ where a value $Z \leftarrow \mathcal{D}_{\mathsf{pw}_0}(Y^*)^a$ needs to be computed, but note that $\mathcal{B}_{\mathsf{CDH}}$ does not know a. Instead, $\mathcal{B}_{\mathsf{CDH}}$ just sets $Z \leftarrow \perp$.

Finally, $\mathcal{B}_{\mathsf{CDH}}$ picks a random entry from Λ_{RO} asked by $\mathcal{Z}$, parses it as $((\mathsf{pw}_0, X^*, Y^*_{\mathcal{Z}}, Z), h)$, looks for an entry $(\beta, Y^*_{\mathcal{Z}})$ in Λ_{CDH} and outputs $Z/(g^a)^\beta$ as a CDH solution.

First, note that if $\mathcal{B}_{\mathsf{CDH}}$ does not abort, it perfectly emulates $\mathcal{Z}$'s view in game $\mathbf{G}_5$, since A, Bg^β are random in $\mathbb{G}$ and the record for $\mathcal{P}_0$ will be interrupted, which means that $\mathcal{F}$ will output a random session key for $\mathcal{P}_0$, overwriting $\perp$. Second, $\mathcal{B}_{\mathsf{CDH}}$ only has to abort if there is a collision upon choosing random values from $\mathbb{G}$.

Assume that $\mathcal{Z}$ outputs $\mathsf{CDH}(\mathcal{D}_{\mathsf{pw}_0}(X^*), \mathcal{D}_{\mathsf{pw}_0}(Y^*_{\mathcal{Z}}))$ with non-negligible probability. This is only possible if $\mathcal{Z}$ asked both corresponding decryption queries. Existence of $(\mathsf{pw}_{\mathcal{Z}}, *, *, \mathcal{E}, Y^*_{\mathcal{Z}})$ with $\mathsf{pw}_{\mathcal{Z}} \neq \mathsf{pw}_0$ in Λ_{IC} ensures that the answer to $\mathcal{D}_{\mathsf{pw}_0}(Y^*_{\mathcal{Z}})$ can be chosen by $\mathcal{B}_{\mathsf{CDH}}$ as described above. Thus, $\mathcal{B}_{\mathsf{CDH}}$ finds a correct CDH solution with non-negligible probability $1/q_{\mathcal{Z}}$, where $q_{\mathcal{Z}}$ is the number of hash queries issued by $\mathcal{Z}$.

Game $\mathbf{G}_6$: $\mathcal{S}$ handles dictionary attacks against the server $\mathcal{P}_1$ using the `TestPwd` interface. Again, in this game we only change the simulation.

Analogously to game $\mathbf{G}_5$, we let $\mathcal{S}$ use the `TestPwd` interface upon receiving adversarially generated $X^*_{\mathcal{Z}}, \ell_{\mathcal{Z}}$ upon simulating $\mathcal{P}_1$. Observe that the only difference is due to the order of flows: if $\mathcal{S}$ extracts an incorrect pass-string, he produces Y^* using this wrong pass-string. However, Y^* will be distributed as before and again, $\mathcal{Z}$ can only detect the change by reproducing $\mathcal{P}_1$'s input to $\mathcal{F}_{\mathsf{RO}}$, namely $(\mathsf{sid}, X^*_{\mathcal{Z}}, Y^*, \mathsf{CDH}(\mathcal{D}_{\mathsf{pw}_1||\ell_{\mathcal{Z}}}(X^*_{\mathcal{Z}}), \mathcal{D}_{\mathsf{pw}_1||\ell'}(Y^*))$.
Using an analogous argument to Lemma 10, indistinguishability from game $\mathbf{G}_5$ follows from the hardness of CDH in $\mathbb{G}$.

Game $\mathbf{G}_7$: $\mathcal{F}$ aligns session keys. Upon receiving a query $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ from $\mathcal{S}$ for a session, if this query is *due* then output $(\mathsf{sid}, \ell^*, \mathsf{k}^*)$ to $\mathcal{P}_i$ where k^* is the session key that was formerly sent to the other party and the label ℓ^* is derived as usual: if there is a record $(\mathcal{P}_{1-i}, \ell^*)$ in $\Lambda_{\mathcal{L}}$, extract ℓ^* from it; otherwise, set $\ell^* \leftarrow \perp$.
We now analyze distinguishability of this game from game $\mathbf{G}_6$. If $\mathcal{Z}$ tampered with the transcript, any player that received a modified message will not have a fresh record anymore (cf. simulation described in games $\mathbf{G}_5$ and $\mathbf{G}_6$) and the output of this player towards $\mathcal{Z}$ is not changed in this game. On the other hand, if $\mathcal{Z}$ does not advise $\mathcal{A}$ to tamper with any message, $\mathcal{F}$ did not overwrite any labels and thus $\ell^* = \ell'$. Additionally, perfect correctness of the EKE2 protocol ensures that, in case of a due record, $\mathsf{k}_i = \mathsf{k}^*$.
Note that $\mathcal{F}$ still differs from the functionality $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$ described in Figure 7 in some aspects. First, it does not output randomly generated session keys towards $\mathcal{Z}$ for honest sessions. Furthermore, it reports all pass-strings to $\mathcal{S}$. We will take care of these remaining differences in the next games.

Game $\mathbf{G}_8$: In some cases, $\mathcal{F}$ generates a random session key when the other party is corrupted. Upon receiving a `NewKey` query $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ from $\mathcal{S}$, if there is a fresh record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$, and this is the first `NewKey` query for $\mathcal{P}_i$, $\mathcal{P}_i$ is honest and $\mathcal{P}_{1-i}$ corrupted and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ in $\Lambda_{\mathcal{P}}$ with $\mathsf{pw}_i \neq \mathsf{pw}_{1-i}$, we let $\mathcal{F}$ pick a new random key k^* of length λ and send $(\mathsf{sid}, \ell^*, \mathsf{k}^*)$ to $\mathcal{P}_i$, where ℓ^*, as usual, is taken from the list $\Lambda_{\mathcal{L}}$ or set to be $\perp$.
The simulation ensures that the record $(\mathcal{P}_i, \mathsf{pw}_i)$ is either compromised or interrupted (cf. description of the simulator in games $\mathbf{G}_5$ and $\mathbf{G}_6$). Thus, the modification has no effect since it only concerns fresh records.

Game $\mathbf{G}_9$: $\mathcal{F}$ generates a random session key for an honest session. Upon receiving a `NewKey` query $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ from $\mathcal{S}$, if there is a fresh record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ in $\Lambda_{\mathcal{P}}$, and this is the first `NewKey` query for $\mathcal{P}_i$, both parties are honest and the `NewKey` query is not *due*, we let $\mathcal{F}$ pick a new random key k^* of length k and send $(\mathsf{sid}, \ell^*, \mathsf{k}^*)$ to $\mathcal{P}_i$, where ℓ^*, as usual, is taken from the list $\Lambda_{\mathcal{L}}$ or set to be $\perp$.
In other words, $\mathcal{F}$ now generates a random session key upon a first `NewKey` query for an honest party $\mathcal{P}_i$ with fresh record $(\mathcal{P}_i, \mathsf{pw}_i)$ where the other party is also honest, if (at least) one of the following events happens:

1. There is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ in $\Lambda_{\mathcal{P}}$ with $\mathsf{pw}_i \neq \mathsf{pw}_{1-i}$;
2. No output was sent to the other party yet;

3. If there was output to the other party, the record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ in $\Lambda_{\mathcal{P}}$ was not fresh and thus interrupted or compromised at that time

In all of these cases, $\mathcal{S}$ chose a fresh k_i following a uniform distribution and ℓ' was contained in the `NewSession` query of $\mathcal{P}_i$'s partner, thus $\ell' = \ell^*$. Regarding the session key, $\mathcal{Z}$ can only notice a difference if it reproduces k_i by computing $\mathcal{P}_i$'s input $(\mathsf{sid}, X^*, Y^*, \mathsf{CDH}(\mathcal{D}_{\mathsf{pw}_i||\ell}(X^*), \mathcal{D}_{\mathsf{pw}_i||\ell'}(Y^*))$ to $\mathcal{F}_{\mathsf{RO}}$ and sending it to $\mathcal{F}_{\mathsf{RO}}$ via the adversary $\mathcal{A}$.
The following lemma bounds the probability that a session key of an unattacked session does not look random.

Lemma 11. *If CDH holds in $\mathbb{G}$, then $\forall \mathsf{pw}, \ell, \ell' \leftarrow \mathcal{Z}$ with $\mathsf{pw} \in \mathbb{F}_p$ and $\ell, \ell' \in \mathcal{L}$ it holds that*

$$\Pr_{\boldsymbol{G}_9}[\mathit{CDH}(\mathcal{D}_{\mathsf{pw}||\ell}(X^*), \mathcal{D}_{\mathsf{pw}||\ell'}(Y^*)) \leftarrow \mathcal{Z}(X^*, Y^*)] = \mathit{negl}(\lambda).$$

Proof. We only sketch the proof since it is similar to the proof of Lemma 10. Namely, the strategy of embedding (randomized versions of) a CDH challenge into the simulation of game $\mathbf{G}_9$ is done by just encrypting both CDH challenge elements to obtain X^* and Y^*. For the final argument, note that $\perp$ is not seen by $\mathcal{Z}$ since it is either replaced using a random session key or a previously computed key.

It follows that game $\mathbf{G}_8$ and game $\mathbf{G}_9$ are indistinguishable.

Game $\mathbf{G}_{10}$: $\mathcal{F}$ always takes all labels from the list $\Lambda_{\mathcal{L}}$. We modify $\mathcal{F}$ as follows: if $\mathcal{F}$ outputs $(\mathsf{sid}, \ell', \mathsf{k}_i)$ towards $\mathcal{P}_i$ where ℓ', k_i are taken from a query $(\mathtt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \ell', \mathsf{k}_i)$ from $\mathcal{S}$, $\mathcal{F}$ extracts ℓ^* from a record $(\mathcal{P}_{1-i}, \ell^*)$ in $\Lambda_{\mathcal{L}}$ or sets $\ell^* \leftarrow \perp$ if such a record does not exist. $\mathcal{F}$ then outputs $(\mathsf{sid}, \ell^*, \mathsf{k}_i)$ towards $\mathcal{P}_i$. We additionally modify $\mathcal{S}$ to remove the labels from the `NewKey` queries altogether.
First observe that we can remove the labels from the `NewKey` queries because, in this and the past games, we ensured that $\mathcal{F}$ now does not access this label anymore. However, we still have to argue indistinguishability of the current and the previous game. The cases where k_i of $\mathcal{S}$ is relayed by $\mathcal{F}$ towards $\mathcal{P}_i$ are the following:

- $\mathcal{P}_i$ has a compromised record
- $\mathcal{P}_i$ is corrupted
- $\mathcal{P}_i$ has a fresh record, its partner is corrupted and has a record with a matching pass-string

In the first case, we have that $\ell' = \ell^*$ since the label ℓ' outputted by $\mathcal{P}_i$ was also contained in a `TestPwd` query by $\mathcal{S}$ and overwrote any existing label send by $\mathcal{P}_i$'s partner. For the second case, observe that since we restrict to static corruption, corrupted players will not have records in $\Lambda_{\mathcal{P}}$ and thus this case will never happen. In the third case, corruption of the partner ensures that $\mathcal{S}$ issued a `TestPwd` query which overwrote any existing label with ℓ', so $\ell' = \ell^*$ as well.
Observe that now $\mathcal{F}$ acts like $\mathcal{F}_{\ell\text{-iPAKE}}$ regarding the output of session keys and labels. The only remaining difference is that the `NewSession` queries

still contain the pass-strings of the parties. In the next games, we will make the simulation independent of these pass-strings.

Game $\mathbf{G}_{11}$: Simulate without pw_1 if server $\mathcal{P}_1$ is honest. In case of receiving a $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_1, \ell')$ from an honest $\mathcal{P}_1$ playing the role of a server, we modify $\mathcal{F}$ by forwarding only $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_1, \ell')$ to $\mathcal{S}$. We now have to modify $\mathcal{S}$ to proceed simulation without knowing pw_1. Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_1, \ell')$ from $\mathcal{F}$ for an honest $\mathcal{P}_1$, we let $\mathcal{S}$ draw uniformly at random a "dummy" pass-string $\mathsf{pw}_{\mathcal{S}}$ and proceed the simulation of $\mathcal{P}_1$ using $\mathsf{pw}_{\mathcal{S}}$ as a pass-string.

We first note that, if at any time $\mathcal{S}$ sends a `NewKey` query to $\mathcal{F}$ containing a session key k_1 for $\mathcal{P}_1$, this session key is only seen by $\mathcal{Z}$ if $\mathcal{P}_1$'s record is compromised. Otherwise, we thus only have to argue indistinguishability of the transcripts of game $\mathbf{G}_{10}$ and game $\mathbf{G}_{11}$.

- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, X^*_{\mathcal{Z}}$, there is a record $(\mathsf{pw}_{\mathcal{Z}}||\hat{\ell}, X, *, \mathcal{E}, X^*)$ in Λ_{IC} for some $\hat{\ell} \in \mathcal{L}$ and $\mathsf{pw}_{\mathcal{Z}} = \mathsf{pw}_1$: since $\mathcal{S}$ will issue a `TestPwd` query that will result in a compromised record (cf. simulation described in game $\mathbf{G}_6$), nothing is changed since $\mathsf{pw}_{\mathcal{S}}$ was never used, and Y^* is generated using the correct pass-string $\mathsf{pw}_{\mathcal{Z}}$.
- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, X^*_{\mathcal{Z}}$, there is a record $(\mathsf{pw}_{\mathcal{Z}}||\hat{\ell}, X, *, \mathcal{E}, X^*)$ in Λ_{IC} for some $\hat{\ell} \in \mathcal{L}$ and $\mathsf{pw}_{\mathcal{Z}} \neq \mathsf{pw}_1$: since $\mathcal{P}_1$ will receive a random session key from $\mathcal{F}$ in this case (the record will be marked `interrupted`), we only have to argue indistinguishability of Y^* generated with $\mathsf{pw}_{\mathcal{Z}}||\ell'$ instead of $\mathsf{pw}_1||\ell'$. Simulation of $\mathcal{F}_{\mathsf{IC}}$ ensures that Y^* is distributed uniformly random as before. Observe that here it is crucial that even for a corrupted session, an interrupted record lets the functionality hand out a random session key, since $\mathcal{S}$ has no means to decide whether it has to output a session key for $\mathcal{P}_1$ that matches the session key that $\mathcal{Z}$ can compute on behalf of $\mathcal{P}_0$ from the message (ℓ', Y^*).
- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, X^*_{\mathcal{Z}}$ and no $\mathcal{E}$ record: the simulation described in game $\mathbf{G}_6$ tells $\mathcal{S}$ to issue a `TestPwd` query, but now using $\mathsf{pw}_{\mathcal{S}}$ instead of pw_1. If, coincidentally, $\mathsf{pw}_1 = \mathsf{pw}_{\mathcal{S}}$, nothing changes. On the other hand, if $\mathsf{pw}_1 \neq \mathsf{pw}_{\mathcal{S}}$, $\mathcal{P}_1$ obtains a random session key from $\mathcal{F}$ as opposed to the game before and Y^* is created using $\mathsf{pw}_{\mathcal{S}}||\ell'$ instead of $\mathsf{pw}_1||\ell'$. This can only be detected if $\mathcal{Z}$ reproduces $\mathcal{P}_1$'s input to $\mathcal{F}_{\mathsf{RO}}$ from game $\mathbf{G}_{10}$, which happens only with negligible probability according to Lemma 12 (see below).
- both parties honest and no injections: $\mathcal{P}_1$ will obtain a uniformly random session key from $\mathcal{F}$ in this case, and thus the only difference is that Y^* was created using $\mathsf{pw}_{\mathcal{S}}||\ell'$ instead of $\mathsf{pw}_1||\ell'$. Again, this is indistinguishable since Y^* is distributed exactly as before.

The following lemma bounds the probability that an injected X^* that was not obtained using encryption leads to a non-random looking session key.

Lemma 12. *If CDH holds in $\mathbb{G}$, then $\forall \mathsf{pw}_1, \ell', \ell_{\mathcal{Z}}, X^*_{\mathcal{Z}} \leftarrow \mathcal{Z}$, where $X^*_{\mathcal{Z}}$ was not generated using $\mathcal{F}_{\mathsf{IC}}$, $\ell', \ell_{\mathcal{Z}} \in \mathcal{L}$ and $\mathsf{pw}_1 \in \mathbb{F}_p$, it holds that*

$$\Pr_{\boldsymbol{G}_{11}}[\mathit{CDH}(\mathcal{D}_{\mathsf{pw}_1||\ell_{\mathcal{Z}}}(X^*_{\mathcal{Z}}), \mathcal{D}_{\mathsf{pw}_1||\ell'}(Y^*)) \leftarrow \mathcal{Z}(Y^*)] = \mathit{negl}(\lambda).$$

Proof. Note that the only difference to Lemma 10 is that this time, no record $(*, *, *, \mathcal{E}, X^*_{\mathcal{Z}})$ exists so the fact that $\mathcal{B}_{\mathsf{CDH}}$ is able to embed an element of its CDH challenge into $\mathcal{D}_{\mathsf{pw}_1}(X^*_{\mathcal{Z}})$ is even more obvious. The rest of the proof is analogously to Lemma 10.

Game $\mathbf{G}_{12}$: Simulate without pw_0 if client $\mathcal{P}_0$ is honest. In a similar fashion, we now let $\mathcal{F}$ cut the pass-string from `NewSession` queries to an honest $\mathcal{P}_0$. We again have to modify $\mathcal{S}$ to proceed simulation without knowing pw_0. Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_0, \ell)$ from $\mathcal{F}$ for an honest $\mathcal{P}_0$, we let $\mathcal{S}$ draw uniformly at random a "dummy" pass-string $\mathsf{pw}_{\mathcal{S}}$. $\mathcal{S}$ proceeds the simulation of $\mathcal{P}_0$ using $\mathsf{pw}_{\mathcal{S}}$ as a pass-string.

Additionally, we further change $\mathcal{S}$ in case of a dictionary attack against client $\mathcal{P}_0$, i.e., upon receiving $\ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}}$ from $\mathcal{Z}$. After submitting a `TestPwd` query with an extracted $\mathsf{pw}_{\mathcal{Z}}$, we let $\mathcal{S}$ now choose $x' \xleftarrow{\$} \mathbb{F}_P$ and add $(\mathsf{pw}_{\mathcal{Z}}||\ell, g^{x'}, x', \perp, X^*)$ to Λ_{IC} and proceed the simulation of $\mathcal{P}_0$ using x' instead of x.

- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}}$, there is a record $(\mathsf{pw}_{\mathcal{Z}}||\hat{\ell}, Y, *, \mathcal{E}, Y^*)$ in Λ_{IC} for some $\hat{\ell} \in \mathcal{L}$ and $\mathsf{pw}_{\mathcal{Z}} = \mathsf{pw}_0$: $\mathcal{S}$ will issue a `TestPwd` query that will result in a compromised record (cf. simulation described in game $\mathbf{G}_5$), resulting in a session key that is computed using $\mathsf{pw}_{\mathcal{Z}}$ instead of $\mathsf{pw}_{\mathcal{S}}$. Additionally, X^* is generated using the incorrect pass-string $\mathsf{pw}_{\mathcal{S}}$. However, adjusting Λ_{IC} as described above still allows $\mathcal{S}$ to know the exponent of $\mathcal{D}_{\mathsf{pw}_{\mathcal{Z}}||\ell}(X^*)$ and continue the simulation, making it look like $\mathsf{pw}_{\mathcal{Z}}$ was used from the beginning. $\mathcal{Z}$'s view is distributed exactly as before since x', x are both uniformly random in $\mathbb{F}_P$.
- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}}$, there is a record $(\mathsf{pw}_{\mathcal{Z}}||\hat{\ell}, Y, *, \mathcal{E}, Y^*)$ in Λ_{IC} for some $\hat{\ell} \in \mathcal{L}$ and $\mathsf{pw}_{\mathcal{Z}} \neq \mathsf{pw}_0$: since $\mathcal{P}_0$ will receive a random session key from $\mathcal{F}$ in this case (the record will be interrupted), we only have to argue indistinguishability of X^* generated with $\mathsf{pw}_{\mathcal{S}}|\ell$ instead of $\mathsf{pw}_0||\ell$. Simulation of $\mathcal{F}_{\mathsf{IC}}$ ensures that Y^* is distributed uniformly random as before. Again, it is crucial here that $\mathcal{F}$ helps $\mathcal{S}$ by randomizing the session key if needed.
- $\mathcal{Z}$ sends $\ell_{\mathcal{Z}}, Y^*_{\mathcal{Z}}$ and no $\mathcal{E}$ record: the simulation described in game $\mathbf{G}_5$ tells $\mathcal{S}$ to issue a `TestPwd` query, but now using $\mathsf{pw}_{\mathcal{S}}||\ell$ instead of $\mathsf{pw}_0||\ell$. If, coincidentally, $\mathsf{pw}_0 = \mathsf{pw}_{\mathcal{S}}$, nothing changes. On the other hand, if $\mathsf{pw}_0 \neq \mathsf{pw}_{\mathcal{S}}$, $\mathcal{P}_0$ obtains a random session key from $\mathcal{F}$ as opposed to the game before and X^* was created using $\mathsf{pw}_{\mathcal{S}}||\ell$ instead of $\mathsf{pw}_0||\ell$. This can only be detected if $\mathcal{Z}$ reproduces $\mathcal{P}_0$'s input to $\mathcal{F}_{\mathsf{RO}}$ from game $\mathbf{G}_{11}$, which happens only with negligible probability using an argument very similar to Lemma 12.
- both parties honest and no injections: $\mathcal{P}_0$ will obtain a uniformly random session key from $\mathcal{F}$ in this case, and thus the only difference is that X^* was created using $\mathsf{pw}_{\mathcal{S}}||\ell$ instead of $\mathsf{pw}_0||\ell$. Again, this is indistinguishable since X^* is distributed exactly as before.

Observe that in game $\mathbf{G}_{12}$, $\mathcal{F} = \mathcal{F}_{\ell\text{-iPAKE}}$[14], and thus the theorem follows. The complete description of the simulator of game $\mathbf{G}_{12}$ interacting with $\mathcal{F}_{\ell\text{-iPAKE}}$ and $\mathcal{Z}$ is given in Figure 28.

G Proof of Theorem 5

For the proof, we describe an honest execution of the protocol $\mathsf{fPAKE}_{\mathsf{RSS}}$ in the UC framework in Figure 29. See [FHH14] for a detailed description on how to execute protocols within the UC framework. This real protocol execution will be the starting point for our proof. We then proceed in a series of games, to end up with the ideal execution running with only dummy parties, a simulator and the ideal functionality $\mathcal{F}^{M}_{\mathsf{fPAKE}}$. For convenience, we refer to a received protocol message as *adversarially generated* if it was not produced by either $\mathcal{P}_0$ or $\mathcal{P}_1$. We also refer to a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from the adversary $\mathcal{S}$ with an honest party $\mathcal{P}_i$ as *due* if

- there is a `fresh` record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$
- this is the first `NewKey` query for $\mathcal{P}_i$
- there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) \leq \delta$ and $\mathcal{P}_{1-i}$ is honest
- a key k_{1-i} was sent to the other party while $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was `fresh` at the time.

We also define a masking function that reveals the positions of the identical bits:

$$m(\mathsf{pw}, \mathsf{pw}') := \{i | \mathsf{pw}_i = \mathsf{pw}'_i, i \in [n]\}$$

Game $\mathbf{G}_0$: The real protocol execution. This is the real execution of $\mathsf{fPAKE}_{\mathsf{RSS}}$ where the environment $\mathcal{Z}$ runs the protocol (cf. Figure 9) with parties $\mathcal{P}_0$ and $\mathcal{P}_1$, both having access to an ideal ℓ-iPAKE functionality $\mathcal{F}_{\ell\text{-iPAKE}}$, and an adversary $\mathcal{A}$ that, w.l.o.g., can be assumed to be the dummy adversary as shown in [Can01, section 4.4.1].

Game $\mathbf{G}_1$: Modeling the ideal layout. We first make some purely conceptual changes that do not modify the input/output interfaces of $\mathcal{Z}$. We add one relay (also referred to as *dummy party*) on each of the wires between $\mathcal{Z}$ and a party. We also add one relay covering all the wires between the dummy parties and real parties and call it $\mathcal{F}$ (and let $\mathcal{F}$ relay messages according to the original wires). We group all the formerly existing instances except for $\mathcal{Z}$ into one machine and call it $\mathcal{S}$. Note that this implies that $\mathcal{S}$ executes the code of the ℓ-iPAKE functionality $\mathcal{F}_{\ell\text{-iPAKE}}$. The differences are depicted in Figure 19 with $\mathcal{F}_{\mathsf{OT}}$ replaced by $\mathcal{F}_{\ell\text{-iPAKE}}$.

[14] We note that we can, w.l.o.g, assume that there are no `NewSession` queries from $\mathcal{Z}$ to corrupted parties. Thus, it is enough to remove the pass-strings from the `NewSession` queries given as input from $\mathcal{Z}$ to honest parties.

The simulator $\mathcal{S}$, initialized with a security parameter λ, first runs a group generation algorithm using $\lambda$to obtain a cyclic group $\mathbb{G}$ with generator g of order q with $\log_2(q) \geq \lambda$. Then, $\mathcal{S}$ initializes the dummy adversary $\mathcal{A}$. $\mathcal{S}$ then interacts with an ideal functionality $\mathcal{F}_{\ell\text{-iPAKE}}$ and an environment $\mathcal{Z}$ via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \ell)$ from $\mathcal{F}_{\ell\text{-iPAKE}}$:**
  - initialize a party $\mathcal{P}_i$, connect it to $\mathcal{A}$ and proceed the UC protocol execution described in Figure 26 using $\mathsf{pw}_{\mathcal{S}} \xleftarrow{\$} \mathbb{F}_p$ as pass-string and $\mathcal{S}$'s random coins. (Cf. games $\mathbf{G}_0$,$\mathbf{G}_{11}$ and $\mathbf{G}_{12}$.)
- **Upon receiving a query (sid, m) from any entity:** (Cf. games $\mathbf{G}_1$ and $\mathbf{G}_2$.)
  If $m \notin \langle g \rangle^3$, then abort. Else:
  - if there is an entry (m, h) then reply with (sid, h).
  - else, choose $h \xleftarrow{\$} \mathbb{F}_q$ and abort if there already is an entry $(*, *, h)$ in Λ_{RO}. Else, reply with (sid, h).
- **If an internally simulated party $\mathcal{P}_i$ produces an output $(\mathsf{sid}, \ell', \mathsf{k}_i)$:** Send $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ to $\mathcal{F}_{\ell\text{-iPAKE}}$. (Cf. game $\mathbf{G}_1$.)
- **If $\mathcal{Z}$ sends $(\mathsf{sid}, \ell_{\mathcal{Z}}, Z^*_{\mathcal{Z}})$ to an honest party $\mathcal{P}_i$:**
  - if $(\mathsf{pw}_{\mathcal{Z}} || \hat{\ell}, *, *, \mathcal{E}, Z^*_{\mathcal{Z}}) \in \Lambda_{IC}$ for any $\hat{\ell} \in \mathcal{L}$, then send $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_{\mathcal{Z}}, \ell_{\mathcal{Z}})$ to $\mathcal{F}_{\ell\text{-iPAKE}}$ and proceed the simulation of $\mathcal{P}_i$ with $\mathsf{pw}_{\mathcal{Z}}$. (Cf. games $\mathbf{G}_5$ and $\mathbf{G}_6$.)
  - if $(*, *, *, \mathcal{E}, Z^*_{\mathcal{Z}}) \notin \Lambda_{IC}$, then send $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_{\mathcal{S}}, \ell_{\mathcal{Z}})$ to $\mathcal{F}_{\ell\text{-iPAKE}}$. (Cf. games $\mathbf{G}_5$ and $\mathbf{G}_6$.)
  - if $\mathcal{P}_i$ was started with input $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \ell)$, choose $x' \xleftarrow{\$} \mathbb{F}_P$, add $(\mathsf{pw}_{\mathcal{Z}} || \ell, g^{x'}, x', \perp, Z^*)$ to Λ_{IC} and proceed as if x' was the value drawn uniformly random from $\mathbb{F}_q$ at the beginning of the simulation of $\mathcal{P}_i$. (Cf. game $\mathbf{G}_{12}$.)
- **Upon receiving a query $(\mathsf{sid}, \texttt{Encrypt}, k, m)$ from any entity:** (Cf. games $\mathbf{G}_1$ and $\mathbf{G}_2$.)
  If $k \notin \mathbb{F}_p$ or $m \notin \mathbb{G}$ then abort. Else:
  - if there is an entry $(k, m, *, *, c)$ in Λ_{IC}, reply with (sid, c)
  - else, choose $c \xleftarrow{\$} \mathbb{G} \setminus \{1\}$. If there already is a record $(*, *, *, *, c)$ then abort. Else, add $(k, m, \perp, \mathcal{E}, c)$ to Λ_{IC} and reply with (sid, c).
- **Upon receiving a query $(\mathsf{sid}, \texttt{Decrypt}, k, c)$ from any entity:** (Cf. games $\mathbf{G}_1$ and $\mathbf{G}_2$.)
  If $k \notin \mathbb{F}_p$ or $c \notin \mathbb{G}$ then abort. Else:
  - if there is an entry $(k, m, *, *, c)$ in Λ_{IC}, reply with (sid, m).
  - else, choose $\alpha \xleftarrow{\$} \mathbb{F}^*_p$. If there already is a record $(*, g^\alpha, *, *, *)$ then abort. Else, add $(k, g^\alpha, \alpha, \mathcal{D}, c)$ to Λ_{IC} and reply with (sid, g^α).
- **Upon receiving a query $(\mathsf{sid}, \texttt{crs})$ from any entity: (Cf. game $\mathbf{G}_1$.)**
  - reply with $(\mathsf{sid}, (g, q))$.

Additionally, $\mathcal{S}$ forwards all other instructions from $\mathcal{Z}$ to $\mathcal{A}$ and reports all output of $\mathcal{A}$ towards $\mathcal{Z}$. Instructions of corrupting a player are only obeyed if they are received before the protocol started, i.e., before $\mathcal{S}$ received any `NewSession` query from $\mathcal{F}_{\ell\text{-iPAKE}}$.

Fig. 28. The Simulator $\mathcal{S}$ for the EKE2 Protocol

The parties $\mathcal{P}_0, \mathcal{P}_1$ are running with $\mathcal{F}_{\ell\text{-iPAKE}}$.
Protocol Steps:

1. When a party $\mathcal{P}_i$, $i \in \{0,1\}$, receives an input $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from $\mathcal{Z}$, it does the following:
   - compute $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SigGen}(1^\lambda)$
   - query n times $\mathcal{F}_{\ell\text{-iPAKE}}$ with $(\texttt{NewSession}, \mathsf{sid}, (\mathsf{pw}_i)_t, \mathsf{vk})$, $t = 1, ..., n$, receiving back $(\mathsf{sid}, \ell_t, (K_t, L_t))$ as answer
   - if not all ℓ_t are equal or $\ell_1 \neq \mathcal{VK}$, then abort
   - choose $U \xleftarrow{\$} \mathbb{F}_q$
   - compute $C \leftarrow \mathsf{Share}(U)$
   - compute $E \leftarrow C + (K_t)_{t=1}^n$
   - compute $\sigma_E \leftarrow \mathsf{Sign}(\mathsf{vk}, E)$
   - send $(\mathsf{sid}, E, \sigma_E, \mathsf{vk})$ to $\mathcal{P}_{1-i}$ and wait for an answer
2. When $\mathcal{P}_i$, who already obtained an input $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ and thus holds a vector $(\mathsf{sid}, \ell_t, (K_t, L_t))$ obtained from $\mathcal{F}_{\ell\text{-iPAKE}}$, a scalar U and a key pair $(\mathsf{vk}, \mathsf{sk})$, receives a message $(\mathsf{sid}, F, \sigma_F, \mathsf{vk}')$ from $\mathcal{P}_{1-i}$, it does the following:
   - abort if $\mathsf{vk}' \neq \ell_1$
   - abort if $\mathsf{Vfy}(\mathsf{vk}', \sigma_F, F) = 0$
   - compute $V \leftarrow \mathsf{Reconstruct}(F - L)$
   - compute $\mathsf{k} \leftarrow U + V$
   - send $(\mathsf{sid}, \mathsf{k})$ towards $\mathcal{Z}$ and terminate the session.

Fig. 29. A UC Execution of $\mathsf{fPAKE}_{\mathsf{RSS}}$

Game $\mathbf{G}_2$: Building $\mathcal{F}^M_{\mathsf{fPAKE}}$. In this game, we start modeling $\mathcal{F}^M_{\mathsf{fPAKE}}$. First, we let $\mathcal{F}$ maintain a list of tuples of the form $(\mathcal{P}_i, \mathsf{pw}_i)$. Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from party $\mathcal{P}_i$, if this is the first `NewSession`-query, or if this is the second `NewSession`-query and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$, then $\mathcal{F}$ records $(\mathcal{P}_i, \mathsf{pw}_i)$ and marks this record as `fresh`. In any case the query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i)$ is relayed to $\mathcal{S}$. Now that $\mathcal{F}$ knows about pass-strings, we can add a `TestPwd` interface to $\mathcal{F}$ as described in Figure 2, using leakage functions L^M_c, L^M_m and L^M_f. We let $\mathcal{S}$ parse outputs towards $\mathcal{F}$ to be of the form $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ by adding the `NewKey` tag and the name of the party who produced the output. Additionally, we let $\mathcal{F}$ translate this back to $(\mathsf{sid}, \mathsf{k}_i)$, send it to $\mathcal{Z}$ via $\mathcal{P}_i$ and mark the corresponding record as `completed`.

None of these modifications changes the output towards $\mathcal{Z}$ compared to the previous game $\mathbf{G}_1$.

Game $\mathbf{G}_3$: $\mathcal{F}$ generates a random session key for an interrupted session. Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from $\mathcal{S}$, if there is a record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$ that is marked as `interrupted`, and this is the first `NewKey` query for $\mathcal{P}_i$, we let $\mathcal{F}$ output a random session key of length λ to $\mathcal{P}_i$. Otherwise, it continues to relay k_i.

Since the simulators described in game $\mathbf{G}_2$ and game $\mathbf{G}_3$ do not make use of the `TestPwd` interface, none of the records of $\mathcal{F}$ are marked as `interrupted` and thus the output towards $\mathcal{Z}$ is equally distributed in both games.

Game $\mathbf{G}_4$: $\mathcal{S}$ handles dictionary attacks using the `TestPwd` interface. In this game, we only change the simulation. Consider the following setting: $\mathcal{P}_i$ obtained input $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ and $\mathcal{P}_{1-i}$ is corrupted and already provided its inputs to $\mathcal{F}_{\ell\text{-iPAKE}}$. In this situation, $\mathcal{S}$ will proceed simulation of $\mathcal{P}_i$ as follows:

$\mathcal{S}$ assembles $\mathsf{pw}_{\mathcal{Z}} \in \mathbb{F}_p^n$ from the queries to $\mathcal{F}_{\ell\text{-iPAKE}}$ that $\mathcal{P}_{1-i}$ issued. $\mathcal{S}$ sends $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_{\mathcal{Z}})$ to $\mathcal{F}$, obtaining either "wrong guess", "correct guess" and perhaps also a mask $M \subseteq [n]$ from $\mathcal{F}$. If $\mathcal{S}$ does not receive a mask, $\mathcal{S}$ is not modified further. Else, let $I := [n] \setminus M$ the set of mismatching indices, and $d := |I| \leq \gamma$ their number. $\mathcal{S}$ sets up n pairs of keys (K, L) with $(K_t, L_t) = (K'_t, L'_t)_t \xleftarrow{\$} \mathbb{F}_q^2$ for the matching indices $t \in M$ and independent $(K_t, L_t) \xleftarrow{\$} \mathbb{F}_p^2$ and $(K'_t, L'_t) \xleftarrow{\$} \mathbb{F}_p^2$ for the mismatching indices $t \in I$, where (K', L') denotes the output of $\mathcal{F}_{\ell\text{-iPAKE}}$ towards $\mathcal{P}_{1-i}$. $\mathcal{S}$ now continues the simulation of $\mathcal{P}_i$ using (K, L) as output of $\mathcal{F}_{\ell\text{-iPAKE}}$.

We have to analyze different cases depending on the different outcomes of `TestPwd`. However, note that the modifications only have an impact on the output k_i of $\mathcal{P}_i$ if the record gets `interrupted`, and only affect the transcript if the answer to the `TestPwd` query contains a mask. Considering the case where `TestPwd`

- outputs $\boldsymbol{m}$ and sets the record `compromised`, i.e. $d \leq \gamma$ since the distribution of K, K', L, L' only depends on the mask of the pass-strings, the view of $\mathcal{Z}$ is identically distributed in game $\mathbf{G}_4$ and game $\mathbf{G}_3$;
- outputs "wrong guess" and sets the record `interrupted`, i.e. $d > \gamma$: $\mathcal{P}_i$ will now obtain a randomly chosen session key from $\mathcal{F}$, substituting the key k_i computed by $\mathcal{S}$. However, in this case, observe that the privacy property implies that nothing is learned about the secret V'. Hence, k_i looks random. We formally show this in Lemma 13, namely, that the probability that $\mathcal{Z}$ outputs an F that lets $\mathcal{P}_i$ output a non-randomly chosen session key is negligible;

Lemma 13. *Consider an honest party $\mathcal{P}_i$, holding an adversarially determined pass-string pw_i, running the protocol with the adversary holding a pass-string pw_{1-i} with $d := d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) > \gamma$. Then the probability that $\mathcal{P}_i$ outputs a non-randomly chosen session key is negligible in λ.*

Proof. d $\mathcal{F}_{\ell\text{-iPAKE}}$ executions have used different pass-strings, hence with high probability $(1-(\gamma+1)/q-o(1/q))$, at least $\gamma+1$ keys are such that $K_t \neq K'_t$. The privacy property of the (n, t, r)-WRSS says that if at least $n - t$ errors occur the shares look random. Since $\gamma + 1 = n - t$ nothing can be learned about V' from F. Since $\mathsf{k}_i \leftarrow U' + V'$ is a one-time pad, with V' uniformly random, k_i is indistinguishable from random.

Game $\mathbf{G}_5$: Excluding man-in-the-middle attacks. Again, in this game, we only change the simulation. We now consider the case where $\mathcal{Z}$ injects a message into a session where both parties are honest. We modify $\mathcal{S}$ as follows: upon receiving an adversarially generated $(\mathsf{sid}, M_{\mathcal{Z}}, \sigma_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}})$ from $\mathcal{Z}$ intended for party $\mathcal{P}_i$, $\mathcal{S}$ aborts.

Observe that the simulation is only changed compared to the previous game if it is not aborted due to protocol instructions. This means that both games are equal unless all checks pass, especially $\mathsf{Vfy}(\mathsf{vk}_{\mathcal{Z}}, \sigma_{\mathcal{Z}}, M_{\mathcal{Z}}) = 1$. Any distinguisher between game $\mathbf{G}_5$ and game $\mathbf{G}_4$ can thus be turned into a forger of a valid message w.r.t the verification key of an honest party. Indinstinguishability thus follows from the security of the one-time signature scheme.

Game $\mathbf{G}_6$: $\mathcal{F}$ aligns session keys. Upon receiving a query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from $\mathcal{S}$, if this query is *due* then output $(\mathsf{sid}, \mathsf{k}_{1-i})$ to $\mathcal{P}_i$ where k_{1-i} is the session key that was formerly sent to the other party.

We now analyze distinguishability of this game from game $\mathbf{G}_5$. If $\mathcal{Z}$ tampered with the transcript, the simulation in game $\mathbf{G}_5$ ensures that the simulation aborts and there is thus no `NewKey` query for $\mathcal{P}_i$. On the other hand, if $\mathcal{Z}$ does not advise $\mathcal{A}$ to tamper with any message, perfect correctness of $\mathsf{fPAKE}_{\mathsf{RSS}}$ protocol ensures that, in case of a due record where the parties hold close pass-strings $\mathsf{pw}_i, \mathsf{pw}_{1-i}$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) \leq n - r$, the output of $\mathcal{F}$ towards $\mathcal{Z}$ is the same as in the previous game $\mathbf{G}_5$. Observe that perfect correctness directly follows from the perfect correctness of $\mathcal{F}_{\ell\text{-}\mathsf{iPAKE}}$ and the r-robustness of the secret sharing, which is *always* able to correct up to $n - r$ errors.

Note that $\mathcal{F}$ still differs from the functionality $\mathcal{F}^M_{\mathsf{fPAKE}}$ in some aspects. First, it does not output randomly generated session keys towards $\mathcal{Z}$ for honest sessions. Furthermore, it reports all pass-strings to $\mathcal{S}$. We will take care of these remaining differences in the next games.

Game $\mathbf{G}_7$: In some cases, $\mathcal{F}$ generates a random session key when the other party is corrupted. Upon receiving a `NewKey` query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from $\mathcal{S}$, if there is a fresh record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$, and this is the first `NewKey` query for $\mathcal{P}_i$, $\mathcal{P}_i$ is honest and $\mathcal{P}_{1-i}$ corrupted and there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) > \delta$, we let $\mathcal{F}$ pick a new random key k from $\mathbb{F}_q$ and send $(\mathsf{sid}, \mathsf{k})$ to $\mathcal{P}_i$.

The simulation ensures that the record $(\mathcal{P}_i, \mathsf{pw}_i)$ is either compromised or interrupted (cf. description of the simulator in game $\mathbf{G}_4$). Thus, the modification has no effect since it only concerns fresh records.

Game $\mathbf{G}_8$: $\mathcal{F}$ generates a random session key for an honest session. Upon receiving a `NewKey` query $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ from $\mathcal{S}$, if there is a fresh record of the form $(\mathcal{P}_i, \mathsf{pw}_i)$, and this is the first `NewKey` query for $\mathcal{P}_i$, both parties are honest and the `NewKey` query is not *due*, we let $\mathcal{F}$ pick a new random key k from $\mathbb{F}_q$ and send $(\mathsf{sid}, \mathsf{k})$ to $\mathcal{P}_i$.

In other words, $\mathcal{F}$ now generates a random session key upon a first `NewKey` query for an honest party $\mathcal{P}_i$ with fresh record $(\mathcal{P}_i, \mathsf{pw}_i)$ where $\mathcal{P}_{1-i}$ is also honest, if (at least) one of the following events happen:

1. There is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) > \delta$; then, the probability that k_i was already random in game $\mathbf{G}_7$ is overwhelming according to Lemma 13.
2. No session key was sent to $\mathcal{P}_{1-i}$ yet; we just have to consider the case where there is a record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ with $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) \leq \delta$ since we already dealt with the other case in the first event. Then, the session key

in the previous game was $U + V$, which is distributed uniformly random in $\mathbb{F}_q$ since U, V are chosen uniformly random from $\mathbb{F}_q$.

3. If there was a session key sent to $\mathcal{P}_{1-i}$, the record $(\mathcal{P}_{1-i}, \mathsf{pw}_{1-i})$ was not fresh and thus interrupted or compromised at that time; since our simulation never issues `TestPwd` queries for honest sessions (in fact, game $\mathbf{G}_5$ states that $\mathcal{S}$ aborts upon man-in-the-middle attacks with overwhelming probability), this event can not happen in our simulation.

Game $\mathbf{G}_9$: Simulating without pass-string if both parties are honest. In case of receiving a $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from an honest $\mathcal{P}_i$, we modify $\mathcal{F}$ by forwarding only $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$. We now have to modify $\mathcal{S}$ to proceed simulation without knowing pw. Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{F}$ for an honest $\mathcal{P}_i$, we let $\mathcal{S}$ draw uniformly at random a "dummy" pass-string $\mathsf{pw}_{\mathcal{S}}$ and proceed the simulation of $\mathcal{P}_i$ using $\mathsf{pw}_{\mathcal{S}}$ as a pass-string. We first observe that $\mathcal{Z}$ is oblivious of k_i contained in the $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ query that $\mathcal{S}$ will eventually send to $\mathcal{F}$ during the simulation (since $\mathcal{F}$ never lets the simulator determine k_i for an honest session). This means that, informally, we have to show that $\mathcal{Z}$, knowing $\mathsf{pw}_i, \mathsf{pw}_{1-i}$ and seeing two transcripts, cannot tell which one was generated using $\mathsf{pw}_i, \mathsf{pw}_{1-i}$ and which one was generated using $\mathsf{pw}_{\mathcal{S}}, \mathsf{pw}_{1-i}$ with a random $\mathsf{pw}_{\mathcal{S}}$ unknown to $\mathcal{Z}$. But this is trivial since the distribution of the values U, V', K, L' does not depend on the pass-strings: U and V' are randomly chosen from $\mathbb{F}_q$. For K and L', $\mathcal{F}_{\ell\text{-iPAKE}}$ ensures that they are both randomly chosen from $\mathbb{F}_q^n$.

Game $\mathbf{G}_{10}$: Simulating without pass-string if someone is corrupted. Upon receiving $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ from $\mathcal{P}_i$ where $\mathcal{P}_{1-i}$ is corrupted, we modify $\mathcal{F}$ to only relay $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$. Additionally, we let $\mathcal{S}$ draw uniformly at random a "dummy" pass-string $\mathsf{pw}_{\mathcal{S}}$ and proceed the simulation of $\mathcal{P}_i$ using $\mathsf{pw}_{\mathcal{S}}$ as a pass-string. Note that due to the simulation described in game $\mathbf{G}_4$, $\mathcal{S}$ will ask a `TestPwd` query, and after this query the simulation described in that game is already independent of pw_i except when $\mathcal{F}$'s reply does not contain a mask. In this case, we now let $\mathcal{S}$ set the output of $\mathcal{F}_{\ell\text{-iPAKE}}$ for $\mathcal{P}_i$ to be a random pair (K, L).

Regarding indistinguishability, first note that in any case the input of $\mathcal{P}_i$ to $\mathcal{F}_{\ell\text{-iPAKE}}$ does not impact any values and thus we only have to argue further in case $\mathcal{S}$ is modified. Then, it holds that $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) > \gamma$. Thus, $\mathcal{P}_i$'s record will get interrupted and $\mathcal{P}_i$ will obtain a uniformly random session key from $\mathcal{F}$, meaning that we only have to argue indistinguishability of E, σ_E, vk (or $F, \sigma_F, \mathsf{vk}'$, respectively) generated with either K, L depending on pw_i (as in the previous game) or $K, L \xleftarrow{\$} \mathbb{F}_q^n$ (as in the current game). Opposed to the situation in game $\mathbf{G}_9$, note that now $\mathcal{Z}$ knows K' and L'.

Since $d(\mathsf{pw}_i, \mathsf{pw}_{1-i}) > \gamma = n - t - 1$, at most t components of K' are the same as K in $\mathbf{G}_9$ with large probability $1 - \frac{n-t}{q}$, and thus w.h.p. $\mathcal{Z}$ learns at most t components of C. Therefore, the t-privacy of the Secret Sharing scheme states that nothing is leaked about U. Hence the transcript of the current and previous games are indistinguishable.

Observe that now $\mathcal{F}$ is equal to $\mathcal{F}_{\mathsf{fPAKE}}^M$ and $\mathcal{S}$ is equal to the simulator described in Figure 30. The theorem thus follows.

The simulator $\mathcal{S}$, initialized with a security parameter λ, initializes the dummy adversary $\mathcal{A}$. $\mathcal{S}$ emulates an ideal labeled iPAKE functionality $\mathcal{F}_{\ell\text{-iPAKE}}$ as depicted in Figure 25 for all calling entities in the system[a]. Additionally, $\mathcal{S}$ interacts with an ideal functionality $\mathcal{F}^{M}_{\mathsf{fPAKE}}$ and a distinguisher, the environment $\mathcal{Z}$, via the following queries:

- **Upon receiving a query $(\texttt{NewSession}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{F}^{M}_{\mathsf{fPAKE}}$:** initialize a party $\mathcal{P}_i$ and connect it to $\mathcal{A}$.
  - If $\mathcal{P}_{1-i}$ is honest, $\mathcal{S}$ proceeds the UC protocol execution as described in Figure 29 using $\mathsf{pw}_{\mathcal{S}} \xleftarrow{\$} \mathbb{F}_p$ as pass-string for $\mathcal{P}_i$ and $\mathcal{S}$'s random coins. (Cf. game $\mathbf{G}_9$.)
  - If $\mathcal{P}_{1-i}$ is corrupted, then $\mathcal{S}$ waits until $\mathcal{P}_{1-i}$ submitted n queries to $\mathcal{F}_{\ell\text{-iPAKE}}$ and then assembles $\mathsf{pw}_{\mathcal{Z}} \in \mathbb{F}_p^n$ from them. $\mathcal{S}$ sends $(\texttt{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_{\mathcal{Z}})$ to $\mathcal{F}^{M}_{\mathsf{fPAKE}}$. If $\mathcal{S}$ receives back a mask M, let $I := [n] \setminus M$, and $\mathcal{S}$ sets up n $\mathbb{F}_q^2$-keys (K, L) with $(K, L)_t = (K', L')_t \ \forall t \in I$ and $(K, L)_t \xleftarrow{\$} \mathbb{F}_p \neq (K', L')_t \ \forall t \in M$, where (K', L') denotes the output of $\mathcal{F}_{\ell\text{-iPAKE}}$ towards $\mathcal{P}_{1-i}$. $\mathcal{S}$ now continues the simulation of $\mathcal{P}_i$ using (K, L) as outputs of $\mathcal{F}_{\ell\text{-iPAKE}}$. (Cf. game $\mathbf{G}_4$.) If $\mathcal{S}$ does not receive a mask, it sets the output of $\mathcal{F}_{\ell\text{-iPAKE}}$ for $\mathcal{P}_i$ to be $K, L \xleftarrow{\$} \mathbb{F}_q^n$. (Cf. game $\mathbf{G}_{10}$.)
- **If an internally simulated party $\mathcal{P}_i$ produces an output $(\mathsf{sid}, \mathsf{k}_i)$:** Send $(\texttt{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{k}_i)$ to $\mathcal{F}^{M}_{\mathsf{fPAKE}}$.
- **If $\mathcal{Z}$ sends $(\mathsf{sid}, M_{\mathcal{Z}}, \sigma_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}})$ to an honest party $\mathcal{P}_i$:** if $\mathcal{P}_{1-i}$ is honest, $\mathcal{S}$ aborts after the Vfy step in the protocol, regardless of its outcome. (Cf. game $\mathbf{G}_5$.)

Additionally, $\mathcal{S}$ forwards all other instructions from $\mathcal{Z}$ to $\mathcal{A}$ and reports all output of $\mathcal{A}$ towards $\mathcal{Z}$. Instructions of corrupting a player are only obeyed if they are received before the protocol started, i.e., before $\mathcal{S}$ received any `NewSession` query from $\mathcal{F}^{M}_{\mathsf{fPAKE}}$.

[a] An entity is any internally simulated ITM such as parties or the real-world adversary as well as ITMs outside $\mathcal{S}$ such as the distinguisher $\mathcal{Z}$.

Fig. 30. The Simulator $\mathcal{S}$ for $\mathsf{fPAKE}_{\mathsf{RSS}}$

H A Natural (But Failed) Approach to fPAKE

A natural idea for building a fPAKE is the use of a fuzzy extractor [DRS04, Boy04], that allows to extract a common secret from two strings close enough, and to compose it with a regular PAKE. This approach was introduced in [BDK+05] (Section 4). Their protocol uses the code-offset construction of a fuzzy sketch [DRS04], a.k.a. fuzzy commitment [JW99], to implement a fuzzy-extractor as a two-party primitive. It is presented in Figure 31.

Theorem 14. *The construction from Figure 31 cannot securely realize $\mathcal{F}^{P}_{\mathsf{fPAKE}}$.*

Proof. Consider the following attack by $\mathcal{Z}$. $\mathcal{Z}$ sends a randomly chosen pw as input to an honest $\mathcal{P}_0$ and obtains a sketch s from $\mathcal{A}$. It then computes $c \leftarrow s - \mathsf{pw}$

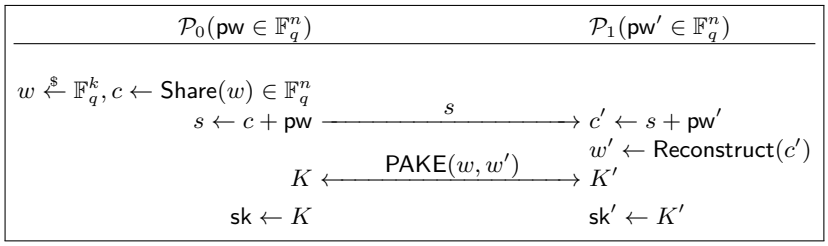

Fig. 31. A First Natural Construction (with code-offset fuzzy sketch and PAKE)

and outputs 1 if c is in the image of Share. In the real world, this happens with probability 1. Now assume there is a simulator $\mathcal{S}$ outputting a simulated sketch $\tilde{s}$ in the ideal world. Since $\mathcal{S}$ does not get to learn pw unless it succeeds at a `TestPwd` query, observe that this output may not depend on pw except with some small (but non-negligible) probability p, namely the probability of guessing a pass-string that makes $\mathcal{F}_{\mathsf{fPAKE}}^{P}$ output pw. Thus, with probability $1 - p \approx 1$, $\tilde{c} := \tilde{s} - \mathsf{pw}$ is randomly distributed in $\mathbb{F}_q^n$ and lies in the image of Share only with probability $1/q^{mn-l}$. More formally, the probability that $\mathcal{Z}$ outputs 1 in the ideal world is

$$\begin{aligned}\Pr[\tilde{c} \in Im(\mathsf{Share})] &= \Pr[\tilde{c} \in Im(\mathsf{Share}) | \mathcal{S} \text{ depends on } \mathsf{pw}] \cdot p \\ &\quad + \Pr[\tilde{c} \in Im(\mathsf{Share}) | \mathcal{S} \text{ does not depend on } \mathsf{pw}] \cdot (1 - p) \\ &\leq p + 1/q^{mn-l}(1 - p) \approx p.\end{aligned}$$