Fuzzy Password-Authenticated Key Exchange

Pierre-Alain Dupont[1,2,3(✉)], Julia Hesse[4], David Pointcheval[2,3], Leonid Reyzin[5], and Sophia Yakoubov[5]

[1] DGA, Paris, France
[2] DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France
{pierre-alain.dupont,david.pointcheval}@ens.fr
[3] INRIA, Paris, France
[4] Technische Universität Darmstadt, Darmstadt, Germany
julia.hesse@ens.fr
[5] Boston University, Boston, USA
reyzin@cs.bu.edu, sonka@bu.edu

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.

J. Hesse—Work done while at École Normale Supérieure.

J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10822, pp. 393–424, 2018.
https://doi.org/10.1007/978-3-319-78372-7_13

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 [15,23,29,39,46,49,66], physically unclonable functions (PUFs) [30,52,57,58,64], noisy channels [61], quantum information [9], and sensor readings of a common environment [32,33].

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, with first formal models introduced in [7,14]). 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.* [20] 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 [51] do the job. The case of active adversaries is covered by the literature on so-called robust extractors defined by Boyen *et al.* [13] and, more generally, by many papers on privacy amplification protocols secure against active adversaries, starting with the work of Maurer [45]. 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 [12] (with the exception of [19]).

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.* [9] and their one-message variants called fuzzy extractors (defined by Dodis *et al.* [26], further enhanced for active security starting by Renner and Wolf [54]) 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 [11, Sect. 5]. Even the PAKE-based construction of Boyen *et al.* [13] suffers from the same problem.

One notable exception is the construction of Canetti *et al.* [19], 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 [17].

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 [17]. 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.* [20]. 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 Sect. 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.* [13], 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 demonstrate this formally in the full version of this paper [28]). Because some information about the pass-strings is unconditionally revealed, high-entropy pass-strings are required. Thus, in order to realize our definition for arbitrary low-entropy pass-strings, we need to construct new protocols.

Realizing our fPAKE definition is easy using general two-party computation techniques for protocols with malicious adversaries and without authenticated channels [4]. 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 generic two-party computation 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 [6,63] and oblivious transfer (see [21] 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 [42], 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.[1] Mohassel and Franklin [48] and Huang *et al.* [34] suggest

[1] Gasti *et al.* [31] 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 as we know, ours is the first use of garbled circuits in the two-party fully malicious setting without calling on an expensive transformation.

a similar technique (known as "dual execution"), but at the cost of leaking a bit of the adversary's choice to the adversary. In contrast, our construction leaks nothing to the adversary at all (as long as the pass-strings are not close). 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.

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 [1,8]). 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 Fig. 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 only. However, these advantages come at the price of efficiency. Our protocols require 2–5 rounds of interaction, as opposed to many single-message protocols in the literature [19,25,60]. 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 [10, Sect. 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.* [20]

to work for pass-strings (a generalization of "passwords") that are similar, but not necessarily equal. Our definition uses measures of the distance $d(\mathsf{pw}, \mathsf{pw}')$ between pass-strings $\mathsf{pw}, \mathsf{pw}' \in \mathbb{F}_p^n$. In Sects. 3.3 and 4, Hamming distance is used, but in the generic construction of Sect. 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 ($\mathcal{P}_0$ and $\mathcal{P}_1$). 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.* [20], 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.* [20], we consider only static corruptions in the standard corruption model of Canetti [17]. 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 ($\mathcal{F}_{\mathsf{iPAKE}}$, Fig. 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 Fig. 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 $(\texttt{NewSession}, \mathsf{sid}, \mathsf{pw}_i)$ 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

- **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 $L_c(\mathsf{pw}_i, \mathsf{pw}'_i)$;
  - 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

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 Fig. 1, except that `TestPwd` is from Fig. 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 Fig. 1, leaks the correctness of the adversary's guess. That is, in the language of Fig. 2,

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

   The classical PAKE functionality from [20] 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 Fig. 1, except that `TestPwd` is from Fig. 2, with L_c and L_m that leak 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 Fig. 1, except that `TestPwd` is from Fig. 2, with

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

   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. The last two functionalities, though weaker, still provide meaningful security, especially when $\gamma = \delta$. Intuitively, this is because strong leakage only occurs when an adversary guesses a “close” pass-string, which enables him to authenticate as though he knows the real pass-string anyway.

In Sect. 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 Sect. 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 [63]. We briefly introduce this primitive in Sect. 3.1 and refer to Yakoubov [62] for a more thorough introduction.

The Yao's garbled circuit-based 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 Sect. 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, as suggested by Mohassel and Franklin [48] and Huang *et al.* [34], since a malicious evaluator (unlike a malicious garbler) cannot compromise the computation, by performing the protocol twice with each party playing each role once, we can protect against malicious behavior. They call this the *dual execution* protocol.

The dual execution protocol has the downside of allowing the adversary to specify and receive a single additional bit of leakage. It is important to note that because of this, dual execution cannot directly be used to instantiate fPAKE, because a single bit of leakage can be too much when the entropy of the pass-strings is low to begin with—a few adversarial attempts will uncover the entire pass-string. Our construction is as efficient as that of Mohassel *et al.* and Huang *et al.*, while guaranteeing no leakage to a malicious adversary in the case that the pass-strings used are not close. We describe how we achieve this in Sect. 3.1.3.

3.1 Building Blocks

In Sect. 3.1.1, we briefly review oblivious transfer. In Sect. 3.1.2, we review Yao's Garbled Circuits. In Sect. 3.1.3, we describe in more detail our take on the dual execution protocol, and how we avoid leakage to the adversary when the pass-strings used are dissimilar.

3.1.1 Oblivious Transfer (OT)

Informally, 1-out-of-2 Oblivious Transfer (see [21] 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 [63]. We refer to Yakoubov [62] for a more detailed description, as well as a summary of some of the Yao's garbled circuits optimizations [3,5,38,40,53,65]. Informally, Yao's garbled circuits are an asymmetric secure two-party computation scheme. They enable 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.

Below, we summarize the garbling scheme formalization of Bellare *et al.* [6], 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. $\mathsf{Gb}(1^\lambda, f) \rightarrow (F, e, d)$. 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. $\mathsf{En}(e, x) \rightarrow X$. The encoding algorithm En takes in the encoding information e and an input x, and returns a garbled input X.
3. $\mathsf{Ev}(F, X) \rightarrow Y$. The evaluation algorithm Ev takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.
4. $\mathsf{De}(d, Y) \rightarrow y$. 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* (each of which is essentially a random string), 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. The 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 then 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.* [6] 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 the full version of this paper [28], 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.

3.1.3 Malicious Security: A New Take on Dual Execution with Privacy-Correctness Tradeoffs

While Yao's garbled circuits are naturally secure against a malicious evaluator, they have the drawback of being insecure against a malicious garbler. 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 Yao's garbled circuits by using the cut-and-choose transformation [35,41,43]. To achieve a $2^{-\lambda}$ probability of cheating without detection, the parties need to exchange λ garbled circuits [41].[2] 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 of λ computational overhead, though, cut-and-choose is expensive, and too heavy a tool for fPAKE. Other, more efficient transformations such as LEGO [50] and authenticated garbling [59] exist as well, but those rely heavily on pre-processing, which cannot be used in fPAKE since it requires advance interaction between the parties.

Mohassel and Franklin [48] and Huang *et al.* [34] suggest an efficient transformation known as "dual execution": each party plays each role (garbler and evaluator) once, and then the two perform a comparison step on their outputs in a secure fashion. Dual execution incurs only a factor of 2 overhead over semi-honest garbled circuits. However, it does not achieve fully malicious security. It guarantees correctness, but reduces the privacy guarantee by allowing a malicious garbler to learn one bit of information of her choice. Specifically, if a malicious garbler garbles a wrong circuit, she can use the comparison step to learn one bit about the output of this wrong circuit on the other party's input. This one extra bit of information could be crucially important, violating the privacy of the evaluator's input in a significant way.

We introduce a tradeoff between correctness and privacy for boolean functions. For one of the two possible outputs (without loss of generality, '0'), we restore full privacy at the cost of correctness. The new privacy guarantee is that if the correct output is '0', then a malicious adversary cannot learn anything beyond this output, but if the correct output is '1', then she can learn a single bit of her choice. The new correctness guarantee is that a malicious adversary

[2] There are techniques [44] that improve this number in the amortized case when many computations are done—however, this does not fit our setting.

can cause the computation that should output '1' to output '0' instead, but not the other way around.

The main idea of dual execution is to have the two parties independently evaluate one another's circuits, learn the output values, and compare the output labels using a secure comparison protocol. In our construction, however, the parties need not learn the output values before the comparison. Instead, the parties can compare output labels *assuming* an output of '1', and if the comparison fails, the output is determined to be '0'.

More formally, let $d_0[0]$, $d_0[1]$ be the two output labels corresponding to $\mathcal{P}_0$'s garbled circuit, and $d_1[0]$, $d_1[1]$ be the two output labels corresponding to $\mathcal{P}_1$'s circuit. Let Y_0 be the output label learned by $\mathcal{P}_1$ as a result of evaluation, and Y_1 be the label learned by $\mathcal{P}_0$. The two parties securely compare $(d_0[1], Y_1)$ to $(Y_0, d_1[1])$; if the comparison succeeds, the output is "1".

Our privacy–correctness tradeoff is perfect for fPAKE. If the parties' inputs are similar, learning a bit of information about each other's inputs is not problematic, since arguably the small amount of noise in the inputs is a bug, not a feature. If the parties' inputs are not similar, however, we are guaranteed to have no leakage at all. We pay for the lack of leakage by allowing a malicious party to force an authentication failure even when authentication should succeed. However, either party can do so anyway by providing an incorrect input.

In Sect. 3.2.2, we describe our Yao's garbled circuit-based fPAKE protocol. Note that in this protocol, we omit the final comparison step; instead, we use the output lables $((d_0[1], Y_1)$ and $(Y_0, d_1[1]))$ to compute the agreed-upon key directly.

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 channels in mind. We therefore follow the framework of Canetti *et al.* [18], 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.* [4] to adapt it to the unauthenticated channel setting. More formally, we proceed in three steps:

1. First, in Sect. 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 Sect. 3.2.2, we build a protocol that securely realizes $\mathcal{F}_{\mathsf{RFE}}$ in the OT-hybrid model, assuming authenticated channels.
3. In Sect. 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 were 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.

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 records 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

3.2.1 The Randomized Fuzzy Equality Functionality

Figure 3 shows the randomized fuzzy equality functionality $\mathcal{F}^P_{\mathsf{RFE}}$, which is essentially what $\mathcal{F}^P_{\mathsf{fPAKE}}$ would look like assuming authenticated channels. The primary difference between $\mathcal{F}^P_{\mathsf{RFE}}$ and $\mathcal{F}^P_{\mathsf{fPAKE}}$ is that the only pass-string guesses allowed by $\mathcal{F}^P_{\mathsf{RFE}}$ are the ones actually used as protocol inputs; this limits the adversary to guessing by corrupting one of the participating parties, not through man in the middle attacks. Like $\mathcal{F}^P_{\mathsf{fPAKE}}$, if a pass-string guess is "similar enough", the entire pass-string is leaked. This leakage could be replaced with any other leakage from Sect. 2; $\mathcal{F}_{\mathsf{RFE}}$ would leak the correctness of the guess, $\mathcal{F}^M_{\mathsf{RFE}}$ 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.* [18], $\mathcal{F}^P_{\mathsf{fPAKE}}$ 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 Fig. 4 we introduce a protocol Π_{RFE} that securely realizes $\mathcal{F}^P_{\mathsf{RFE}}$ 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

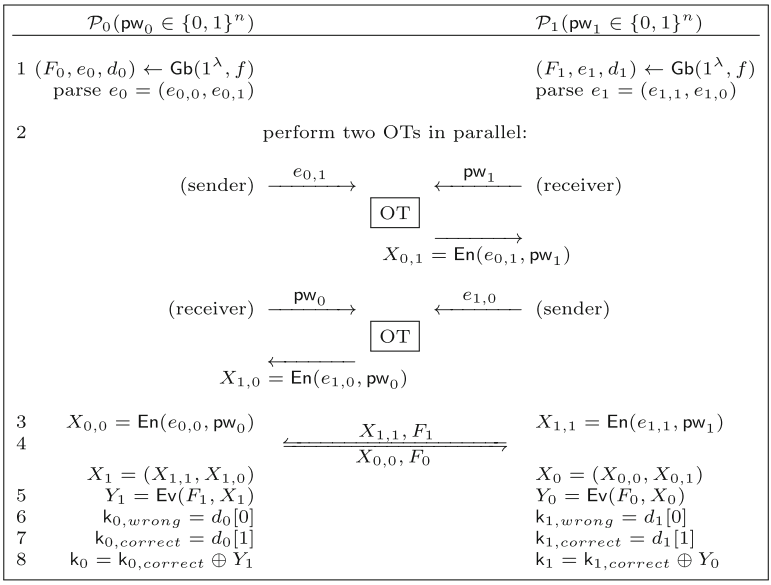

Fig. 4. A protocol Π_{RFE} realizing $\mathcal{F}^P_{\mathsf{RFE}}$ using Yao's garbled circuits and an Ideal OT Functionality. If at any point an expected message fails to arrive (or arrives malformed), the parties output a random key. Subscripts are used to indicate who produced the object in question. 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 not a problem.

adversaries by having each party play each role once, as describe in Sect. 3.1.3. 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 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 Π_{RFE} only realizes $\mathcal{F}_{\mathsf{RFE}}^{P}$, and not a stronger functionality with less leakage. We argue this assuming (without loss of generality) that $\mathcal{P}_1$ 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 $\mathcal{P}_1$ 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 some predicate p of $\mathcal{P}_1$'s choosing on the two pass-strings pw_0 and pw_1. Therefore, in the ideal world, the keys should also match or not match based on $p(\mathsf{pw}_0, \mathsf{pw}_1)$; otherwise, the environment will be able to distinguish between the two worlds. In order to make that happen, since the simulator does not know the predicate p in question, 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* $d(\mathsf{pw}, \mathsf{pw}') > \delta$*.*

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 the full version of this paper. 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 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 the full version of this paper [28].

3.2.3 From Split Randomized Fuzzy Equality to fPAKE

The Randomized Fuzzy Equality (RFE) functionality $\mathcal{F}^P_{\mathsf{RFE}}$ 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.* [4]. 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 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 Fig. 5. It is simplified from Fig. 1 in Barak *et al.* [4] 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 the full version of this paper [28], 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.
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.

The functionality $s\mathcal{F}_{\mathsf{RFE}}^{P}$ 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** $(\texttt{Init}, \mathsf{sid})$ **from a party** $\mathcal{P}_i \in \{\mathcal{P}_0, \mathcal{P}_1\}$, send $(\texttt{Init}, \mathsf{sid}, \mathcal{P}_i)$ to the adversary $\mathcal{S}$.
  - **Upon receiving a query** $(\texttt{Init}, \mathsf{sid}, \mathcal{P}_i, H, \mathsf{sid}_H)$ **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 $(\texttt{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to $\mathcal{P}_i$, and locally initialize a new instance of the original RFE functionality $\mathcal{F}_{\mathsf{RFE}}$ denoted $H\mathcal{F}_{\mathsf{RFE}}^{P}$, letting the adversary play the role of $\{\mathcal{P}_0, \mathcal{P}_1\} - H$ in $H\mathcal{F}_{\mathsf{RFE}}^{P}$.
- **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 $H\mathcal{F}_{\mathsf{RFE}}^{P}$. Otherwise, ignore the query.
  - **Upon receiving a query from the adversary** $\mathcal{S}$ **on behalf of** $\mathcal{P}_i$ **corresponding to set** H**,** if $H\mathcal{F}_{\mathsf{RFE}}^{P}$ is initialized and $\mathcal{P}_i \notin H$, then forward the query to $H\mathcal{F}_{\mathsf{RFE}}^{P}$. Otherwise, ignore the query.

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

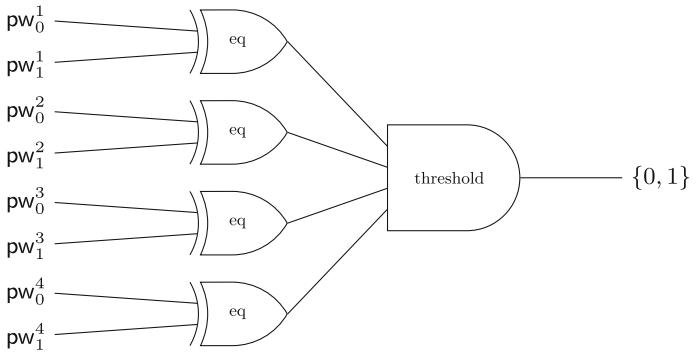

Fig. 6. The f circuit

This circuit, illustrated in Fig. 6, is very efficient to garble; it only requires n ciphertexts. Below, we briefly explain this garbling. Our explanation assumes familiarity with YGC literature [62, and references therein]. Briefly, garbled gadget labels [3] 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 pre-processing on the garbler's side eliminates the need to send $X_{0,0}$ and $X_{1,1}$ in Fig. 4.)
2. Compute a n-input threshold gate, as illustrated in Fig. 6 of Yakoubov [62]. 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 (Fig. 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 the full version of this paper [28], 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. 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 Building Blocks

4.1.1 Robust Secret Sharing

We recall the definition of a robust secret sharing scheme, slightly simplified for our purposes from Cramer *et al.* [22]. 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 $(c_i)_{i \in A}$.

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 $c_A = \tilde{c}_A$, it holds that $\mathsf{Reconstruct}(\tilde{c}) = s$.*

In other words, an (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 it contains, and is thus equal to q^k. The weight of a word $w \in \mathbb{F}_q^n$ is the number of its 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, the 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 [55].

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 [47].

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 [7].

The standard PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ from [20]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 Fig. 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 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

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 $\mathcal{P}_1$ via the following queries:

- **Upon receiving a query (`NewSession`, sid, pw_i) from party $\mathcal{P}_i$:**
  - Send (`NewSession`, 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 (`TestPwd`, sid, $\mathcal{P}_i$, 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 (`NewKey`, sid, $\mathcal{P}_i$, 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 (sid, 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 (sid, sk') to $\mathcal{P}_i$;
  - In any other case, pick a new random key sk' of length λ and send (sid, sk') to $\mathcal{P}_i$.
  - Mark the record $(\mathcal{P}_i, \mathsf{pw}_i)$ as `completed`.

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

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, as shown in the full version of this paper [28]. The resulting functionality $\mathcal{F}_{\ell\text{-iPAKE}}$ idealizes what we call *labeled implicit-only PAKE* (or ℓ-iPAKE for short), resembling the notion of labeled public key encryption as formalized in [56]. 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.

A UC-Secure ℓ-iPAKE Protocol. In the seminal paper by Bellovin and Merritt [8], the Encrypted Key Exchange protocol (EKE) is proposed, which is essentially a Diffie-Hellman [24] 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.* [7] under the name EKE2. We present its labeled variant in Fig. 8. The idea of appending the label to the symmetric key is taken from [1]. We prove security of this protocol in the $\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{IC}}, \mathcal{F}_{\mathsf{CRS}}$-hybrid model. That is, we use an ideal random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ to model the hash function, and ideal cipher functionality $\mathcal{F}_{\mathsf{IC}}$ to model the encryption scheme and assume a publicly available common reference string modeled by $\mathcal{F}_{\mathsf{CRS}}$. Formal definitions of these functionalities are given in the full version of this paper [28].

$A(\mathsf{pw} \in \mathbb{F}_p)$		$B(\mathsf{pw}' \in \mathbb{F}_p)$
$x \xleftarrow{\$} \mathbb{F}_P,\ \ell \leftarrow \mathcal{L},\ X \leftarrow g^x$		
$X^* \leftarrow \mathcal{E}_{\mathsf{pw}\|\|\ell}(X)$	$\xrightarrow{\ell, X^*}$	$y \xleftarrow{\$} \mathbb{F}_P,\ \ell' \leftarrow \mathcal{L},\ Y \leftarrow g^y$
	$\xleftarrow{\ell', Y^*}$	$Y^* \leftarrow \mathcal{E}_{\mathsf{pw}'\|\|\ell'}(Y)$
$Z \leftarrow \mathcal{D}_{\mathsf{pw}\|\|\ell'}(Y^*)^x$		$Z' \leftarrow \mathcal{D}_{\mathsf{pw}'\|\|\ell}(X^*)^y$
$k \leftarrow H(X^*, Y^*, Z)$		$k' \leftarrow H(X^*, Y^*, Z')$
output (ℓ', k)		output (ℓ, k')

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 $\mathbb{F}_p \times \mathcal{L}$, where $\mathcal{L}$ is the label space.

Theorem 4. *If the CDH assumption holds in $\mathbb{G}$, the protocol EKE2 depicted in Fig. 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 the full version of this paper [28].

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 the full version of this paper [28] 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 Fig. 9. It works as follows:

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) [27,36] 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 Lemma 3 can also be used here instead of the code-offset construction.

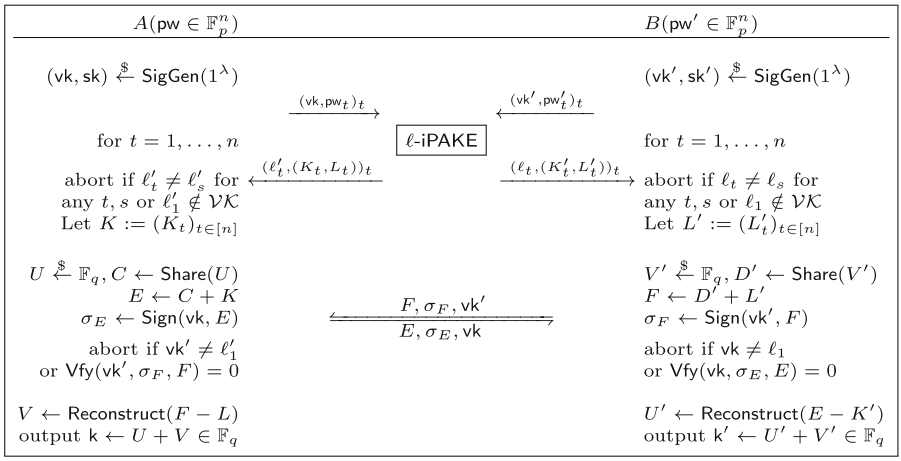

Fig. 9. Protocol $\mathsf{fPAKE}_{\mathsf{RSS}}$ where $q \approx 2^\lambda$ is a prime number and $+$ denotes the group operation in $\mathbb{F}_q^n$. $(\mathsf{Share}, \mathsf{Reconstruct})$ is a Robust Secret Sharing scheme with $\mathsf{Share} : \mathbb{F}_q \to \mathbb{F}_q^n$, and $(\mathsf{SigGen} \to \mathcal{VK} \times \mathcal{SK}, \mathsf{Sign}, \mathsf{Vfy})$ 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.

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

We show that our protocol realizes functionality $\mathcal{F}_{\mathsf{fPAKE}}^M$ in the $\mathcal{F}_{\ell\text{-iPAKE}}$-hybrid model. In a nutshell, the idea is to simulate without the pass-strings by adjusting the keys outputted by $\mathcal{F}_{\ell\text{-iPAKE}}$ 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* $\mathcal{F}_{\ell\text{-iPAKE}}$*-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* $\mathcal{F}_{\ell\text{-iPAKE}}$*-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[3]. 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 [20] and recalled in the full version of this paper [28] provides an interface to the adversary to test

[3] Instead of labels and one-time signature, one could just sign all the messages, as would be done using the split-functionality [4], 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.

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 Sect. 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 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[4].

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 Sect. 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 [37]. This protocol also securely realizes our $\mathcal{F}_{\ell\text{-iPAKE}}$ functionality[5]. However, it is more costly than our ℓ-iPAKE protocol since both messages each contain one non-interactive zero knowledge proof.

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

[5] 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 [37], 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.

Since fPAKE implies a regular PAKE (simply set $\delta = 0$), [20] gives strong evidence that we cannot hope to realize $\mathcal{F}_{\mathsf{fPAKE}}$ without a CRS.

5 Comparison of fPAKE Protocols

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

Then, in Fig. 11, we describe the efficiency of the constructions when concrete primitives (OT/ℓ-iPAKE) are used to instantiate them. $\mathsf{fPAKE}_{\mathsf{RSS}}$ is instantiated as the construction in Fig. 9 with the ℓ-iPAKE in Fig. 8 and an RSS. $\mathsf{fPAKE}_{\mathsf{YGC}}$ is instantiated as the construction in Fig. 4 with the UC-secure oblivious transfer protocol of Chou and Orlandi [21], with the garbling scheme of Bal *et al.* [3], and with the split functionality transformation of Barak *et al.* [4]. Though $\mathsf{fPAKE}_{\mathsf{YGC}}$ can handle any efficiently computable notion of distance, Fig. 11 assumes that both constructions use Hamming distance (and that, specifically, $\mathsf{fPAKE}_{\mathsf{YGC}}$ uses the circuit described in Fig. 6). We describe efficiency in terms of sub-operations (per-party, not in aggregate).

	Assumptions	Threshold δ	Gap $\gamma - \delta$
$\mathsf{fPAKE}_{\mathsf{RSS}}$	UC-secure ℓ-iPAKE	$< n/2$	δ
$\mathsf{fPAKE}_{\mathsf{YGC}}$	(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. $\mathsf{fPAKE}_{\mathsf{RSS}}$ is the construction in Fig. 9. $\mathsf{fPAKE}_{\mathsf{YGC}}$ is the construction in Fig. 4 with the split functionality transformation of Barak *et al.* [4].

	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
$\mathsf{fPAKE}_{\mathsf{YGC}}$	$\{0,1\}^\lambda$	5	$3n+2$	$4n+7$	$2n$	n	–	–	1	5	5

Fig. 11. Efficiency (in terms of sub-operations) of the two constructions. $\mathsf{fPAKE}_{\mathsf{RSS}}$ is the construction in Fig. 9 instantiated with the ℓ-iPAKE in Fig. 8. $\mathsf{fPAKE}_{\mathsf{YGC}}$ is the construction in Fig. 4 instantiated with the UC-secure oblivious transfer protocol of Chou and Orlandi [21], the garbling scheme of Bal *et al.* [3], and with the split functionality transformation of Barak *et al.* [4].

Note that these concrete primitives each have their own set of required assumptions. Specifically, the ℓ-iPAKE in Fig. 8 requires a random oracle (RO), ideal cipher (IC) and common reference string (CRS). The oblivious transfer protocol of Chou and Orlandi [21] requires a random oracle. The garbling scheme of Bal *et al.* [3] 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. The last of these is combined with sending the garbled circuits. Two additional rounds of communication come from the split functionality transformation. The need for signatures also arises from the split functionality transformation.

Efficiency Optimizations to $\mathsf{fPAKE}_{\mathsf{YGC}}$. We can make several small efficiency improvements to the $\mathsf{fPAKE}_{\mathsf{YGC}}$ construction which are not reflected in Fig. 11. First, instead of using the split functionality transformation of Barak *et al.* [4], we can use the split functionality of Camenisch *et al.* [16]. 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. Second, if the pass-strings are more than λ bits long (where λ is the security parameter), OT extensions that are secure against malicious adversaries [2] can be used. If the pass-strings are fewer than λ bits long, then nothing is to be gained from using OT extensions, since OT extensions require λ "base OTs". However, if the pass-strings are longer—say, if they are some biometric measurement that is thousands of bits long—then OT extensions would save on the number of public key operations, at the cost of an extra round of communication.

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

1. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_22
2. Afshar, A., Hu, Z., Mohassel, P., Rosulek, M.: How to efficiently evaluate RAM programs with malicious security. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 702–729. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_27

3. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for boolean and arithmetic circuits. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 565–577. ACM Press, New York (2016)
4. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 361–377. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_22
5. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990
6. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACM Press, New York (2012)
7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11
8. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press, May 1992
9. Bennett, C.H., Brassard, G., Robert, J.M.: Privacy amplification by public discussion. SIAM J. Comput. **17**(2), 210–229 (1988)
10. Bitansky, N., Canetti, R., Kalai, Y.T., Paneth, O.: On virtual grey box obfuscation for general circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 108–125. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_7
11. Blanton, M., Hudelson, W.M.P.: Biometric-based non-transferable anonymous credentials. In: Qing, S., Mitchell, C.J., Wang, G. (eds.) ICICS 2009. LNCS, vol. 5927, pp. 165–180. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-11145-7_14
12. Boyen, X.: Reusable cryptographic fuzzy extractors. In: Atluri, V., Pfitzmann, B., McDaniel, P. (eds.) ACM CCS 2004, pp. 82–91. ACM Press, New York (2004)
13. Boyen, X., Dodis, Y., Katz, J., Ostrovsky, R., Smith, A.: Secure remote authentication using biometric data. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 147–163. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_9
14. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_12
15. Brostoff, S., Sasse, M.A.: Are passfaces more usable than passwords? A field trial investigation. In: McDonald, S., Waern, Y., Cockton, G. (eds.) People and Computers XIV – Usability or Else!, pp. 405–424. Springer, London (2000). https://doi.org/10.1007/978-1-4471-0515-2_27
16. Camenisch, J., Casati, N., Gross, T., Shoup, V.: Credential authenticated identification and key exchange. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 255–276. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_14
17. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001

18. Canetti, R., Dachman-Soled, D., Vaikuntanathan, V., Wee, H.: Efficient password authenticated key exchange via oblivious transfer. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 449–466. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_27
19. Canetti, R., Fuller, B., Paneth, O., Reyzin, L., Smith, A.: Reusable fuzzy extractors for low-entropy distributions. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 117–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_5
20. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
21. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 40–58. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_3
22. Cramer, R., Damgård, I.B., Döttling, N., Fehr, S., Spini, G.: Linear secret sharing schemes from error correcting codes and universal hash functions. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 313–336. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_11
23. Daugman, J.: How iris recognition works. IEEE Trans. Circuits Syst. Video Technol. **14**(1), 21–30 (2004)
24. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
25. Dodis, Y., Kanukurthi, B., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. IEEE Trans. Inf. Theory **58**(9), 6207–6222 (2012). https://doi.org/10.1109/TIT.2012.2200290
26. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. SIAM J. Comput. **38**(1), 97–139 (2008)
27. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_31
28. Dupont, P.A., Hesse, J., Pointcheval, D., Reyzin, L., Yakoubov, S.: Fuzzy authenticated key exchange. Cryptology ePrint Archive, Report 2017/1111 (2017). https://eprint.iacr.org/2017/1111
29. Ellison, C., Hall, C., Milbert, R., Schneier, B.: Protecting secret keys with personal entropy. Future Gener. Comput. Syst. **16**(4), 311–318 (2000)
30. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon physical random functions. In: Atluri, V. (ed.) ACM CCS 2002, pp. 148–160. ACM Press, New York (2002)
31. Gasti, P., Sedenka, J., Yang, Q., Zhou, G., Balagani, K.S.: Secure, fast, and energy-efficient outsourced authentication for smartphones. Trans. Info. For. Sec. **11**(11), 2556–2571 (2016). https://doi.org/10.1109/TIFS.2016.2585093
32. Han, J., Chung, A., Sinha, M.K., Harishankar, M., Pan, S., Noh, H.Y., Zhang, P., Tague, P.: Do you feel what I hear? Enabling autonomous IoT device pairing using different sensor types. In: IEEE Symposium on Security and Privacy (2018)
33. Han, J., Harishankar, M., Wang, X., Chung, A.J., Tague, P.: Convoy: physical context verification for vehicle platoon admission. In: 18th ACM International Workshop on Mobile Computing Systems and Applications (HotMobile) (2017)

34. Huang, Y., Katz, J., Evans, D.: Quid-Pro-Quo-tocols: strengthening semi-honest protocols with dual execution. In: 2012 IEEE Symposium on Security and Privacy, pp. 272–284. IEEE Computer Society Press, May 2012
35. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 18–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_2
36. Juels, A., Wattenberg, M.: A fuzzy commitment scheme. In: ACM CCS 1999, pp. 28–36. ACM Press, November 1999
37. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_18
38. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_25
39. Kolesnikov, V., Rackoff, C.: Password mistyping in two-factor-authenticated key exchange. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 702–714. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_57
40. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
41. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_1
42. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_20
43. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. J. Cryptol. **28**(2), 312–350 (2015)
44. Lindell, Y., Riva, B.: Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_27
45. Maurer, U.: Information-theoretically secure secret-key agreement by NOT authenticated public discussion. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 209–225. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_15
46. Mayrhofer, R., Gellersen, H.: Shake well before use: intuitive and secure pairing of mobile devices. IEEE Trans. Mob. Comput. **8**(6), 792–806 (2009)
47. McEliece, R.J., Sarwate, D.V.: On sharing secrets and Reed-Solomon codes. Commun. ACM **24**(9), 583–584 (1981). http://doi.acm.org/10.1145/358746.358762
48. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_30

49. Monrose, F., Reiter, M.K., Wetzel, S.: Password hardening based on keystroke dynamics. Int. J. Inf. Secur. **1**(2), 69–83 (2002)
50. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_22
51. Nisan, N., Zuckerman, D.: More deterministic simulation in logspace. In: 25th ACM STOC, pp. 235–244. ACM Press, May 1993
52. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589), 2026–2030 (2002)
53. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15
54. Renner, R., Wolf, S.: The exact price for unconditionally secure asymmetric cryptography. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 109–125. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_7
55. Roth, R.: Introduction to Coding Theory. Cambridge University Press, New York (2006)
56. Shoup, V.: A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112 (2001). http://eprint.iacr.org/2001/112
57. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th Annual Design Automation Conference, pp. 9–14. ACM (2007)
58. Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 369–383. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_29
59. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 21–37. ACM Press, New York (2017)
60. Woodage, J., Chatterjee, R., Dodis, Y., Juels, A., Ristenpart, T.: A new distribution-sensitive secure sketch and popularity-proportional hashing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 682–710. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_23
61. Wyner, A.D.: The wire-tap channel. Bell Syst. Tech. J. **54**, 1355–1387 (1975)
62. Yakoubov, S.: A gentle introduction to Yao's garbled circuits (2017). http://web.mit.edu/sonka89/www/papers/2017ygc.pdf
63. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (Oct 1986)
64. Yu, M.D.M., Devadas, S.: Secure and robust error correction for physical unclonable functions. IEEE Des. Test **27**(1), 48–65 (2010)
65. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole: reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8
66. Zviran, M., Haga, W.J.: A comparison of password techniques for multilevel authentication mechanisms. Comput. J. **36**(3), 227–237 (1993)