Randomized Half-Ideal Cipher on Groups with applications to UC (a)PAKE

Bruno Freitas Dos Santos[0009−0009−5474−0008], Yanqi Gu[0000−0001−6577−2704], and Stanislaw Jarecki[0000−0002−5055−2407]

University of California, Irvine. Email: {brunof,yanqig1,sjarecki}@uci.edu

Abstract. An Ideal Cipher (IC) is a cipher where each key defines a random permutation on the domain. Ideal Cipher on a group has many attractive applications, e.g., the *Encrypted Key Exchange* (EKE) protocol for Password Authenticated Key Exchange (PAKE) [10], or asymmetric PAKE (aPAKE) [41, 37]. However, known constructions for IC on a group domain all have drawbacks, including key leakage from timing information [15], requiring 4 hash-onto-group operations if IC is an 8-round Feistel [27], and limiting the domain to half the group [12] or using variable-time encoding [57, 49] if IC is implemented via (quasi-) bijections from groups to bitstrings [41].

We propose an IC relaxation called a *(Randomized) Half-Ideal Cipher* (HIC), and we show that HIC on a group can be realized by a *modified 2-round Feistel* (m2F), at a cost of 1 hash-onto-group operation, which beats existing IC constructions in versatility and computational cost. HIC weakens IC properties by letting part of the ciphertext be non-random, but we exemplify that it can be used as a drop-in replacement for IC by showing that EKE [10] and aPAKE of [41] realize respectively UC PAKE and UC aPAKE even if they use HIC instead of IC. The m2F construction can also serve as IC domain extension, because m2F constructs HIC on domain D from an RO-indifferentiable hash onto D and an IC on 2κ-bit strings, for κ a security parameter. One application of such extender is a modular lattice-based UC PAKE using EKE instantiated with HIC and anonymous lattice-based KEM.

1 Introduction

The Ideal Cipher Model (ICM) dates back to the work of Shannon [56], and it models a block cipher as an Ideal Cipher (IC) oracle, where every key, even chosen by the attacker, defines an independent random permutation.[1] Formally, an efficient adversary who evaluates a block cipher on any key k of its choice cannot distinguish computing the cipher on that key in the forward and backward direction from an interaction with oracles $E_k(\cdot)$ and $E_k^{-1}(\cdot)$, where $\{E_i\}$ is a family of random permutations on the cipher domain. The Ideal Cipher Model has seen a variety of applications in cryptographic analysis, e.g. [58, 54, 34, 55,

[1] This is an extended version of a paper which appears in Eurocrypt'23 [36].

48, 29, 16, 45], e.g. the analysis of the Davies-Meyer construction of a collision-resistant hash [55, 16], of the Even-Mansour construction of a cipher from a public pseudorandom permutation [34], or of the DESX method for key-length extension for block ciphers [48]. A series of works [32, 23, 44, 24, 27] shows that ICM is equivalent to the Random Oracle Model (ROM) [9]. Specifically, these papers show that n-round Feistel, where each round function is a Random Oracle (RO), implements IC for some n, and the result of Dai and Steinberger [27] shows that $n = 8$ is both sufficient and necessary. Other IC constructions include iterated Even-Mansour and key alternating ciphers [26, 5, 33], wide-input (public) random permutations [14, 13, 25], and domain extension mechanisms, e.g. [22, 42], constructions based on

Ideal Ciphers on Groups: Applications. All the IC applications above consider IC on a domain of fixed-length bitstrings. However, there are also attractive applications of IC whose domain is a *group*. A prominent example is a Password Authenticated Key Exchange (PAKE) protocol called *Encrypted Key Exchange* (EKE), due to Bellovin and Meritt [10]. EKE is a compiler from plain key exchange (KE) whose messages are pseudorandom in some domain D, and it implements a secure PAKE if parties use an IC on domain D to password-encrypt KE messages.[2] The EKE solution to PAKE is attractive because it realizes UC PAKE given any key-private (a.k.a. anonymous) KEM [7], or KE with a mild "random message" property, at a cost which is the same as the underlying KE(M) *if* the cost of IC on KE(M) message domain(s) is negligible compared to the cost of KE(M) itself. However, instantiating EKE with e.g. Diffie-Hellman KE (DH-KE) [30] requires an IC on a group because DH-KE messages are random group elements.

Recently Gu et al. [41] and Freitas et al. [37] extended the EKE paradigm to cost-minimal compilers which create UC *asymmetric* PAKE (aPAKE), i.e. PAKE for the client-server setting where one party holds a one-way hash of the password instead of a password itself, from any key-hiding Authenticated Key Exchange (AKE). The AKE-to-aPAKE compilers of [41, 37] are similar to the "EKE" KE-to-PAKE compiler of [10] in that they also require IC-encryption of KE-related values, but they use IC to password-encrypt a KEM public key rather than KE protocol messages. The key-hiding AKE's exemplified in [41, 37], namely HMQV [50] and 3DH [52], are variants and generalizations of DH-KE where public keys are group elements, hence the AKE-to-aPAKE compilers of [41, 37] instantiated this way also require IC on a group.

Ideal Ciphers on Groups: Existing Constructions. The above motivates searching for efficient constructions of IC on a domain of an arbitrary group. Note first that a standard block cipher on a bitstring domain does not work. The elements of any group G can be encoded as bitstrings of some fixed length

[2] Bellare et al. [8] showed that EKE+IC is a game-based secure PAKE, then Abdalla et al. [2] showed that EKE variant with explicit key confirmation realizes UC PAKE, and recently McQuoid et al. [53] showed that a round-minimal EKE variant realizes UC PAKE as well (however, see more on their analysis below).

n, but unless these encodings cover almost all n-bit strings, i.e. unless $(1-|G|/2^n)$ is negligible, encrypting G elements under a password using IC on n-bit strings exposes a scheme to an offline dictionary attack, because the adversary can decrypt a ciphertext under any password candidate and test if the decrypted plaintext encodes a G element.

Black and Rogaway [15] showed an elegant black-box solution for an IC on G given an IC on n-bit strings provided that $c = (2^n/|G|)$ is a constant: To encrypt element $x \in G$ under key k, use the underlying n-bit IC in a loop, i.e. set x_0 to the n-bit encoding of x, and $x_{i+1} = \mathsf{IC.Enc}_k(x_i)$ for each $i \geq 0$, and output as the ciphertext the first x_i for $i \geq 1$ s.t. x_i encodes an element of group G. (Decryption works the same way but using $\mathsf{IC.Dec}$.) This procedure takes expected c uses of $\mathsf{IC.Enc}$, but timing measurement of either encryption or decryption leaks roughly $\log c$ bits of information on key k per each usage, because given the ciphertext one can eliminate all keys which form decryption cycles whose length does not match the length implied by the timing data.

To the best of our knowledge there are only two other types of constructions of IC on a group. First, the work of [32, 23, 44, 24, 27] shows that n-round Feistel network implements an IC for $n \geq 8$. Although not stated explicitly, these results imply a (randomized) IC on a group, where one Feistel wire holds group elements, the xor gates on that wire are replaced by group operations, and hashes onto that wire are implemented as RO hashes onto the group. However, since $n = 8$ rounds is minimal [27], this construction incurs four RO hashes onto a group per cipher operation. Whereas there is progress regarding RO-indifferentiable hashing on Elliptic Curve (EC) groups, see e.g. [35], current implementations report an RO hash costs in the ballpark of 25% of scalar multiplication. Hence, far from being negligible, the cost of IC on group implemented in this way would roughly equal the DH-KE cost in the EKE compiler. The second construction of (randomized) IC combines any (randomized) quasi-bijective encoding of group elements as bitstrings with an IC on the resulting bitstrings [41]. However, we know of only two quasi-bijective encodings for Elliptic Curve groups, Elligator2 of Bernstein et al. [12] and Elligator^2 of Tibouchi et al. [57, 49], and both have some practical disadvantages. Elligator2 works for only some elliptic curves, and it can encode only half the group elements, which means that any application has to re-generate group elements until it finds one in the domain of Elligator2. Elligator^2 works for a larger class of curves, but its encoding procedure is non-constant time and it appears to be significantly more expensive than one RO hash onto a curve. Elligator^2 also encodes each EC element as a pair of underlying field elements, effectively doubling the size of the EC element representation.

IC Alternative: Programmable-Once Public Function. An alternative path was recently charted by McQuoid et al. [53], who showed that a 2-round Feistel, with one wire holding group elements, implements a randomized cipher on a group which has some IC-like properties, which [53] captured in a notion of Programmable Once Public Function (POPF). Moreover, they argue that POPF can replace IC in several applications, exemplifying it with an argument that EKE realizes UC PAKE if password encryption is implemented with a

POPF in place of IC. This would be very attractive because if 2-round Feistel can indeed function as an IC replacement in applications like the PAKE of [10] or the aPAKE's of [41, 37], this would form the most efficient and flexible implementation option for these protocols, because it works for any group which admits RO-indifferentiable hash, and it uses just one such hash-onto-group per cipher operation.

However, it seems difficult to use the POPF abstraction of [53] as a replacement for IC in the above applications because the POPF notion captures 2-round Feistel properties with game-based properties which appear not to address *non-malleability*. For that reason we doubt that it can be proven that UC PAKE is realized by EKE with IC replaced by POPF as defined in [53]. (See below for more details.) The fact that the POPF abstraction appears insufficient does not preclude that UC PAKE can be realized by EKE with encryption implemented as 2-round Feistel, but such argument would not be modular. Moreover, each application which uses 2-round Feistel in place of IC would require a separate non-modular proof. Alternatively, one could search for a "POPF+" abstraction, realized by a 2-round Feistel, which captures sufficient non-malleability properties to be useful as an IC replacement in PAKE applications, but in this work we chose a different route.

Our Results: Modified 2-Feistel as (Randomized) Half-Ideal Cipher. Instead of trying to work with 2-Feistel itself, we show that adding a block cipher BC to one wire in 2-Feistel makes this transformation non-malleable, and we capture the properties of this construction in the form of a UC notion we call a (Randomized) Half-Ideal Cipher (HIC). In Figure 1 we show a simple pictorial comparison of 2-Feistel, denoted $\mathsf{2F}$, and our modification, denoted $\mathsf{m2F}$. The modified 2-Feistel has the same efficiency and versatility as the 2-Feistel used by McQuoid et al. [53]: It works for any group with an RO-indifferentiable hash onto a group, it runs in fixed time, and it requires only one RO hash onto a group per cipher operation.

One drawback of $\mathsf{m2F}$ is that the ciphertext is longer than the plaintext by 2κ bits, where κ is a security parameter. However, that is less than any IC implementation above (including POPF, which does not realize IC) except for Elligator2: IC results from n-round Feistel have loose security bounds, hence they need significantly longer randomness to achieve the same provable security; Elligator2 adds κ bits for general moduli, due to encoding of field elements as random bitstrings; Elligator^2 uses an additional field element, which adds at least 2κ bits, plus another κ bits for the field-onto-bits encoding; Finally, 2-Feistel requires at least 3κ bits of randomness when used in EKE [53].

The UC HIC notion is a relaxation of an Ideal Cipher notion, but it does not prevent applicability in protocols like [10, 41, 37], which we exemplify by showing that the following protocols remain secure with (any realization of) IC replaced by (any realization of) HIC:

(I) UC PAKE is realized by an EKE variant with IC replaced by HIC, using round-minimal KE with a random-message property;

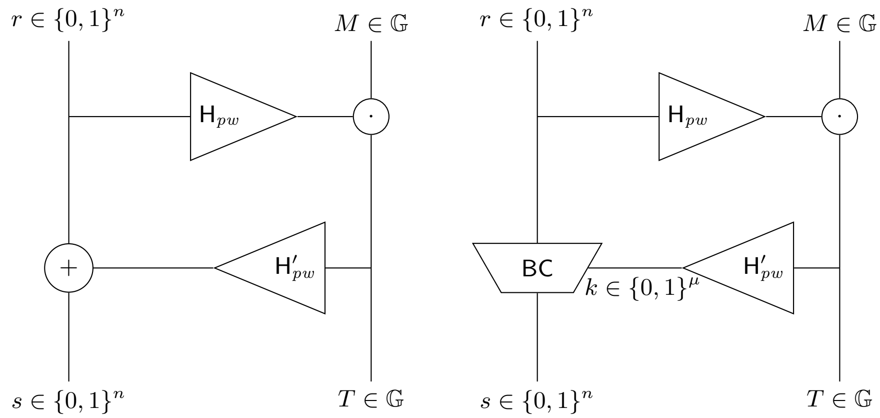

Fig. 1: Left: two-round Feistel ($\mathsf{2F}$) used in McQuoid et al. [53]; Right: our circuit $\mathsf{m2F}$. The change from $\mathsf{2F}$ to $\mathsf{m2F}$ is small: If $k = H'(pw, T)$, then $\mathsf{2F}$ sets $s = k \oplus r$, whereas $\mathsf{m2F}$ sets $s = \mathsf{BC.Enc}(k, r)$, where BC is a block cipher.

(II) UC PAKE is realized by an EKE variant with IC replaced by HIC, using anonymous KEM with a uniform public keys property;
(III) UC aPAKE is realized by KHAPE [41] with IC replaced by HIC, using key-hiding AKE.

Regarding the first two proofs, we are not aware of full proofs exhibited for the corresponding statements where these EKE variants use IC instead of HIC, but the third proof follows the blueprint of the proof given in [41] for the KHAPE protocol using IC, and it exemplifies how little such proof changes if IC is replaced by HIC.

Half-Ideal Cipher. The first difference between IC on group $\mathbb{G}$ and HIC on group $\mathbb{G}$ is that the latter is a cipher on an extended domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$ where $\mathcal{R} = \{0,1\}^n$ is the randomness space, for $n \geq 2\kappa$ where κ is the security parameter. In the decryption direction, HIC acts exactly like IC on domain $\mathcal{D}$, i.e. unless ciphertext $c \in \mathcal{D}$ is already associated with some plaintext in the permutation table defined by key k, an adversarial decryption of c under key k returns a random plaintext m, chosen by the HIC functionality with uniform distribution over those elements in domain $\mathcal{D}$ which are not yet assigned to any ciphertext in the permutation table for key k. However, in the encryption direction HIC is only *half-ideal* in the following sense: If plaintext m is not yet associated with any ciphertext in the permutation table for key k then encryption of m under key k returns a ciphertext $c = (s, T) \in \mathcal{D} = \mathcal{R} \times \mathbb{G}$ s.t. the $T \in \mathbb{G}$ part of c *can be freely specified by the adversary*, and the $s \in \mathcal{R}$ part of c is then chosen by the HIC functionality at random with uniform distribution over s's s.t. $c = (s, T)$ is not yet assigned to any plaintext in the permutation table for key k. In short, HIC decryption on any (k, c) returns a random plaintext m (subject to the constraint that $\mathsf{HIC}(k, \cdot)$ is a permutation on $\mathcal{D}$), but HIC encryption on any (k, m) returns $c = (s, T)$ s.t. T can be correlated with other values in an arbitrary way, which is modeled by allowing the adversary to choose it, but s is random (subject to the constraint that $\mathsf{HIC}(k, \cdot)$ is a permutation).[3]

Intuitively, the reason the adversarial ability to manipulate part of IC ciphertext does not affect typical IC applications is that these applications typically rely on the following properties of IC: (1) that decryption of a ciphertext on any other key from the one used in encryption outputs a random plaintext, (2) that any change to a ciphertext implies that the corresponding plaintext is random and hence uncorrelated to the plaintext in the original ciphertext, and (3) that no two encryption operations can output the same ciphertext, regardless of the keys used, and moreover that the simulator can straight-line extract the unique key used in a ciphertext formed in the forward direction. Only properties (2) and (3) could be affected by the adversarial ability to choose the T part of a ciphertext in encryption, but the fact that the s part is still random, and that $|s| \geq 2\kappa$, means that just like in IC, except for negligible probability each encryption outputs a ciphertext which is different from all previously used ones. Consequently, just like in IC, a HIC ciphertext commits the adversary to (at most) a *single* key used to create that ciphertext in a forward direction, the simulator can straight-line extract that key, and the decryption of this ciphertext under any other key samples random elements in the domain.

Further Applications: IC domain extension, LWE-based UC PAKE. The modified 2-Feistel construction can also be used as a *domain extender* for (randomized) IC on *bitstrings*. Given an RO hash onto $\{0,1\}^t$ and an IC on $\{0,1\}^{2\kappa}$, the $\mathsf{m2F}$ construction creates a HIC on $\{0,1\}^t$, for any $t = \mathsf{poly}(\kappa)$. The modified 2-Feistel is simpler than other IC domain extenders, e.g. [22, 42], and it has better exact security bounds, hence it is an attractive alternative in applications where HIC can securely substitute for IC on a large bitstring domain. For example, by our result (II) above, $\mathsf{m2F}$ on long bitstrings can be used to implement UC PAKE from any lattice-based IND-secure and anonymous KEM. This includes several post-quantum LWE-based KEM proposals in the NIST competition, including Saber [28], Kyber [17], McEliece [3], NTRU [43], Frodo [4], and possibly others.[4] Such UC PAKE construction would add only 3κ bits in bandwidth to the underlying KEM, and its computational overhead over the underlying KEM operations would be negligible, i.e. the LWE-based UC PAKE would have essentially exactly the

[3] This describes only the *adversarial* interface to the HIC functionality. Honest parties' interface is as in IC in both directions, except that it hides encryption randomness, i.e. encryption takes only input $M \in \mathbb{G}$ and decryption outputs only the $M \in \mathbb{G}$ part of the "extended" HIC plaintext $m \in \mathcal{D}$.

[4] Two recent papers [51, 59] investigate anonymity of several CCA-secure LWE-based KEMs achieved via variants of the Fujisaki-Okamoto transform [38] applied to the IND-secure versions of these KEM's. However, the underlying IND-secure KEM's are all anonymous, see e.g. [51, 59] and the references therein.

same cost as the LWE-based unauthenticated Key Exchange, i.e. an IND-secure KEM. We show a concrete construction of UC PAKE from Saber KEM in Appendix E.

Half-Ideal Cipher versus POPF. Our modified 2-Feistel construction and the UC HIC abstraction we use to capture its properties can be thought of as a "non-malleability upgrade" to the 2-Feistel, and to the game-based POPF abstraction used by McQuoid et al. [53] to capture its properties. One reason why the UC HIC notion is an improvement over the POPF notion is that a UC tool is easier to use in protocol applications than a game-based abstraction. More specifically, the danger of game-based properties is that they often fail to adequately capture non-malleability properties needed in protocol applications, e.g. in the EKE protocol, where the man-in-the-middle attacker can modify the ciphertexts exchanged between Alice and Bob.[5] Indeed, POPF properties seem not to capture ciphertext non-malleability. As defined in [53], POPF has two security properties, *honest simulation* and *uncontrollable outputs*. The first one says that if ciphertext c is output by a simulator on behalf of an honest party, then decrypting it under any key results in a random element in group $\mathbb{G}$, except for the (key,plaintext) pair, denoted (x^*, y^*) in [53], which was programmed into this ciphertext by the simulator. The second property says that any ciphertext c^* output by an adversary decrypts to random elements in group $\mathbb{G}$ for all keys except for key k^*, denoted x^* in [53], which was used by the adversary to create c^* in the forward direction, and which can be straight-line extracted by the simulator.[6] However, these properties do not say that the (key,plaintext) pairs behind the adversary's ciphertext c^* cannot bear any relation to the (key,plaintext) pairs behind the simulator's ciphertext c.

Note that non-malleability is necessary in a protocol application like EKE, and for that reason we think that it is unlikely that EKE can provably realize UC PAKE based on the POPF properties alone. Consider a cipher Enc on a multiplicative group s.t. there is an efficient algorithm A s.t. if $c = \mathsf{Enc}(k, M)$ and $c^* = A(c)$ then $M^* = \mathsf{Dec}(k, c^*)$ satisfies relation $M^* = M^2$ if $\mathsf{lsb}(k) = 0$, and $m^* = m^3$ if $\mathsf{lsb}(k) = 1$. If this cipher is used in EKE for password-encryption of DH-KE messages then the attacker would learn lsb of password pw used by Alice and Bob: If the attacker passes Alice's message $c_A = \mathsf{Enc}(pw, g^x)$ to Bob, but replaces Bob's message $c_B = \mathsf{Enc}(pw, g^y)$ by sending a modified message $c_B^* = A(c_B)$ to Alice, then $c_B^* = \mathsf{Enc}(pw, g^{y\cdot(2+b)})$ where $b = \mathsf{lsb}(pw)$, hence an attacker who sees Alice's output $k_A = g^{xy\cdot(2+b)}$ and Bob's output $k_B = g^{xy}$, can learn bit b by testing if $k_A = (k_B)^{(2+b)}$. More generally, any attack A which transforms ciphertext $c = \mathsf{Enc}(k, M)$ to ciphertext $c^* = \mathsf{Enc}(k^*, M^*)$ s.t. (k, M, k^*, M^*) are in some non-trivial relation, is a potential danger for EKE.

[5] A potential benefit of a game-based notion over a UC notion is that the former *could* be easier to state and use, but this does not seem to be the case for the POPF properties of [53], because they are quite involved and subtle.

[6] Technically [53] state this property as pseudorandomness of outputs of any weak-PRF on the decryptions of c^* for any $k \neq k^*$, and not the pseudorandomness of the decrypted plaintexts themselves.

We do not believe that 2-Feistel is subject to such attacks, but POPF properties defined in [53] do not seem to forbid them.

If one uses 2-Feistel directly rather than the POPF abstraction then it might still be possible to prove that EKE with 2-Feistel realizes UC PAKE. We note that 2-Feistel is subject to the following restricted form of "key-dependent malleability", which appears not to have been observed in [53] and which would have to be accounted for in such proof. Namely, consider an adversary who given ciphertext $c = (s, T)$ outputs ciphertext $c^* = (s^*, T^*)$ for any T^* and s^* s.t. $s^* \oplus \mathsf{H}'(pw^*, T^*) = s \oplus \mathsf{H}'(pw^*, T)$. Note that this adversary is not performing a decryption of c under pw^*, because it is not querying $\mathsf{H}(pw^*, r)$ for $r = s \oplus \mathsf{H}'(pw^*, T)$, but plaintexts $M^* = \mathsf{Dec}(pw, c^*)$ and $M = \mathsf{Dec}(pw, c)$ satisfy a non-trivial relation $M^*/M = T^*/T$ if $pw = pw^*$ and not otherwise. On the other hand, since this adversarial behavior seems to implement just a different form of an online attack using a unique password guess pw^*, it is still possible that EKE realizes UC PAKE even when password encryption is implemented as 2-Feistel. However, rather than considering such non-modular direct proofs for each application of IC on a group, in this paper we show that a small change in the 2-Feistel circuit implies realizing a HIC relaxation of the IC model, and this HIC relaxation is as easy to use as IC in the security proofs for protocols like EKE [10] or aPAKE's of Gu et al. [41, 37].

Finally, we note that an extension of the above attack shows that 2-Feistel itself, without our modification, cannot realize the HIC abstraction. Observe that if the adversary computes t hashes $Z_i = \mathsf{H}(pw, r_i)$ for some pw and $r_1, ..., r_t$ and then t hashes $k_j = \mathsf{H}'(pw, T_j)$ for some $T_1, ..., T_t$, then it can combine them to form t^2 valid (plaintext, ciphertext) pairs (M_{ij}, c_{ij}) under key pw where $M_{ij} = Z_i \cdot T_j$ and $c_{ij} = (r_i \oplus k_j, T_j)$. Note that the t^2 plaintexts are formed using just $2t$ group elements $(Z_1, T_1), ..., (Z_t, T_t)$, so they are correlated. For example, the value of quotient $M_{ij}/M_{i'j}$ is the same for every j. Creating such correlations on plaintexts is impossible in the UC HIC, hence 2-Feistel by itself, without our modification, does not realize it.

Roadmap. In Section 2, we recall the syntax and properties of Key Exchange (KE) and Key Encapsulation Mechanism (KEM). In Section 3 we define the UC notion of Half-Ideal Cipher (HIC). In Section 4 we present the modified 2-Feistel construction, and we show that it realizes UC HIC. In Section 5 we define two variants of the EKE protocol, denoted EKE and EKE-KEM, based on respectively KE and KEM, with password encryption implemented as HIC, and we show that both variants realizes UC PAKE.

Because of space constraints we defer some parts to the Appendix. Appendix A and Appendix B contain the details of game changes used in the security proofs of the above two results, i.e. that modified 2-Feistel realizes UC RIC and that EKE with encryption using HIC realizes UC PAKE. Appendix C contains the security proof of the EKE-KEM protocol. Appendix D shows that the KHAPE protocol of [41] realizes UC aPAKE with IC encryption replaced by HIC. In Appendix E we illustrate an instantiation of EKE-KEM protocol with Saber KEM [28], and we compare the resulting protocol to prior lattice-based PAKEs. Finally in Appendix F we include the standard UC PAKE and UC aPAKE functionalities for reference.

2 Preliminaries

We focus our treatment of the EKE protocol to instantiations that use Key Exchange (KE) with either a single simultaneous flow or 2 flows. Since a 2-flow KE is equivalent to a key encapsulation mechanism (KEM), we will use "KE" to refer to a single-round key exchange, and "KEM" to a KEM *and* to a two-flow key exchange implied by it.

2.1 Single-round Key Exchange (KE) Scheme

A (single-round) KE scheme is a pair of algorithms $\mathsf{KA} = (\mathsf{msg}, \mathsf{key})$, where:

- msg, on input a security parameter κ, generates message M and state x;
- key, on input state x and incoming message M', generates session key K.

The correctness requirement is that if two parties exchange honestly generated messages then they both output the same session key, i.e. if $(x_1, M_1) \leftarrow \mathsf{msg}(1^\kappa)$ and $(x_2, M_2) \leftarrow \mathsf{msg}(1^\kappa)$ then $\mathsf{key}(x_1, M_2) = \mathsf{key}(x_2, M_1)$. The KE security requirement is that a KE transcript hides the session key, but as noted by Bellare et al. [8], the EKE protocol requires an additional property of KE called a *random-message* property, namely that messages output by msg are indistinguishable from values sampled from a uniform distribution over some domain $\mathcal{M}$. (In the security analysis of EKE by [8], the EKE employs an Ideal Cipher on domain $\mathcal{M}$ for password-encryption of KE protocol messages.)

Definition 1. *KE scheme* $(\mathsf{msg}, \mathsf{key})$ *is* secure *if distributions* $\{(M_1, M_2, K)\}$ *and* $\{(M_1, M_2, K^*)\}$ *are computationally indistinguishable, where* $(x_1, M_1) \leftarrow \mathsf{msg}(1^\kappa)$*,* $(x_2, M_2) \leftarrow \mathsf{msg}(1^\kappa)$*,* $K \leftarrow \mathsf{key}(x_1, M_2)$*, and* $K^* \xleftarrow{r} \{0,1\}^\kappa$*.*

Definition 2. *KE scheme* $(\mathsf{msg}, \mathsf{key})$ *has the* random-message *property on domain* $\mathcal{M}$*, indexed by sec. par.* κ*, if the distribution* $\{M \mid (x, M) \leftarrow \mathsf{msg}(1^\kappa)\}$ *is computationally indistinguishable from uniform over set* $\mathcal{M}[\kappa]$*.*

2.2 Key Encapsulation Mechanism (KEM)

A KEM scheme is a tuple of efficient algorithms $\mathsf{KEM} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$, where:

- kg, on input secpar κ, generates public and private keys pk and sk;
- enc, on input a public key pk, generates ciphertext e and session key K;
- dec, on input a private key sk and a ciphertext e, outputs a session key K.

The correctness requirement is that if $(sk, pk) \leftarrow \mathsf{kg}(1^\kappa)$ and $(e, K) \leftarrow \mathsf{enc}(pk)$ then $\mathsf{dec}(sk, e) = K$. Note that KEM models any 2-flow key exchange scheme, where the public key pk is the initiator's message, and the ciphertext e is the responder's message. We require IND security of KEM, and two additional randomness/anonymity properties: First, public keys must be *uniform* in the sense that their distribution must be indistinguishable from a uniform distribution over some set $\mathcal{PK}$. Secondly, KEM must be anonymous [7], i.e. ciphertexts must be unlinkable to public keys. Note that these are slightly weaker properties than we asked of KA. Since a key exchange implied by KEM takes 2 flows, the EKE variant using KEM, see Figure 10 in Section 5.1, can use the (randomized) ideal cipher only for the first flow, i.e. the public key, while the second flow, i.e. the KEM ciphertext, can be sent as is, as long as the responder attaches to it a key confirmation message. Consequently, the second message must be unlinkable to the first, but it does not have to be indistinguishable from a random element in a domain of an ideal cipher.

Definition 3. *KEM scheme is* IND secure *if distributions* $\{(pk, e, K)\}$ *and* $\{(pk, e, K^*)\}$ *are computationally indistinguishable, where* $(sk, pk) \xleftarrow{r} \mathsf{kg}(1^\kappa)$, $(e, K) \xleftarrow{r} \mathsf{enc}(pk)$ *and* $K^* \xleftarrow{r} \{0,1\}^\kappa$.

Definition 4. *KEM scheme has* uniform public keys *for domain* $\mathcal{PK}$*, indexed by the security parameter* κ*, if the distribution* $\{pk \mid (sk, pk) \xleftarrow{r} \mathsf{kg}(1^\kappa)\}$ *is computationally indistinguishable from uniform over set* $\mathcal{PK}[\kappa]$

Definition 5. *KEM scheme is* anonymous *if distributions* $\{(pk_0, pk_1, e_0)\}$ *and* $\{(pk_0, pk_1, e_1)\}$ *are computationally indistinguishable, where* $(sk_0, pk_0) \xleftarrow{r} \mathsf{kg}(1^\kappa)$, $(sk_1, pk_1) \xleftarrow{r} \mathsf{kg}(1^\kappa)$, $(e_0, K_0) \xleftarrow{r} \mathsf{enc}(pk_0)$, *and* $(e_1, K_1) \xleftarrow{r} \mathsf{enc}(pk_1)$.

Note that the last two properties are trivially achieved by the Diffie-Hellman KEM, where both the public keys and ciphertexts are random group elements. However, both properties are also achieved by several lattice-based KEM's, as discussed in Section 1.

3 Universally Composable Half-Ideal Cipher

We define a new functionality $\mathcal{F}_{\mathsf{HIC}}$ in the UC framework ([19]), called a *(Randomized) Half-Ideal Cipher* (HIC), where the 'half' in the name refers to the fact that only half of the ciphertext is random to the adversary during encryption, as we explain below.

UC HIC is a weakening of the UC Ideal Cipher notion. Intuitively, we allow adversaries to predict or control part of the output of the cipher while the remainder is indistinguishable from random just as in the case of IC. Formally, we can interpret this as allowing the adversary to embed some tuples in the table that the functionality uses - but in a very controlled manner. We define the UC notion of Half-Ideal Cipher via functionality $\mathcal{F}_{\mathsf{HIC}}$ defined in Figure 2.[7]

Notes on $\mathcal{F}_{\mathsf{HIC}}$ interfaces. A half-ideal cipher functionality $\mathcal{F}_{\mathsf{HIC}}$ is parametrized by the (randomized) cipher domain $\mathcal{D} = \mathcal{R} \times \mathcal{G}$, where the first

[7] In Figure 2 we use pw to denote keys used in the HIC cipher because we use variables k and K for other keys in the later sections. Moreover, in PAKE and aPAKE applications the role of a HIC key is played by a password.

Notation: Functionality $\mathcal{F}_{\mathsf{HIC}}$ is parametrized by domain $\mathcal{D} = \mathcal{R} \times \mathcal{G}$, and it is indexed by a session identifier sid which is a global constant, hence we omit it from notation. We denote HIC keys as passwords pw to conform to the usage of $\mathcal{F}_{\mathsf{HIC}}$ in PAKE and aPAKE applications, but keys pw are arbitrary bitstrings.

Initialization: For all $pw \in \{0,1\}^*$, initialize THIC_{pw} as an empty table.

Interfaces for Honest Parties P*:*

on query (Enc, pw, M) from party P, for $M \in \mathcal{G}$:

$r \xleftarrow{r} \mathcal{R}$
if $\exists c$ s.t. $((r, M), c) \in \mathsf{THIC}_{pw}$ then return c to P, else do:
  $c \xleftarrow{r} \{\hat{c} \in \mathcal{D} : \nexists m \text{ s.t. } (m, \hat{c}) \in \mathsf{THIC}_{pw}\}$
  add $((r, M), c)$ to THIC_{pw} and return c to P

on query (Dec, pw, c) from party P, for $c \in \mathcal{D}$:

query $(r, M) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(pw, c)$ and return M to P

Interfaces for Adversary $\mathcal{A}$ *(or corrupt parties):*

on query $(\mathsf{AdvEnc}, pw, (r, M), T)$ from adversary $\mathcal{A}$, for $(r, M) \in \mathcal{D}$ and $T \in \mathcal{G}$:

if $\exists c$ s.t. $((r, M), c) \in \mathsf{THIC}_{pw}$ then return c to $\mathcal{A}$, else do:
  $s \xleftarrow{r} \{\hat{s} \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (\hat{s}, T)) \in \mathsf{THIC}_{pw}\}$
  set $c \leftarrow (s, T)$, add $((r, M), c)$ to THIC_{pw}, and return c to $\mathcal{A}$

on query (AdvDec, pw, c) from adversary $\mathcal{A}$, for $c \in \mathcal{D}$:

if $\exists m$ s.t. $(m, c) \in \mathsf{THIC}_{pw}$ then return m to $\mathcal{A}$, else do:
  $m \xleftarrow{r} \{\hat{m} \in \mathcal{D} : \nexists \hat{c} \text{ s.t. } (\hat{m}, \hat{c}) \in \mathsf{THIC}_{pw}\}$
  add (m, c) to THIC_{pw} and return m to $\mathcal{A}$

Fig. 2: Ideal functionality $\mathcal{F}_{\mathsf{HIC}}$ for *(Randomized) Half-Ideal Cipher* on $\mathcal{D} = \mathcal{R} \times \mathcal{G}$

component is the randomness and the second is the plaintext. Figure 2 separates between $\mathcal{F}_{\mathsf{HIC}}$ interfaces Enc and Dec which are used by honest parties, and the adversarial interfaces AdvEnc and AdvDec. Interfaces Enc and Dec model honest-party's usage of HIC, i.e. a real-world implementation of HIC will consists of two algorithms, Enc and Dec, where Enc on input key pw and plaintext $M \in \mathcal{G}$ outputs a ciphertext $c \in \mathcal{D}$ and Dec on input key pw and ciphertext $c \in \mathcal{D}$ outputs a plaintext $M \in \mathcal{G}$. Our target realization of these procedures is a *randomized cipher*, i.e. a family of functions Π_{pw} s.t. for each $pw \in \{0,1\}^*$, Π_{pw} is a permutation on $\mathcal{D}$, and both Π_{pw} and Π_{pw}^{-1} are efficiently evaluable given pw. Given cipher Π, algorithm $\mathsf{Enc}(pw, M)$ picks $r \xleftarrow{r} \mathcal{R}$ and outputs $c \leftarrow \Pi_{pw}(m)$ for $m = (r, M)$, while $\mathsf{Dec}(pw, c)$ computes $m \leftarrow \Pi_{pw}^{-1}(c)$ and output M for $(r, M) = m$.

Functionality walk-through. Functionality $\mathcal{F}_{\mathsf{HIC}}$ reflects honest user's interfaces to randomized encryption: When an honest party P encrypts a message it specifies only $M \in \mathcal{G}$ and delegates the choice of randomness $r \xleftarrow{r} \mathcal{R}$

to the functionality. Similarly, when an honest party decrypts a ciphertext, the functionality discards the randomness r and reveals only M to the application. This implies that honest parties must use fresh randomness at each encryption and must discard it (or at least not use it) at decryption. By contrast, an adversary $\mathcal{A}$ has stronger interfaces than honest parties (for notational simplicity we assume corrupt parties interact to $\mathcal{F}_{\mathsf{HIC}}$ via $\mathcal{A}$), namely: (1) When $\mathcal{A}$ encrypts it can choose randomness r at will; (2) When $\mathcal{A}$ decrypts it learns the randomness r and does not have to discard it; (3) $\mathcal{A}$ can manipulate the (plaintext, ciphertext) table of each permutation Π_{pw} in the following way: If we denote ciphertexts as $c = (s, T) \in \mathcal{R} \times \mathcal{G}$, the adversary has no control of the s component of the ciphertext at encryption, i.e. it is random in $\mathcal{R}$ (up to the fact that the map has to remain a permutation), but the adversary can freely choose the T component. Items (1) and (2) are consequences of the fact that HIC is a *randomized* cipher, but item (3) is what makes this cipher *Half-Ideal*, because the adversary can control part of the value $c = \mathsf{Enc}(pw, m)$ during encryption, namely its $\mathcal{G}$ component.

The above relaxations of Ideal Cipher (IC) properties are imposed by the modified 2-Feistel construction, which in Section 4 we show realizes this model. However, this relaxation is harmless for many IC applications the following reason: In a typical IC application the benefit of ciphertext randomness is that it (1) hides the plaintext, and (2) it prevents the adversary from creating the same ciphertext as an encryption of two different plaintexts under two different keys. For both purposes randomness in the $s \in \mathcal{R}$ component of the ciphertext suffices as long as $\mathcal{R}$ is large enough to prevent ever encountering collisions.

The adversarial interfaces AdvEnc and AdvDec of $\mathcal{F}_{\mathsf{HIC}}$ reflect the above, and give more powers than the honest party's interfaces Enc and Dec. In encryption query AdvEnc, the adversary is allowed to pick its own randomness r *and* the $T \in \mathcal{G}$ part of the resulting ciphertext, while its s part is chosen at random in $\mathcal{R}$. In decryption AdvDec, the adversary can decrypt any ciphertext $c = (s, T)$ and it learns the full plaintext $m = (r, M)$, but $\mathcal{F}_{\mathsf{HIC}}$ chooses the whole plaintext m at random. (This is another motivation for the monicker 'half-ideal': $\mathcal{F}_{\mathsf{HIC}}$ lets the adversary have some control over ciphertexts in encryption but it does not let the adversary have any control over plaintexts in decryption.)

Our goal when designing $\mathcal{F}_{\mathsf{HIC}}$ was to keep all IC properties which are useful in applications while allowing for efficient concrete instantiation of $\mathcal{F}_{\mathsf{HIC}}$ for a group domain $\mathcal{G}$. Most importantly, ciphertext collisions in encryption can occur only with negligible probability, which is crucial in our HIC applications: An adversarial ciphertext c commits the adversary to a single key pw on which the adversary could have computed c as an encryption of some message of its choice. Secondly, just as with an ideal cipher, the adversary cannot learn any information on encrypted plaintexts except via decryption with correct decryption key.

4 Half-Ideal Cipher Construction: Modified 2-Feistel

We modify the two-round Feistel construction of the Programmable Once Public Functions (POPF) of McQuoid et al. [53] by replacing the xor operation in the second round by an application of an ideal block cipher BC on bitstrings, with keys and plaintext block both of size 2κ where κ is the security parameter. We call this construction a *modified 2-Feistel*, denoted $\mathsf{m2F}$. This construction takes (1) an ideal cipher BC on bitstrings, i.e. an ideal cipher whose domain is $\{0,1\}^n$ and key space is $\{0,1\}^\mu$, (2) a random oracle hash H' with range $\{0,1\}^\mu$, and (3) a random oracle hash H whose range is an arbitrary *group* $\mathbb{G}$, and creates a *(Randomized) Half-Ideal Cipher* (HIC) over domain $\mathcal{D} = \mathcal{R} \times \mathbb{G}$ where $\mathcal{R} = \{0,1\}^n$. In essence, we combine a random oracle hash onto a group and a bitwise ideal cipher to create a half-ideal cipher over a group. The exact security analysis of the $\mathsf{m2F}$ construction shows that μ and n can both be set to 2κ for this construction to realize UC HIC.

For each key pw, function $\mathsf{m2F}_{pw}$ is pictorially shown in Figure 1. Here we define it by the algorithms which compute $\mathsf{m2F}_{pw}$ and $\mathsf{m2F}^{-1}_{pw}$. (Throughout the paper we denote group $\mathbb{G}$ operation as a multiplication, but this is purely a notational choice, and the construction applies to additive groups as well.)

$$\mathsf{m2F}_{pw} \;:\; \{0,1\}^n \times \mathbb{G} \;\;\rightarrow\;\; \{0,1\}^n \times \mathbb{G} \tag{1}$$

where:

$\mathsf{m2F}_{pw}(r, M)$:

1. $T \leftarrow M/\mathsf{H}(pw, r)$
2. $k \leftarrow \mathsf{H}'(pw, T)$
3. $s \leftarrow \mathsf{BC.Enc}(k, r)$
4. Output (s, T)

$\mathsf{m2F}^{-1}_{pw}(s, T)$:

1. $k \leftarrow \mathsf{H}'(pw, T)$
2. $r \leftarrow \mathsf{BC.Dec}(k, s)$
3. $M \leftarrow \mathsf{H}(pw, r) \cdot T$
4. Output (r, M)

The following theorem captures the security of the $\mathsf{m2F}$ construction:

Theorem 1. *Construction* $\mathsf{m2F}$ *realizes functionality* $\mathcal{F}_{\mathsf{HIC}}$ *in the domain* $\mathcal{R} \times \mathbb{G}$ *for* $\mathcal{R} = \{0,1\}^n$ *if* $\mathsf{H} : \{0,1\}^* \times \{0,1\}^n \rightarrow \mathbb{G}$, $\mathsf{H}' : \{0,1\}^* \times \mathbb{G} \rightarrow \{0,1\}^\mu$ *are random oracles,* $\mathsf{BC} : \{0,1\}^\mu \times \{0,1\}^n \rightarrow \{0,1\}^n$ *is an ideal cipher, and* μ *and* n *are both* $\Omega(\kappa)$.

Proof. The proof for Theorem 1 must exhibit a simulator algorithm SIM, which plays a role of an ideal-world adversary interacting with functionality $\mathcal{F}_{\mathsf{HIC}}$, and then show that no efficient environment $\mathcal{Z}$ can distinguish, except for negligible probability, between (1) a *real-world game*, i.e. an interaction with (1a) honest parties who execute $\mathcal{Z}$'s encryption and decryption queries using Enc and Dec implemented with circuit $\mathsf{m2F}$ (see Section 3), and (1b) RO/IC oracles H, H', BC, BC^{-1}, and (2) an *ideal-world game*, i.e. an interaction with (2a) parties P who execute $\mathcal{Z}$'s encryption and decryption using interfaces $\mathsf{Enc}, \mathsf{Dec}$ of $\mathcal{F}_{\mathsf{HIC}}$, and (2b) simulator SIM, who services $\mathcal{Z}$'s calls to H, H', BC, BC^{-1} using interfaces AdvEnc and AdvDec of $\mathcal{F}_{\mathsf{HIC}}$.

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on adversary's query $\mathsf{H}(pw, r)$

if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$:
  $h \xleftarrow{r} \mathbb{G}$
  add (pw, r, h) to TH
return h

on adversary's query $\mathsf{H}'(pw, T)$

if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
  $k \xleftarrow{r} \{0,1\}^\mu$
  if $\exists(\hat{pw}, \hat{T})$ s.t. $(\hat{pw}, \hat{T}, k) \in \mathsf{TH}'$ then abort (col.abort)
  if $\exists(\hat{r}, \hat{s})$ s.t. $(k, \hat{r}, \hat{s}) \in \mathsf{TBC}$ then abort (bckey.abort)
  add (pw, T, k) to TH'
return k

on adversary's query $\mathsf{BC.Enc}(k, r)$

if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw, T)^a$:
    $M \leftarrow \mathsf{H}(pw, r) \cdot T$
    $(s, \hat{T}) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$
    if $\hat{T} \neq T$ then abort (advenc.abort)
  else:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k, \hat{r}, s) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return s

[a] If it exists, we denote by $\mathsf{TH}'(pw, T)$ the (unique) k s.t. $(pw, T, k) \in \mathsf{TH}'$

on adversary's query $\mathsf{BC.Dec}(k, s)$

if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw, T)$:
    $(r, M) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(pw, (s, T))$
    if $\exists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}$ then abort (advdec.abort)
    if $\exists h$ s.t. $(pw, r, h) \in \mathsf{TH}$ then abort (rcol.abort)
    add $(pw, r, M \cdot T^{-1})$ to TH
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return r

Fig. 3: Simulator SIM for the proof of Theorem 1

We start by describing the simulator algorithm SIM, shown in Figure 3. Note that SIM interacts with an adversarial environment algorithm $\mathcal{Z}$ by servicing $\mathcal{Z}$'s queries to the RO and IC oracles $\mathsf{H}, \mathsf{H}', \mathsf{BC}, \mathsf{BC}^{-1}$. Intuitively, SIM populates input, output tables for these functions, $\mathsf{TH}, \mathsf{TH}'$ and TBC, in the same way as these idealized oracles would, except when SIM detects a possible encryption or decryption computation of the modified 2-Feistel circuit. In case SIM decides that these queries form either computation of $\mathsf{m2F}$ or $\mathsf{m2F}^{-1}$ on new input, SIM detects that input, invokes the adversarial interfaces AdvEnc or AdvDec of $\mathcal{F}_{\mathsf{HIC}}$ to find the corresponding output, and it embeds proper values into these tables to emulate the circuit leading to the computation of this output. The detection of $\mathsf{m2F}$ and $\mathsf{m2F}^{-1}$ evaluation is relatively straightforward: First, SIM treats every $\mathsf{BC.Dec}$ query (k, s) as a possible $\mathsf{m2F}^{-1}$ evaluation on key pw and ciphertext $c = (s, T)$ for T s.t. $k = \mathsf{H}'(pw, T)$. If it is, SIM queries $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$ on (pw, c) to get $m = (r, M)$. Since this is a random sample from the HIC domain, with overwhelming probability H was not queried on r so SIM can set $\mathsf{H}(pw, r)$ to M/T. Second, SIM treats every $\mathsf{BC.Enc}$ query (k, r) as possible $\mathsf{m2F}$ evaluation on (r, M) s.t. $M = \mathsf{H}(pw, r) \cdot T$ for T s.t. $k = \mathsf{H}'(pw, T)$. However, here is where the difference between IC and HIC shows up: The $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ query fixes the encryption of $m = (r, M)$ to $c = (s, T)$, and whereas s can be random (and SIM can set $\mathsf{BC.Enc}(k, r) := s$ for any $c = (s, T)$ returned by $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ as encryption of m under key pw), value T was fixed by H' output k (except for the negligible probability of finding collisions in H'). This is why our $\mathcal{F}_{\mathsf{HIC}}$ model must allow the simulator, i.e. the ideal-world adversary, to fix the T part of the ciphertext in the adversarial encryption query AdvEnc.

Proof Overview. The proof must show that for any environment $\mathcal{Z}$, its view of the real-world game defined by algorithms $\mathsf{Enc}, \mathsf{Dec}$ which use the randomized cipher $\mathsf{m2F}$, and the ideal-world game defined by functionality $\mathcal{F}_{\mathsf{HIC}}$ and simulator SIM of Figure 3. The proof starts from the ideal-world view, which we denote as Game 0, and via a sequence of games, each of which we show is indistinguishable from the next, it reaches the real-world view, which we denote as Game 9. For space-constraint reasons we include the details of the game changes and reductions to Appendix A, but we show the code of all successive games in Figures 4, 5, and 6. Figure 4 describes the ideal-world Game 0 and its mild modification Game 1. All these games, starting from Game 0 in Figure 4, interact with an adversarial environment $\mathcal{Z}$, and each game provides two types of interfaces corresponding two types of $\mathcal{Z}$'s queries: (a) the honest party's interfaces $\mathsf{Enc}, \mathsf{Dec}$, which $\mathcal{Z}$ can query via any honest party, and (b) RO/IC oracles $\mathsf{H}, \mathsf{H}', \mathsf{BC}, \mathsf{BC}^{-1}$, which $\mathcal{Z}$ can query via its "real-world adversary" interface. Figure 4 defines two sub-procedures, $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ and $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$, whose code matches exactly the corresponding interfaces of $\mathcal{F}_{\mathsf{HIC}}$. These subprocedures are used internally by Game 0: They are invoked by the code that services $\mathcal{Z}$'s queries $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$, because Game 0 follows SIM's code on these queries, and AdvDec is also invoked by Dec, because this is how $\mathcal{F}_{\mathsf{HIC}}$ implements Dec.

Figures 5 and 6 describe the modifications created by all subsequent games, except for the last one, the real-world game denoted Game 9, which is very similar to Game 8, which is the last game shown in Figure 6. By the arguments for indistinguishability of successive games shown in Appendix A, the total distinguishing advantage of environment $\mathcal{Z}$ between the real-world and the ideal-world interaction is upper-bounded by the following expression, which sums up the bounds given by equations (3) to (7) in Appendix A:

$$|P_0 - P_9| \leq q^2 \left(\frac{10}{2^n} + \frac{4}{2^n \cdot |\mathbb{G}|} + \frac{6}{2^\mu} \right) \leq q^2 \left(\frac{14}{2^n} + \frac{6}{2^\mu} \right)$$

Since this quantity is negligible, this implies Theorem 1 $\square$

Notes on Exact Security. By the above equation, the distinguishability advantage implies by our proof can be upper-bounded as $O(q^2/2^n) + O(q^2/2^\mu)$. We assert that both of these factors are unavoidable for our $\mathsf{m2F}$ construction. First, while in the $\mathcal{F}_{\mathsf{HIC}}$ functionality we allow the T component of two AdvEnc adversarial calls to be completely independent, this is not the case in our modified two-round Feistel encryption: reuse of a (pw, r) pair implies relations between the T component of different encryption calls that are not seen in

<u>Initialization</u>

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

For each $pw \in \{0,1\}^*$, initialize empty sets THIC_{pw} and usedR_{pw}.

<u>define $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw,(r,M),T)$:</u>

```
if ∄c s.t. ((r,M),c) ∈ THIC_pw:
    s ←r {ŝ ∈ {0,1}^n : (*,(ŝ,T)) ∉ THIC_pw}
    c ← (s,T)
    add ((r,M),c) to THIC_pw
return c
```

<u>define $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(pw,(s,T))$:</u>

```
if ∄(r,M) s.t. ((r,M),(s,T)) ∈ THIC_pw:
    (r,M) ←r D
    if ∃ĉ s.t. ((r,M),ĉ) ∈ THIC_pw then abort
    abort if r ∈ usedR_pw else add r with tag m2F
    add ((r,M),(s,T)) to THIC_pw
return M
```

<u>on query $\mathsf{Enc}(pw,M)$:</u>

```
r ←r {0,1}^n
abort if r ∈ usedR_pw, else add r with tag m2F
if ∄c s.t. ((r,M),c) ∈ THIC_pw:
    c ←r {ĉ : ∄m̂ s.t. (m̂,ĉ) ∈ THIC_pw}
    add ((r,M),c) to THIC_pw
return c
```

<u>on query $\mathsf{Dec}(pw,c)$:</u>

```
(r,M) ← F_HIC.AdvDec(pw,c)
return M
```

<u>on query $\mathsf{H}(pw,r)$</u>

```
abort if r ∈ usedR_pw tagged m2F, else add r
if ∄h s.t. (pw,r,h) ∈ TH:
    h ←r G
    add (pw,r,h) to TH
return h
```

<u>on query $\mathsf{H}'(pw,T)$</u>

```
if ∄k s.t. (pw,T,k) ∈ TH':
    k ←r {0,1}^μ
    if ∃ (pŵ,T̂) s.t. (pŵ,T̂,k) ∈ TH' then abort (col.abort)
    if ∃ (r̂,ŝ) s.t. (k,r̂,ŝ) ∈ TBC then abort (bckey.abort)
    add (pw,T,k) to TH'
return k
```

<u>on query $\mathsf{BC.Enc}(k,r)$</u>

```
if k = TH'(pw,T):
    if r ∈ usedR_pw is tagged m2F then abort
    else add r to usedR_pw
if ∄s s.t. (k,r,s) ∈ TBC:
    if k = TH'(pw,T):
        M ← H(pw,r) · T
        (s,T̂) ← F_HIC.AdvEnc(pw,(r,M),T)
        if T̂ ≠ T then abort          (advenc.abort)
    else:
        s ←r {s ∈ {0,1}^n : ∄r̂ s.t. (k,r̂,s) ∈ TBC}
    add (k,r,s) to TBC
return s
```

<u>on query $\mathsf{BC.Dec}(k,s)$</u>

```
if ∄r s.t. (k,r,s) ∈ TBC:
    if k = TH'(pw,T):
        (r,M) ← F_HIC.AdvDec(pw,(s,T))
        if ∃ŝ s.t. (k,r,ŝ) ∈ TBC then abort (advdec.abort)
        if ∃ h s.t. (pw,r,h) ∈ TH then abort (rcol.abort)
        add (pw,r,M · T^{-1}) to TH
    else:
        r ←r {r ∈ {0,1}^n : ∄ŝ s.t. (k,r,ŝ) ∈ TBC}
    add (k,r,s) to TBC
if k = TH'(pw,T):
    remove tag m2F from record r ∈ usedR_pw
return r
```

Fig. 4: The ideal-world Game 0, and its modification Game 1 (text in gray)

Game 2: replacing decryption by circuit

on query $\mathsf{m2F.Dec}(pw,(s,T))$:

$k \leftarrow \mathsf{H}'(pw,T)$
$r \leftarrow \mathsf{BC.Dec}(k,s)$
$M \leftarrow \mathsf{H}(pw,r)\cdot T$
if $\mathsf{m2F.Dec}$ query was fresh, add tag $\mathsf{m2F}$ to $r \in \mathsf{usedR}_{pw}$
return M

Game 3: Enc calls AdvDec

on query $\mathsf{m2F.Enc}(pw,M)$:

$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it with tag $\mathsf{m2F}$
if $\nexists c$ s.t. $((r,M),c) \in \mathsf{THIC}_{pw}$:
  $T \xleftarrow{r} \mathbb{G}$
  $c \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw,(r,M),T)$
return c

Game 4: replacing encryption by circuit

on query $\mathsf{m2F.Enc}(pw,M)$:

$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort
$T \leftarrow M/\mathsf{H}(pw,r)$
$k \leftarrow \mathsf{H}'(pw,T)$
$s \leftarrow \mathsf{BC.Enc}(k,r)$
assign tag $\mathsf{m2F}$ to r in the set usedR_{pw}
return (s,T)

Game 5: H is a random oracle

$\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$ not used anymore

on query $\mathsf{BC.Dec}(k,s)$:

if $\nexists\, r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $r \xleftarrow{r} \{0,1\}^n$
    if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
    $h \leftarrow \mathsf{H}(pw,r)$
    $M \leftarrow h\cdot T$
    if $\exists \hat{c}$ s.t. $((r,M),\hat{c}) \in \mathsf{THIC}_{pw}$ then abort
    add $((r,M),(s,T))$ to THIC_{pw}
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}$
  add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

Game 6: simplifying parameters

define $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw,r,T)$:

if $\nexists s$ s.t. $(r,(s,T)) \in \mathsf{THIC}_{pw}$:
  $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(\hat{r},(\hat{s},T)) \in \mathsf{THIC}_{pw}\}$
  add $(r,(s,T))$ to THIC_{pw}
return s

on query $\mathsf{BC.Dec}(k,s)$:

if $\nexists r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $r \xleftarrow{r} \{0,1\}^n$
    if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
    query $\mathsf{H}(pw,r)$ and discard the output
    if $\exists \hat{c}$ s.t. $(r,\hat{c}) \in \mathsf{THIC}_{pw}$ then abort
    add $(r,(s,T))$ to THIC_{pw}
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}$
  add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

on query $\mathsf{BC.Enc}(k,r)$:

if $k = \mathsf{TH}'(pw,T)$:
  if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$ then abort
    else add r to usedR_{pw}
if $\nexists s$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    query $\mathsf{H}(pw,r)$ and discard the output
    $s \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw,r,T)$
  else:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k,\hat{r},s) \in \mathsf{TBC}\}$
  add (k,r,s) to TBC
return s

Game 7: using k

Initialization: $\forall\, k$ initialize empty THIC_k

define $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(k,r)$:

if $\nexists s$ s.t. $(r,s) \in \mathsf{THIC}_k$:
  $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(\hat{r},\hat{s}) \in \mathsf{THIC}_k\}$
  add (r,s) to THIC_k
return s

on query $\mathsf{BC.Dec}(k,s)$:

if $\nexists r$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $r \xleftarrow{r} \{0,1\}^n$
    if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
    if $\exists \hat{s}$ s.t. $(r,\hat{s}) \in \mathsf{THIC}_k$ then abort
    add (r,s) to THIC_k
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k,r,\hat{s}) \in \mathsf{TBC}$
  add (k,r,s) to TBC
remove tag $\mathsf{m2F}$ from $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw,T)$
return r

on query $\mathsf{BC.Enc}(k,r)$:

if $k = \mathsf{TH}'(pw,T)$:
  if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$ then abort
    else add r to usedR_{pw}
if $\nexists s$ s.t. $(k,r,s) \in \mathsf{TBC}$:
  if $k = \mathsf{TH}'(pw,T)$:
    $s \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(k,r)$
  else:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k,\hat{r},s) \in \mathsf{TBC}\}$
  add (k,r,s) to TBC
return s

Fig. 5: Game-changes (part 1) in the proof of Theorem 1

Game 8: THIC is redundant

Initialization: Drop THIC usage.
$\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ not used anymore

on query $\mathsf{BC.Enc}(k, r)$:
if $k = \mathsf{TH}'(pw, T)$:
    if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$, abort, else add $r \in \mathsf{usedR}_{pw}$
if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
    $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k, \hat{r}, s) \in \mathsf{TBC}\}$
    add (k, r, s) to TBC
return s

on query $\mathsf{BC.Dec}(k, s)$:
if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
    if $\exists (pw, T)$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
        $r \xleftarrow{r} \{0,1\}^n$
        if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
    else:
        $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k, r, \hat{s}) \in \mathsf{TBC}\}$
    add (k, r, s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw, T)$
return r

Fig. 6: Game-changes (part 2) in the proof of Theorem 1

$\mathcal{F}_{\mathsf{HIC}}$. Hence we must avoid r collisions in Enc calls, irrespective of how our proof is structured, and asymptotically this gives a $q^2/2^n$ factor in the distinguishing advantage.

Secondly, we need to avoid H' collisions. Indeed, if $\mathsf{H}'(pw, T) = \mathsf{H}'(\hat{pw}, \hat{T})$ then $\mathsf{m2F}$'s decryptions using (pw, T) and $(\hat{pw}, \hat{T})$ create the same $s \mapsto r$ map, which would be in stark contrast to our functionality's ideal-cipher like decryption behavior. We conclude that the $q^2/2^\mu$ term also can't be avoided. Notice that these two terms dominate the probability of the environment distinguishing $\mathsf{m2F}$ from our functionality $\mathcal{F}_{\mathsf{HIC}}$. In particular, they do not involve $|\mathbb{G}|$, i.e., the size of the message space of our $\mathcal{F}_{\mathsf{HIC}}$.

5 Encrypted Key Exchange with Half-Ideal Cipher

We show that the Encrypted Key Exchange (EKE) protocol of Bellovin and Meritt [10] is a universally composable PAKE if the password encryption is implemented with a (Randomized) Half-Ideal Cipher on the domain of messages output by the key exchange scheme, provided that the key exchange scheme has the random-message property (see Section 2). As discussed in the introduction, the same statement was argued by Rosulek et al. [53] with regards to password-encryption implemented using a Programmable Once Public Function (POPF) notion defined therein, which can also be thought of as a weak form of ideal cipher. However, since as we explain in the introduction, the POPF notion is unlikely to suffice in an EKE application, so we need to verify that the notion of UC (Randomized) Half-Ideal Cipher *does* suffice in such application.

In Figure 7 we show the Encrypted Key Exchange protocol EKE, specialized to use a Half-Ideal Cipher for the password-encryption of the message flows of the underlying Key Agreement scheme KA. In Figure 7 we assume that KA is a *single-round* scheme. In Section 5.1 we extend this to the case of two-flow KA, i.e. to EKE protocol instantiated with a KEM scheme. We note that these two treatments are incomparable because in the case of single-flow KA we start from a more restricted KA scheme and we argue security of a single-flow version of

EKE, whereas in the case of two-flow KA, i.e. if $\mathsf{KA} = \mathsf{KEM}$, we start from a more general KA scheme but we argue security of a two-flow version of EKE.

The EKE instantiation shown in Figure 7 assumes that the Half-Ideal Cipher HIC works on domain $\mathcal{D} = \mathcal{R} \times \mathcal{M}$ where $\mathcal{M}$ is the message domain of the scheme KA. The "randomness" set $\mathcal{R}$ is arbitrary, but its size influences the security bound we show for such EKE instantiations. In particular we require that $\log(|\mathcal{R}|) \geq 2\kappa$. If HIC is instantiated with the modified 2-Feistel construction $\mathsf{m2F}$ of Section 4, one can set $\mathcal{R} = \{0,1\}^{2\kappa}$, and this instantiation of EKE will send messages whose sizes match those of the underlying KA scheme extended by 2κ bits of randomness due to the Half-Ideal Cipher encryption.

In Figure 7 for presentation clarity we assume that party identifiers $\mathsf{P}_0, \mathsf{P}_1$ are lexicographically ordered. The full protocol will use two helper functions order and bit, defined as $\mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}) = (\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ and $\mathsf{bit}(\mathsf{P}, \mathsf{CP}) = 0$ if $\mathsf{P} <_{lex} \mathsf{CP}$, and $\mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP}) = (\mathsf{sid}, \mathsf{CP}, \mathsf{P})$ and $\mathsf{bit}(\mathsf{P}, \mathsf{CP}) = 1$ if $\mathsf{CP} <_{lex} \mathsf{P}$ [8]. Party P on input $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ will then set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ and $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$ and it will use $\mathsf{HIC.Enc}$ on key $\hat{pw}_b = (\mathsf{fullsid}, b, pw)$ to encrypt its outgoing message, and it will use $\mathsf{HIC.Dec}$ on key $\hat{pw}_{\neg b} = (\mathsf{fullsid}, \neg b, pw)$ to decrypt its incoming message.

- Single-round Key Exchange $\mathsf{KA} = (\mathsf{msg}, \mathsf{key})$ with message space $\mathcal{M}$
- Half-Ideal Cipher HIC on domain $\mathcal{R} \times \mathcal{M}$ for $\mathcal{R} = \{0,1\}^{\Omega(\kappa)}$

$\underline{\mathsf{P}_0 \text{ on } \mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1, pw_0)}$		$\underline{\mathsf{P}_1 \text{ on } \mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_1, \mathsf{P}_0, pw_1)}$
	(Assume $\mathsf{P}_0 <_{lex} \mathsf{P}_1$ and let $\mathsf{fullsid} = (\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1)$)	
$(x_0, M_0) \xleftarrow{r} \mathsf{KA.msg}$		$(x_1, M_1) \xleftarrow{r} \mathsf{KA.msg}$
$c_0 \leftarrow \mathsf{HIC.Enc}((\mathsf{fullsid}, 0, pw_0), M_0)$	$\xrightarrow{c_0}$ $\xleftarrow{c_1}$	$c_1 \leftarrow \mathsf{HIC.Enc}((\mathsf{fullsid}, 1, pw_1), M_1)$
$\hat{M}_1 \leftarrow \mathsf{HIC.Dec}((\mathsf{fullsid}, 1, pw_0), c_1)$		$\hat{M}_0 \leftarrow \mathsf{HIC.Dec}((\mathsf{fullsid}, 0, pw_1), c_0)$
output $K_0 \leftarrow \mathsf{KA.key}(x_0, \hat{M}_1)$		output $K_1 \leftarrow \mathsf{KA.key}(x_1, \hat{M}_0)$

Fig. 7: EKE: Encrypted Key Exchange with Half-Ideal Cipher

In Theorem 2 below we show that protocol EKE realizes the (multi-session version of) the PAKE functionality of Canetti et al. [20], denoted $\mathcal{F}_{\mathsf{pwKE}}$ (included in Figure 23 in Appendix F). The reason we target the multi-session version of PAKE functionality directly, rather than targeting its single-session version and then resorting to Canetti's composition theorem [19] to imply the security of an arbitrary (and concurrent) number of EKE instances, is that for the latter to work we would need the underlying UC HIC to be instantiated separately for each EKE session identifier sid. Our UC HIC notion of Section 3 is a "global" functionality, i.e. it does not natively support separate instances indexed by session identifiers. The modified 2-Feistel construction *could*

[8] We assume that no honest P ever executes $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot)$ for $\mathsf{CP} = \mathsf{P}$.

support such independent instances of HIC by prepending sid to the inputs of all its building block functions $\mathsf{H}, \mathsf{H}', \mathsf{BC}$, where in the last case value sid would have to be prepended to the key of the (ideal) block-cipher BC. However, this implies longer inputs for each of these blocks, which is especially problematic in case of the block cipher, so it is preferable not to rely on it and show security for a protocol variant where each EKE instance accesses a single HIC functionality, and hence can be implemented with the same instantiation of the modified 2-Feistel HIC construction.

Theorem 2. *If* KA *is a secure key-exchange scheme with the random-message property on domain* $\mathcal{M}$ *and* HIC *is a UC Half-Ideal Cipher over domain* $\mathcal{R} \times \mathcal{M}$*, then protocol* EKE*, Figure 7, realizes the UC PAKE functionality* $\mathcal{F}_{\mathsf{pwKE}}$*.*

Proof. Let $\mathcal{Z}$ be an arbitrary efficient environment. In the rest of the proof we will assume that the real-world adversary $\mathcal{A}$ is an interface of $\mathcal{Z}$. In Figure 8 we show the construction of a simulator algorithm SIM, which together with functionality $\mathcal{F}_{\mathsf{pwKE}}$ defines the ideal-world view of $\mathcal{Z}$. As is standard, the role of SIM is to emulate actions of honest parties executing protocol EKE given the information revealed by functionality $\mathcal{F}_{\mathsf{pwKE}}$, and to convert the actions of the real-world adversary into queries to $\mathcal{F}_{\mathsf{pwKE}}$. (In Figure 8 we use $\mathsf{P}^{\mathsf{sid}}$ to denote P's session indexed by sid which is emulated by SIM.) The proof then consists of a sequence of games, shown in Figure 9, starting from the real-world game, Game 0, where $\mathcal{Z}$ interacts with the honest parties running protocol EKE, and ending with the ideal-world game, Game 7, where $\mathcal{Z}$ interacts via dummy honest parties with functionality $\mathcal{F}_{\mathsf{pwKE}}$ which in turn interacts with simulator SIM. (This last game is not shown in Figure 9 because its code can be derived from the code of simulator SIM, Figure 8, and functionality $\mathcal{F}_{\mathsf{pwKE}}$, see Figure 23 in Appendix F.) We note that in each game in Figure 9 we write $\boxed{\text{output [...]}}$ for output of queries that service $\mathcal{Z}$'s interaction with EKEinstances, and we write "return [...]" for output of queries that service $\mathcal{Z}$'s interaction with $\mathcal{F}_{\mathsf{HIC}}$.

At each step we prove that the two consecutive games are indistinguishable, which implies the claim by transitivity of computational indistinguishability. Note that we argue security of EKE in the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model. Specifically, algorithm SIM emulates a "global" $\mathcal{F}_{\mathsf{HIC}}$ functionality which services any number of EKE protocol instances. Note that $\mathcal{Z}$ or $\mathcal{A}$ can call $\mathcal{F}_{\mathsf{HIC}}$ on keys which correspond to all strings $\hat{pw} = (\mathsf{fullsid}, b, pw)$ including for $\mathsf{fullsid}$ corresponding to sessions which were not (yet) started by $\mathcal{Z}$. Indeed, algorithm SIM treats queries pertaining to any key $\hat{pw}$ equally, and embeds random ciphertext c in response to Enc queries, random partial ciphertext s in response to AdvEnc queries, and random KA message M in response to AdvDec and Dec queries, saving the corresponding KA local state in $(\mathsf{backdoor}, \ldots)$ records. Since Dec is a wrapper over AdvDec we assume that the adversary uses only interface AdvDec, and we implement the EKE code of $\mathsf{P}^{\mathsf{sid}}$ using AdvDec as well.

The intuition for the simulation is that it sends an outgoing EKE message on behalf of $\mathsf{P}^{\mathsf{sid}}$ at random, since this is how HIC encryptions are formed. SIM

SIM interacts with environment $\mathcal{Z}$'s interface $\mathcal{A}$ and with functionality $\mathcal{F}_{\mathsf{pwKE}}$. W.l.o.g. we assume that $\mathcal{A}$ uses AdvDec to implement Dec queries to $\mathcal{F}_{\mathsf{HIC}}$.

Initialization: Set $\mathsf{Cset} = \{\}$, set $\mathsf{THIC}_{\hat{pw}}$ as an empty table and $\mathsf{c2pw}[c] := \perp$ for all values $\hat{pw}$ and c.

Notation (used in all security games in Figure 9)

Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$ be a shortcut for set $\{s \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (s, T)) \in \mathsf{THIC}_{\hat{pw}}\}$.
Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{c}$ be a shortcut for set $\{c \in \mathcal{D} : \nexists \hat{m} \text{ s.t. } (\hat{m}, c) \in \mathsf{THIC}_{\hat{pw}}\}$.
Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{m}$ be a shortcut for set $\{m \in \mathcal{D} : \nexists \hat{c} \text{ s.t. } (m, \hat{c}) \in \mathsf{THIC}_{\hat{pw}}\}$.

On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}_{\mathsf{pwKE}}$:

Set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $c \xleftarrow{r} \mathcal{D}$ (*abort* if $c \in \mathsf{Cset}$), add c to Cset, record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, c)$, return c.

Emulating functionality $\mathcal{F}_{\mathsf{HIC}}$:

- On $\mathcal{A}$'s query $(\mathsf{Enc}, \hat{pw}, M)$ to $\mathcal{F}_{\mathsf{HIC}}$: Set $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$. If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return c; Else pick $c \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{c}$ (*abort* if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvEnc}, \hat{pw}, m, T)$ to $\mathcal{F}_{\mathsf{HIC}}$: If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return c; Else pick $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$ (*abort* if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvDec}, \hat{pw}, c)$ to $\mathcal{F}_{\mathsf{HIC}}$: If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return m; Else pick $r \xleftarrow{r} \mathcal{R}$ and $(x, M) \xleftarrow{r} \mathsf{KA.msg}$, set $m \leftarrow (r, M)$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$ (*abort* if $\exists\, \hat{c} \neq c$ s.t. $(m, \hat{c}) \in \mathsf{THIC}_{\hat{pw}}$), save $(\mathsf{backdoor}, c, \hat{pw}, x)$, return m.

On $\mathcal{A}$'s message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only the first such message)

Retrieve record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, c)$ and do:

1. If there is record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, \hat{c})$: send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
2. Otherwise set $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and do the following:
   (a) If $\hat{pw} = \perp$ or $\hat{pw} = (\widehat{\mathsf{fullsid}}, \hat{b}, \cdot)$ for $(\widehat{\mathsf{fullsid}}, \hat{b}) \neq (\mathsf{fullsid}, \neg b)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$;
   (b) If $\hat{pw} = (\mathsf{fullsid}, \neg b, pw^*)$ retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$ and:
      i. service $\mathcal{F}_{\mathsf{HIC}}$'s query $(\mathsf{AdvDec}, (\mathsf{fullsid}, b, pw^*), c)$, retrieve $(\mathsf{backdoor}, c, (\mathsf{fullsid}, b, pw^*), x)$;
      ii. set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$.

Fig. 8: Simulator SIM for the proof of Theorem 2

services HIC encryption queries as $\mathcal{F}_{\mathsf{HIC}}$ does except that it collects the ciphertexts created by any encryption query and the ciphertexts chosen for every honest session in set Cset, and aborts if either process regenerates a ciphertext in Cset. Here we use the fact that even though an adversary can set the T part of the ciphertext $c = (s, T)$ resulting from an adversarial encryption query AdvEnc, the s part of c is chosen at random, and this prevents ciphertext collisions (except with negligible probability) if $|\mathcal{R}| \geq 2^{2\kappa}$. Hence, assuming that $\mathcal{R}$ is big enough, we have that (1) each adversarial ciphertext can be matched to (at most) one password on which it decrypts to a non-random value in space $\mathcal{M}$, and (2) the simulator can extract this unique password and retrieve the corresponding plaintext (SIM stores the key $\hat{pw}$ which was used to create ciphertext c in the $\mathsf{c2pw}$ table by setting $\mathsf{c2pw}[c] \leftarrow \hat{pw}$). Moreover, since by the same collision-resistant property of $\mathcal{F}_{\mathsf{HIC}}$ ciphertexts the adversary cannot "hit" any honest session $\mathsf{P}^{\mathsf{sid}}$'s ciphertext c via an encryption query, the decryption of $\mathsf{P}^{\mathsf{sid}}$'s ciphertext on each password is also a random value in $\mathcal{M}$. By the message-randomness property of KA, simulator SIM can embed messages of fresh KA instances into each decryption query, and combining this with fact (1) above allows for a reduction of EKE instances corresponding to "wrong" password guesses to the KA's security.

Let q_{IC} be the bound on the number of queries $\mathcal{Z}$ makes to the interfaces of the (randomized) ideal cipher $\mathcal{F}_{\mathsf{HIC}}$, and let q_P be the upper-bound on the number of honest EKE sessions $\mathsf{P}^{\mathsf{sid}}$ which $\mathcal{Z}$ invokes for any identifiers P, sid. [9] Let $\varepsilon_{\mathsf{KA.sec}}$ and $\varepsilon_{\mathsf{KA.rand}}$ be the upper-bounds on the distinguishing advantage against, respectively, the security and the random-message properties of the key exchange scheme KA (see Section 2) of an adversary whose computational resources are roughly those of an environment $\mathcal{Z}$ extended by execution of $q_{IC} + q_P$ instances of the key exchange scheme KA.[10]

For space-constraint reasons we defer the details of the game changes and reductions to Appendix B, but we show the code of all successive games in Figure 9. By the arguments for indistinguishability of successive games, the total distinguishing advantage of environment $\mathcal{Z}$ between the real-world and the ideal-world interaction is upper-bounded by the following expression, which sums up the bounds given by equations (8), (9), (10), (11) in Appendix B:

$$(q_{IC} + q_P)\left[\frac{1}{|\mathcal{R}|} \cdot \left\{2q_P + q_{IC} + 2 \cdot \frac{q_{IC} + q_P}{|\mathcal{M}|}\right\} + \varepsilon_{\mathsf{KA.rand}} + q_P \cdot \varepsilon_{\mathsf{KA.sec}}\right] \quad (2)$$

Since this quantity is negligible if $\mathcal{R} = \{0,1\}^n$ for $n = O(\kappa)$, it implies Theorem 2. $\square$

Notes on Exact Security. The dominating factors are $(q_{IC} + q_P)^2/|\mathcal{R}|$ and $(q_{IC} + q_P) \cdot (\varepsilon_{\mathsf{KA.rand}} + q_P \cdot \varepsilon_{\mathsf{KA.sec}})$. The first factor is due to possible collisions in Half-Ideal Cipher, and it is unavoidable using an arbitrary HIC realization

[9] We assume that $\mathcal{Z}$ invokes at most two sessions for any fixed identifier sid.

[10] This bound involves $q_{IC} + q_P$ instead of q_P key exchange instances because our reductions to KA security run $\mathsf{KA.msg}$ for each adversarial AdvDec query to $\mathcal{F}_{\mathsf{HIC}}$.

Game 0: real-world interaction

initialization

Initialize $\mathsf{Cset} = \{\}$ and $\forall\, \hat{pw}$ empty $\mathsf{THIC}_{\hat{pw}}$

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, b, pw)$
$(x, M) \xleftarrow{r} \mathsf{KA.msg}$
$c \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, M)$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, $\boxed{\text{output } c}$

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one):

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, \cdot, \perp)$:
  $(\hat{r}, \hat{M}) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$
  $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ and $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K)}$

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, M)$:

$r \xleftarrow{r} \mathcal{R}$, set $m \leftarrow (r, M)$
If $\exists\, c$ s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return c
else:
  pick $c \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{c}$,
  add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$
  return c

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:

if $\exists\, c$ s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return c
else:
  $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$,
  add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$
  return c

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:

if $\exists\, m$ s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return m
else:
  $m \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{m}$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$
return m

Game 1: randomizing protocol communication

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

set $(\mathsf{fullsid}, b, \hat{pw})$ as in Game 0
$(x, M) \xleftarrow{r} \mathsf{KA.msg}$, $r \xleftarrow{r} \mathcal{R}$, $c \xleftarrow{r} \mathcal{D}$
abort if $((r, M), *) \in \mathsf{THIC}_{\hat{pw}}$ or $c \in \mathsf{Cset}$
add $((r, M), c)$ to $\mathsf{THIC}_{\hat{pw}}$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, $\boxed{\text{output } c}$

Game 2: binding adversarial ciphertexts to passwords

on $\mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, M)$ or $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:

Before adding c to Cset, do the following:
  abort if $c \in \mathsf{Cset}$
  set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$

Game 3: adding trapdoors to decryption

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:

if $\exists m$ s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return m, otherwise:
  $(x, M) \xleftarrow{r} \mathsf{KA.msg}(1^\kappa)$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$
  abort if $(m, *) \in \mathsf{THIC}_{\hat{pw}}$
  add (m, c) to $\mathsf{THIC}_{\hat{pw}}$
  save $(\mathsf{backdoor}, c, \hat{pw}, x)$, return m

Game 4: KA messages via AdvDec

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

set $(\mathsf{fullsid}, b, \hat{pw})$ as in Game 0
$c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$
retrieve $(\mathsf{backdoor}, c, \hat{pw}, x)$
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$, $\boxed{\text{output } c}$

Game 5: extracting passwords

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $\mathsf{rec} = (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, pw, \cdot, \hat{c}, \hat{K})$
  s.t. $\mathcal{Z}$ sent c to $\mathsf{CP}^{\mathsf{sid}}$:
    $K \leftarrow \hat{K}$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$:
      retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$,
      set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$
    else:
      $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, K)$
  $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K)}$

Game 6: delaying password usage

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$
$c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \perp, c, \perp)$, $\boxed{\text{output } c}$

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:

if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \perp, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, \neg b, pw, \perp, \hat{c}, \hat{K})$:
    $K \leftarrow \hat{K}$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$:
      query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, b, pw), c)$,
      retrieve $(\mathsf{backdoor}, c, \cdot, x)$
      retrieve $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$,
      set $K \leftarrow \mathsf{KA.key}(x, \hat{M})$
    else:
      $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, x, c, K)$
  $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K)}$

Fig. 9: Game changes for the proof of Theorem 2 (compare Fig. 8 for notation)

because it is the probability of generating the same ciphertext c as an encryption of two different KA instances under two different passwords, which would also form an explicit attack on the security of EKE (the adversary would effectively make two password guesses in one on-line interaction). However, whereas the bound $(q_{IC})^2/|\mathcal{R}|$ is tight if the encryption is modeled as a Half-Ideal Cipher, we do not know if it is tight in relation to the specific modified 2-Feistel instantiation of Half-Ideal Cipher, because we do not know how to stage an explicit attack on EKE using modified 2-Feistel along these lines. This relates to the fact that whereas the modified 2-Feistel realizes functionality $\mathcal{F}_{\mathsf{HIC}}$, this functionality allows more freedom to the adversary than the modified 2-Feistel construction. Namely, whereas $\mathcal{F}_{\mathsf{HIC}}$ allows the adversary to encrypt any messages M using a ciphertext $c = (s, T)$ where T can be freely set, the same is not true about the modified 2-Feistel construction, where for any fixed M the adversary can choose T from the set of values of the form $T = M/\mathsf{H}(pw, r)$ for some r.

The second factor is due to reductions to KA security properties. Note that some KA schemes, e.g. Diffie-Hellman, have perfect message-randomness, i.e. $\varepsilon_{\mathsf{KA.rand}} = 0$. Further, if the KA scheme is *random self-reducible*, as is Diffie-Hellman, then this factor can be reduced to $\varepsilon_{\mathsf{KA.sec}}$ because a reduction to KA security for the transition between Games 4 and 5, see Appendix B, can then be modified so that it deals with all honest sessions at once instead of staging a hybrid argument over all sessions, and it embeds randomized versions of the KA challenge into each decryption query rather than guessing a target query.

5.1 EKE with Half-Ideal Cipher: the KEM version

In Figure 10 we show protocol $\mathsf{EKE\text{-}KEM}$, which is a KEM version of the EKE protocol using a Half-Ideal Cipher. In the 1-flow protocol EKE considered in Figure 7, the message flows are generated by a single-round KA scheme, whereas here we consider an EKE variant which is built from any two-flow key exchange, i.e. KEM, see Section 2.2. The drawback is that it is 2-flow instead of 1-flow, but the benefits are that the HIC can be used only for one message, so if KEM is instantiated with Diffie-Hellman and HIC is implemented using $\mathsf{m2F}$, this implies a single RO hash onto a group per party instead of two such hashes. Moreover, this version of EKE can use any CPA-secure KEM as a black box, as long as the KEM satisfies the anonymity and uniform public keys properties, which implies, e.g., lattice-based UC PAKE given any lattice-based KEM with these properties.

Note that in the protocol of Fig. 10 party P_0 outputs a random session key if the key confirmation message τ fails to verify. This is done only so that the protocol conforms to the implicit-authentication functionality $\mathcal{F}_{\mathsf{pwKE}}$. In practice P_0 could output $\perp$ in this case, and this would implement explicit authentication in the P_1-to-P_0 direction.

Theorem 3. *If* KEM *is IND secure, anonymous, and has uniform public keys in domain $\mathcal{PK}$ (see Section 2.2),* HIC *is a UC Half-Ideal Cipher in domain $\mathcal{R} \times \mathcal{PK}$, and* H *is an RO hash, then protocol* $\mathsf{EKE\text{-}KEM}$ *realizes the UC PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$.*

- KEM scheme $\mathsf{KEM} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$ with public key space $\mathcal{PK}$
- Half-Ideal Cipher HIC on domain $\mathcal{R} \times \mathcal{PK}$ for $\mathcal{R} = \{0,1\}^{\Omega(\kappa)}$
- Random oracle hash H onto $\{0,1\}^\kappa$

$\underline{\mathsf{P}_0 \text{ on } \mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1, pw_0)}$		$\underline{\mathsf{P}_1 \text{ on } \mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_1, \mathsf{P}_0, pw_1)}$
	(Assume $\mathsf{P}_0 \leq_{lex} \mathsf{P}_1$ and let $\mathsf{fullsid} = (\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1)$)	
$(sk, pk) \xleftarrow{r} \mathsf{kg}$		
$c \leftarrow \mathsf{HIC.Enc}((\mathsf{fullsid}, pw_0), pk)$	$\xrightarrow{\quad c \quad}$	$pk' \leftarrow \mathsf{HIC.Dec}((\mathsf{fullsid}, pw_1), c)$
		$(e, K) \leftarrow \mathsf{enc}(pk')$,
		$\tau \leftarrow \mathsf{H}(K, pk')$
$K \leftarrow \mathsf{dec}(sk, e)$	$\xleftarrow{\quad e, \tau \quad}$	output $K_1 \leftarrow \mathsf{H}(K)$
if $\tau = \mathsf{H}(K, pk)$ output $K_0 \leftarrow \mathsf{H}(K)$, else $K_0 \xleftarrow{r} \{0,1\}^\kappa$		

Fig. 10: $\mathsf{EKE\text{-}KEM}$: Encrypted Key Exchange with Half-Ideal Cipher (KEM version)

The proof of Theorem 3 is deferred to Appendix C. It follows the same blueprint as the proof of Theorem 2. The most important intuition needed for the adaptation of the proof of Theorem 2 to the proof of Theorem 3 is why it works for KEMs that satisfy the anonymity property: The key issue is that we need anonymity of the KEM ciphertext e only for honest keys pk and not for adversarial ones, and the reason for this is that the only non-random pk under which an honest party encrypts is the key pk decrypted under a unique password guess pw^* used in the adversarial ciphertext c this party receives. If pw^* equals to P_1's password pw then this session is already successfully attacked, so the non-randomness of P_1's ciphertext is not an issue. But if $pw^* \neq pw$ then KEM ciphertext e is effectively encrypted under key $pk' = \mathsf{AdvDec}(pw, c)$ which is random, and the key confirmation works as a commitment to the KEM key pk decrypted from HIC ciphertext c, hence also to the password used in that decryption. This commitment is also effectively encrypted under the KEM session key K, hence it can be verified only by a party which created pk and HIC-encrypted it under the right pw. Here we again rely on the property of HIC, which just like IC assures that decryption under any password except for the unique password committed in the ciphertext results in a random plaintext, i.e. a random KEM public key pk, which makes the KEM session key K encrypted under such pk hidden to the adversary by KEM security.

We note that the key confirmation could involve directly pw instead of pk, but pk is a commitment to pw unless the adversary creates a collision in HIC plaintext, and using pk instead of pw lets P_0 erase pw after sending its first message. This way an adaptive compromise on party P_0 during protocol execution allows for offline dictionary attack on the password, but does not leak it straight away. (Note that adaptive party compromise is not part of our security model.) We note also that RO hash H can probably be replaced by a key derivation function which is both a CRH (because it needs to commit to pk) and a PRF (because it must encrypt this commitment under K), but since

HIC implies RO hash (and indeed our m2Fuses it) we opt for the simpler option of RO hash to compute the authenticator.

6 Applications of Half-Ideal Cipher to aPAKE

Gu et al. [41] proposed an asymmetric PAKE protocol called KHAPE which is a generic compiler from any UC *key-hiding* Authenticated Key Exchange (AKE), using an Ideal Cipher on the domain formed by (private, public) key pairs of the AKE. We show that KHAPE realizes UC aPAKE if IC is replaced by HIC. For lack of space the proof of the following Theorem is deferred to Appendix D. For reference, for AKE functionality $\mathcal{F}_{\mathsf{khAKE}}$ see e.g., [41], and for aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ see Figure 24 in Appendix F.

Theorem 4. *Protocol* KHAPE *of [41] realizes the UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *if the AKE protocol realizes the Key-Hiding AKE functionality* $\mathcal{F}_{\mathsf{khAKE}}$ *assuming that* kdf *is a secure PRF and* HIC *is a half-ideal cipher over message space of private and public key pairs in AKE.*

We note that Freitas et al. [37] showed a UC aPAKE which improves upon protocol KHAPE of [41] in round complexity. The aPAKE of [37] relies on IC in a similar way as protocol KHAPE, and the proof therein should also generalize to the case when IC is replaced by HIC.

References

1. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Heidelberg (Aug 2020). `https://doi.org/10.1007/978-3-030-56784-2_10`
2. 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 (Apr 2008). `https://doi.org/10.1007/978-3-540-79263-5_22`
3. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic mceliece: Nist round 3 submission, `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions` (2021)
4. Alkim, .E., Bos, J.W., Ducas, L., Longa, P., Mironov, I., Naehrig, M., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebil, D.: Frodokem: Nist round 3 submission, `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions` (2021)
5. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the indifferentiability of key-alternating ciphers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 531–550. Springer, Heidelberg (Aug 2013). `https://doi.org/10.1007/978-3-642-40041-4_29`

6. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 719–737. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
7. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_33
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Advances in Cryptology – EUROCRYPT 2000. pp. 139–155. Springer (2000)
9. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
10. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: IEEE Computer Society Symposium on Research in Security and Privacy – S&P 1992. pp. 72–84. IEEE (1992)
11. Benhamouda, F., Blazy, O., Ducas, L., Quach, W.: Hash proof systems over lattices revisited. In: Public-Key Cryptography – PKC 2018. Public-Key Cryptography – PKC 2018, vol. 10770, pp. 644–674. Springer (2018). https://doi.org/10.1007/978-3-319-76581-5_22
12. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 967–980. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516734
13. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli: a cross-platform permutation. Cryptology ePrint Archive, Report 2017/630 (2017), http://eprint.iacr.org/2017/630
14. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 313–314. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_19
15. Black, J., Rogaway, P.: Ciphers with arbitrary finite domains. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 114–130. Springer, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-45760-7_9
16. Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_21
17. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: Crystals - kyber: A cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS P). pp. 353–367 (2018). https://doi.org/10.1109/EuroSP.2018.00032
18. Boyko, V., MacKenzie, P.D., 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 (May 2000). https://doi.org/10.1007/3-540-45539-6_12
19. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: IEEE Symposium on Foundations of Computer Science – FOCS 2001. pp. 136–145. IEEE (2001)

20. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Advances in Cryptology – EUROCRYPT 2005. pp. 404–421. Springer (2005)
21. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. Cryptology ePrint Archive, Report 2005/196 (2005), `http://eprint.iacr.org/2005/196`
22. Coron, J.S., Dodis, Y., Mandal, A., Seurin, Y.: A domain extender for the ideal cipher. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 273–289. Springer, Heidelberg (Feb 2010). `https://doi.org/10.1007/978-3-642-11799-2_17`
23. Coron, J.S., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 1–20. Springer, Heidelberg (Aug 2008). `https://doi.org/10.1007/978-3-540-85174-5_1`
24. Dachman-Soled, D., Katz, J., Thiruvengadam, A.: 10-round Feistel is indifferentiable from an ideal cipher. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 649–678. Springer, Heidelberg (May 2016). `https://doi.org/10.1007/978-3-662-49896-5_23`
25. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Trans. Symm. Cryptol. **2018**(4), 1–38 (2018). `https://doi.org/10.13154/tosc.v2018.i4.1-38`
26. Dai, Y., Seurin, Y., Steinberger, J.P., Thiruvengadam, A.: Indifferentiability of iterated Even-Mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 524–555. Springer, Heidelberg (Aug 2017). `https://doi.org/10.1007/978-3-319-63697-9_18`
27. Dai, Y., Steinberger, J.P.: Indifferentiability of 8-round Feistel networks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 95–120. Springer, Heidelberg (Aug 2016). `https://doi.org/10.1007/978-3-662-53018-4_4`
28. D'Anvers, J.P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2018. pp. 282–305. Springer International Publishing, Cham (2018)
29. Desai, A.: The security of all-or-nothing encryption: Protecting against exhaustive key search. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 359–375. Springer, Heidelberg (Aug 2000). `https://doi.org/10.1007/3-540-44598-6_23`
30. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976)
31. Ding, J., Alsayigh, S., Lancrenon, J., RV, S., Snook, M.: Provably secure password authenticated key exchange based on rlwe for the post-quantum world. In: Handschuh, H. (ed.) Topics in Cryptology – CT-RSA 2017. pp. 183–204. Springer International Publishing, Cham (2017)
32. Dodis, Y., Puniya, P.: Feistel networks made public, and applications. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 534–554. Springer, Heidelberg (May 2007). `https://doi.org/10.1007/978-3-540-72540-4_31`
33. Dodis, Y., Stam, M., Steinberger, J.P., Liu, T.: Indifferentiability of confusion-diffusion networks. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 679–704. Springer, Heidelberg (May 2016). `https://doi.org/10.1007/978-3-662-49896-5_24`

34. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT'91. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (Nov 1993). https://doi.org/10.1007/3-540-57332-1_17
35. Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R., Wood, C.: Hashing to elliptic curves, irft-cfrg active draft, https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/ (2022)
36. Freitas Dos Santos, B., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2023)
37. Freitas Dos Santos, B., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2022)
38. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34
39. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Advances in Cryptology – CRYPTO 2006. pp. 142–159. Springer (2006)
40. Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. p. 516–525. CCS '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1866307.1866365, https://doi.org/10.1145/1866307.1866365
41. Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: Asymmetric PAKE from key-hiding key exchange. In: Advances in Cryptology - Crypto 2021. pp. 701–730 (2021), https://ia.cr/2021/873
42. Guo, C., Lin, D.: Improved domain extender for the ideal cipher. Cryptography Commun. **7**(4), 509–533 (dec 2015). https://doi.org/10.1007/s12095-015-0128-7, https://doi.org/10.1007/s12095-015-0128-7
43. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. pp. 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
44. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 89–98. ACM Press (Jun 2011). https://doi.org/10.1145/1993636.1993650
45. Jaulmes, É., Joux, A., Valette, F.: On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 237–251. Springer, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-45661-9_19
46. Jiang, S., Gong, G., He, J., Nguyen, K., Wang, H.: Pakes: New framework, new techniques and more efficient lattice-based constructions in the standard model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. pp. 396–427. Springer International Publishing, Cham (2020)
47. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices (2009)

48. Kilian, J., Rogaway, P.: How to protect DES against exhaustive key search. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 252–267. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_20
49. Kim, T., Tibouchi, M.: Invalid curve attacks in a GLS setting. In: Tanaka, K., Suga, Y. (eds.) IWSEC 15. LNCS, vol. 9241, pp. 41–55. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-319-22425-1_3
50. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_33
51. Maram, V., Grubbs, P., Paterson, K.G.: Anonymous, robust post-quantum public key encryption. In: EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2022)
52. Marlinspike, M., Perrin, T.: The X3DH key agreement protocol, https://signal.org/docs/specifications/x3dh/ (2016)
53. McQuoid, I., Rosulek, M., Roy, L.: Minimal symmetric PAKE and 1-out-of-n OT from programmable-once public functions. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. (2020). https://doi.org/10.1145/3372297.3417870, https://eprint.iacr.org/2020/1043
54. Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_40
55. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_31
56. Shannon, C.E.: Communication theory of secrecy systems. The Bell System Technical Journal **28**(4), 656–715 (1949). https://doi.org/10.1002/j.1538-7305.1949.tb00928.x
57. Tibouchi, M.: Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 139–156. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-662-45472-5_10
58. Winternitz, R.S.: Producing a one-way hash function from DES. In: Chaum, D. (ed.) CRYPTO'83. pp. 203–207. Plenum Press, New York, USA (1983)
59. Xagawa, K.: Anonymity of nist pqc round 3 kems. In: EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2022)
60. Zhang, J., Yu, Y.: Two-round pake from approximate sph and instantiations from lattices. In: ASIACRYPT (3). pp. 37–67. Springer (2017). https://doi.org/10.1007/978-3-319-70700-6_2

A Game changes for Theorem 1

In this section we present the game changes used in our proof of Theorem 1. We refer the reader to Section 4 for the notation, and to Figures 4 and 5 for the specification of all successive games.

Let P_i be the probability that the environment $\mathcal{Z}$ outputs 1 when interacting with the i-th game. We will show that $|P_i - P_{i+1}|$ is negligible for every i.

Game 0 *(ideal world)*: This is the ideal-world game played between $\mathcal{Z}$ and SIM. We describe it formally in Figure 4, except that Game 0 omits the gray boxes and AdvDec is answered just as in our $\mathcal{F}_{\mathsf{HIC}}$ construction: $(r, M) \xleftarrow{r} \{m \in \{0,1\}^n \times \mathbb{G} : \nexists \hat{c} \text{ s.t. } (\hat{m}, \hat{c}) \in \mathsf{THIC}_{pw}\}$.

Game 1 *(adding* usedR *and randomizing* AdvDec*)*: In this first game change we start by randomizing AdvDec and then adding aborts for certain accesses to the randomness r used by $\mathsf{m2F}$, see Figure 4. We randomize $\mathsf{AdvDec}(pw, (s, T))$ by picking $(r, M) \xleftarrow{r} \mathcal{D}$ and then aborting if this pair $((r, M), (s, T))$ happens to be in the table THIC_{pw}. This is clearly a negligible change, in fact, as long as this negligible abort (which happens with probability at most $|\mathsf{THIC}_{pw}|/(2^n \cdot |\mathbb{G}|)$) does not happen then the games are the same.

Moreover, we want to avoid distinct (fresh)[11] calls to Enc and Dec reusing the same r (or them being called after $\mathsf{TH}(pw, r)$ has been set). This is motivated by the fact that the Feistel circuit would impose, for a plaintext-ciphertext pair $((r, M), (s, T)) \in \mathsf{THIC}_{pw}$, the relation $M/T = \mathsf{H}(pw, r)$. If the same (pw, r) pair were used for multiple calls, then we can't expect M/T to be the same except with negligible probability, and thus we wouldn't be able to embed the correct value (since there are multiple) into TH. Similarly, if the r used by $\mathsf{m2F}$ is already at TH, the adversary could notice the discrepancy to the relation in the Feistel circuit. In fact, in the current game a direct call to $\mathsf{m2F}$ by the environment has no relationship to the other oracle tables, and in particular we need to disallow the adversary to query $\mathsf{TH}(pw, r)$ right after such a $\mathsf{m2F}$ invocation. This is a valid game change (i.e. the adversary can't force such an abort except with negligible probability) since the r used is not leaked, neither for a Dec nor an Enc call. The exception is when $\mathcal{Z}$ does run the decryption circuit (hence learning r through the $\mathsf{BC.Dec}$ call that is part of it) after the $\mathsf{m2F}$ call. To avoid this we introduce a flag denoted $\mathsf{m2F}$ for such calls before the decryption circuit is attempted.

Now it is clear that the size $\sum_r |\mathsf{usedR}_{pw}|$ of the set of used r is bound by $q_{\mathsf{m2F}} + q_{\mathsf{BC}}$. Hence no usedR abort happens except with (negligible) probability $(q_{\mathsf{m2F}} + q_{\mathsf{BC}}) \cdot (q_{\mathsf{m2F}} + q_{\mathsf{BC}} + q_{\mathsf{H}})/2^n$.

Therefore this game change is indistinguishable except with probability

$$|P_0 - P_1| \leq \frac{(q_{\mathsf{m2F}} + q_{\mathsf{BC}})^2}{2^n \cdot |\mathbb{G}|} + \frac{(q_{\mathsf{m2F}} + q_{\mathsf{BC}}) \cdot (q_{\mathsf{m2F}} + q_{\mathsf{BC}} + q_{\mathsf{H}})}{2^n} \leq 2 \cdot \frac{q^2}{2^n} \qquad (3)$$

Game 2 *(replacing decryption* $\mathsf{m2F.Dec}$ *by circuit)*: We replace queries to $\mathsf{m2F.Dec}$ by the circuit of our construction. First we argue that this is a valid change for a fresh $\mathsf{m2F.Dec}$ query, see Fig. 11.

[11] In this paragraph, and henceforth, a call usually, but not always, implicitly means a fresh call, i.e., this call is not a simple table lookup.

In this simplified case the $\mathsf{BC.Dec}$ call is also fresh, in either the current game or the previous. This implies that it calls a fresh AdvDec itself, fixing the H table so that the output M is the same as in Game 1, namely it comes from an AdvDec query. Suppose instead that $(k, r, s) \in \mathsf{TBC}$ for some r. Then this triple was added to the table by either a $\mathsf{BC.Enc}$ or $\mathsf{BC.Dec}$ query. The latter cannot happen since such a query would have inputs (k, s) and therefore its $\mathsf{AdvDec}(pw, (s, T))$ call would have populated THIC_{pw} - note that we are using the fact that bckey.col abort was not reached. Similarly, a $\mathsf{BC.Enc}(k, r)$ that returns s would have run $(s, T) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$ so that the $\mathsf{m2F.Dec}$ query couldn't be fresh either. We conclude that the newly introduced $\mathsf{BC.Dec}$ calls are fresh, and that these queries make fresh calls to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$.

But then the internal AdvDec call is generating (r, M) uniformly, so r is uniform. There are only two side effect of this game change in the environment's view: 1) $h \leftarrow \mathsf{TH}(pw, r)$ now satisfies $h = M/T$, while in the previous game this is not true right after the $\mathsf{m2F.Dec}$ query (an $\mathsf{H}(pw, r)$ query would return an independent, uniform value) and 2) (k, r, s) is added to TBC where $k = \mathsf{H}'(pw, T)$. But we added r to usedR with flag $\mathsf{m2F}$ and our usedR aborts in Game 1 guarantee that there is no call $\mathsf{H}(pw, r)$ or $\mathsf{TBC}(k, r)$ before the adversary itself runs the decryption circuit, i.e. $\mathsf{BC.Dec}(k, s)$. But this call would embed the same relationship in the TH and TBC table since bckey.abort does not happen, so this change is not visible to the adversary.

We do need to take into consideration the new aborts that are possible by this game change. The H' query that is now implicitly called by $\mathsf{m2F.Dec}$ can only abort if it is fresh and there is a collision with the H' table or TBC table (see definition of H' queries in Figure 3). This happens with probability at most $(|\mathsf{TH}'| + |\mathsf{TBC}|)/2^{\mu}$ for each added H' call. The $\mathsf{BC.Dec}$ procedure, which if fresh is executing its innermost if, will only abort if either AdvDec aborts, advdec.abort is reached or rcol.abort happens. As we argued in the previous paragraph, r is uniform hence we obtain the negligible probability bound $|\mathsf{THIC}_{pw}|/(2^n \times |\mathbb{G}|) + |\mathsf{usedR}_{pw}|/2^n + |\mathsf{TBC}_k|/2^n + |\mathsf{TH}_{pw}|/2^n \leq 4 \cdot q/2^n$.

Finally, looking at our argument above, we see that even if $\mathsf{BC.Dec}(k, s)$ was not a fresh query during a (necessarily non fresh) $\mathsf{m2F.Dec}$ call, the r paired with (k, s) in the table TBC satisfies $(r, \mathsf{TH}(pw, r) \cdot T) = \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(pw, (s, T))$ where $k = \mathsf{TH}'(pw, T)$. As we saw above this follows from how both $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ are defined. We conclude that the change to the circuit is valid even for non fresh AdvDec queries and we get

$$|P_2 - P_1| \leq q^2 \cdot \left\{ \frac{2}{2^{\mu}} + \frac{4}{2^n} \right\} \tag{4}$$

Game 3 *(using* AdvEnc *to answer* Enc *queries)*: We replace $\mathsf{m2F.Enc}(pw, M)$ by a call to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(pw, (r, M), T)$ using uniform T. The goal is to link the AdvEnc queries done in $\mathsf{BC.Enc}$ with the way $\mathsf{m2F.Enc}$ is computed - this will help with our next goal of changing Enc to match the circuit. This modification skews the distribution of THIC_{pw}, but the statistical difference this introduces is negligible. The difference is that in Game 2 (s, T) is chosen uniformly from

$$\begin{array}{l} \underline{\text{on query } \mathsf{m2F.Dec}(pw,(s,T)):} \\ \text{if } \nexists(r,M) \text{ s.t. } ((r,M),(s,T)) \in \mathsf{THIC}_{pw}: \\ \quad k \leftarrow \mathsf{H}'(pw,T) \\ \quad r \leftarrow \mathsf{BC.Dec}(k,s) \\ \quad M \leftarrow \mathsf{H}(pw,r) \cdot T \\ \quad \text{add tag } \mathsf{m2F} \text{ to } r \in \mathsf{usedR}_{pw} \\ \text{return } M \end{array}$$

Fig. 11: Fresh queries to $\mathsf{m2F.Dec}$ are replaced by the circuit

set $\{c \in \{0,1\}^n \times \mathbb{G} : \nexists \hat{m} \text{ s.t. } (\hat{m}, c) \in \mathsf{THIC}_{pw}\}$, while in Game 3 first T is chosen uniformly in $\mathbb{G}$ and then s is chosen uniformly from set $\{s \in \{0,1\}^n : \nexists \hat{m} \text{ s.t. } (\hat{m},(s,T)) \in \mathsf{THIC}_{pw}\}$. Since there are at most q elements in table THIC_{pw}, the skew this introduces on the distribution of a chosen pair (s,T) is at most $4q/(2^n \cdot |\mathbb{G}|)$ per encryption query, leading to the following upper bound:

$$|P_3 - P_2| \leq \frac{4 \cdot q \cdot q_{\mathsf{Enc}}}{2^n \cdot |\mathbb{G}|} \leq 4 \cdot \frac{q^2}{2^n \cdot |\mathbb{G}|} \tag{5}$$

Game 4 *($\mathsf{m2F.Enc}$ can also be replaced by the two-round Feistel circuit)*: We now assert that replacing Enc from the previous game by the $\mathsf{m2F}$ encryption circuit is also a valid game change.

Since we check $r \notin \mathsf{usedR}_{pw}$, the r that is picked by Enc does not appear in the H table and thus $T \leftarrow M/\mathsf{H}(pw,r)$ will assign an uniform value to T just as Game 3 does. The $s \leftarrow \mathsf{BC.Enc}(k,r)$ call, much like our $\mathsf{BC.Dec}$ query in Game 2, will in turn call AdvEnc indirectly for $\mathsf{m2F.Enc}$ making the output of the latter the same as in the previous game. As in Game 2, the side effect of modifying Enc in this way is that now we have certain relationships between the table values. But since r is by definition not leaked by Enc this is a negligible change just as before. In fact, looking at the newly introduced aborts, we see that the only possible ones (note we may assume $r \notin \mathsf{usedR}_{pw}$) are the ones inside the H' query. This leads us to the bound

$$|P_4 - P_3| \leq \frac{2q^2}{2^\mu} \tag{6}$$

Game 5 *(H is a random oracle)*: If we are to reach the real-world game described in Figure 9, we need to show that H is indistinguishable from a random oracle. Currently, the only obstacle in the way of this proof is that TH is not only modified in response to a (direct or indirect) H query, but it is also changed during a $\mathsf{BC.Dec}$ call. In this game we drop AdvDec usage in $\mathsf{BC.Dec}$ and make clearer that this modification to TH is still uniform. We start this process by expanding $\mathsf{BC.Dec}$, see Figure 12.

```
on query BC.Dec(k, s)

if ∄ r s.t. (k, r, s) ∈ TBC:
    if k = TH′(pw, T):
        if ∄m s.t. (m, (s, T)) ∈ THIC_pw:
            (r, h) ←r D
            if r ∈ usedR_pw abort, else add r to it with tag m2F
            M ← h · T
            if ∃ĉ s.t. ((r, M), ĉ) ∈ THIC_pw then abort
            add ((r, M), (s, T)) to THIC_pw
        else:
            let ((r, M), (s, T)) ∈ THIC_pw
            h ← M · T^{-1}
        if ∃ŝ s.t. (k, r, ŝ) ∈ TBC then abort                  (advdec.abort)
        if ∃ĥ s.t. (pw, r, ĥ) ∈ TH then abort                   (rcol.abort)
        add (pw, r, h) to TH
    else:
        r ←r {r ∈ {0, 1}^n : ∄ŝ s.t. (k, r, ŝ) ∈ TBC}
    add (k, r, s) to TBC
remove tag m2F from record r ∈ usedR_pw if k = TH′(pw, T)
return r
```

Fig. 12: Expanding BC.Dec

In fact, if we look thoroughly at the current game, we notice that the innermost else in this figure of the expanded BC.Dec query will never be reached. Namely, say a (necessarily fresh) BC.Dec query reaches this line in the execution. Then there is m such that $(m, (s, T))$ is in THIC_{pw}. But since we removed direct AdvEnc and AdvDec queries from m2F invocations, this tuple $((r, M), (s, T))$ must have been added to THIC_{pw} by a BC query. The only BC.Dec query that could have caused $((\cdot, \cdot), (s, T))$ to have been added to THIC_{pw} is one with (k, s) as input, which would make the current query not fresh (i.e. a contradiction). Similarly, a BC.Enc query couldn't have added $((\cdot, \cdot), (s, T))$ to THIC_{pw} since this implies that $(k, \cdot, s)$ would have been added to TBC - which again would contradict the freshness of the current BC.Dec query.

Moreover, considering the above we can conclude that if either $\exists \hat{h}$ s.t. $(pw, r, \hat{h}) \in \mathsf{TH}$ or $\exists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}$ with $k = \mathsf{TH}'(pw, T)$ is true, then $r \in \mathsf{usedR}_{pw}$. In particular, if the latter is not the case then a call to $\mathsf{H}(pw, r)$ returns an uniform h. So we can let a H query in BC.Dec pick h by itself, instead of doing $(r, h) \xleftarrow{r} \mathcal{D}$ ourselves. We can also assume that (k, r) is available in the TBC table, so that we are allowed to drop this abort in BC.Dec as it is already caught by the usedR_{pw} abort.

The above remarks allows us to simplify BC.Dec considerably for Game 5 while not changing the view of the environment: $P_5 = P_4$.

Game 6 *(simplifying parameters)*: With our previous game changes THIC_{pw} is now only accessed/modified by (possibly indirect) BC.Enc and BC.Dec queries. It is clear from their definition that any call to $\mathcal{F}_{\mathsf{HIC}}$.AdvEnc uses the correct T, namely these calls return $\hat{T} = T$. So there is no need to return $\hat{T}$ and we can drop this component of the output. Similarly, any tuple $((r, M), (s, T)) \in \mathsf{THIC}_{pw}$ satisfies $M = \mathsf{TH}(pw, r) \cdot T$ hence we can remove this component of the table THIC_{pw}, i.e., we now use triples $(r, (s, T))$ and recover M with this equation when needed. As these are just synctactic changes, the games are the same: $P_6 = P_5$.

Game 7 *(replacing* (pw, T) *by its* H' *output)*: Since there are no collisions in the H' table, every (pw, T) pair that appears in a call to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$, or a modification to THIC_{pw} in $\mathsf{BC.Dec}$, corresponds to a unique k s.t. $k := \mathsf{H}'(pw, T)$[12]. So we can switch the parameters of the table THIC (and consequently $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$) from (pw, r, T) to (k, r). Besides this, the H queries in $\mathsf{BC.Enc}$ and $\mathsf{BC.Dec}$ can be delayed indefinitely until the adversary actually queries these tables, so we drop these extraneous calls from the definition of BC.

Once again, since we are avoiding our aborts this game change is immaterial and we get $P_7 = P_6$.

Game 8 *(*THIC *is redundant)*: We drop THIC altogether, since in the previous game it is always copied over to TBC. To be precise, if THIC_k is not empty - which at this point implies that there exists (a unique) (pw, T) with $\mathsf{TH}'(pw, T) = k$ - then all subsequent (resp. past) accesses and modifications to $\mathsf{TBC}(k, \cdot)$ are (resp. were) done through invoking $\mathcal{F}_{\mathsf{HIC}}$, that is, using THIC_k. This follows since we are avoiding bckey.abort.

The resulting $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ using TBC directly is presented in Figure 13. As these queries are now only made during a $\mathsf{BC.Enc}$ call, we can actually drop AdvEnc usage altogether and expand its definition directly in $\mathsf{BC.Enc}$. The result is that $\mathsf{BC.Enc}$ is simplified into the usual idealized block cipher encryption definition. Likewise, $\mathsf{BC.Dec}$ is also simplified but it is not yet the idealized block-cipher decryption (see next game). Note that for $\mathsf{BC.Dec}$, the check $r \notin \mathsf{usedR}_{pw}$ implies that there is no $\hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}$. As in the previous games, this is just a syntactic change and $P_8 = P_7$.

define $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(k, r)$
if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
  $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k, \hat{r}, \hat{s}) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return s

Fig. 13: Replacing usage of THIC by direct access to TBC

The current full game is given in Figure 14.

[12] i.e., we could invert such k by $k \mapsto (pw, T)$ where $(pw, T, k) \in \mathsf{TH}'$.

Game 9 *(real-world)*: In Figure 15 we present the real-world game between the environment and our $\mathsf{m2F}$ circuit. This game change consists of dropping the aborts and changing how r is picked in $\mathsf{BC.Dec}$ so as to make TBC consistent with the standard definition of an ideal-cipher. We refer the reader to Figures 14 and 15.

We start by removing the H' aborts. As before we can bound $|\mathsf{TH}'|$ and $|\mathsf{TBC}|$ by q so that these aborts happen with probability $\leq 2q/2^\mu$. H' now matches the real-world definition of Game 9. Then, we modify $\mathsf{BC.Dec}$. We drop the $r \in \mathsf{usedR}_{pw}$ abort in the innermost if and replace the "remove tag $\mathsf{m2F}$..." line with "add r to usedR_{pw}; if it is flagged $\mathsf{m2F}$, remove the flag". We can do so as long as this abort does not happen - i.e. except with probability $\leq |\mathsf{usedR}_{pw}|/2^n \leq q/2^n$. Finally, we compute r as is done in an ideal-cipher even when $k = \mathsf{TH}'(pw, T)$ just as in the real-world. This last change is valid except when our uniform choice of r in Game 8 collides with another $(k, r, \hat{s}) \in \mathsf{TBC}$. This gives us the bound $|\mathsf{TBC}|/2^n \leq q/2^n$. $\mathsf{BC.Dec}$ now matches the real-world in Figure 15 except that we have the usedR_{pw} line above.

Now, we are at the real-world game except that we have usedR_{pw} aborts in $\mathsf{m2F.Enc}$, H and $\mathsf{BC.Enc}$. The probability of the first one is trivially bound by another $|\mathsf{usedR}_{pw}|/2^n \leq q/2^n$ factor. The last two, much like in our argument for the change from Game 0 to Game 1, do not happen except when the adversary is lucky enough to completely guess r since r tagged $\mathsf{m2F}$ is never leaked to the environment. This gives us the following overall bound, which completes the proof:

$$|P_9 - P_8| \leq q^2 \left\{ \frac{2}{2^\mu} + \frac{4}{2^n} \right\} \tag{7}$$

B Game changes for Theorem 2

In this section we present the game changes used in our proof of Theorem 2. We refer the reader to Section 5 for the notation used and to Figure 9 for an algorithmic description of the games.

Game 0 (real-world game): This is the real world where parties follow the protocol. Technically, it is an *hybrid* world where the half-ideal cipher is replaced by functionality $\mathcal{F}_{\mathsf{HIC}}$, and the adversary can query $\mathcal{F}_{\mathsf{HIC}}$ through interfaces $\mathsf{Enc}, \mathsf{AdvEnc}, \mathsf{AdvDec}$.

Game 1 (randomizing protocol communication): We change the game so ciphertext c sent by $\mathsf{P}^{\mathsf{sid}}$ is purely random, except the game aborts if plaintext (r, M) occurred in table $\mathsf{THIC}_{\hat{pw}}$ or ciphertext c was output by any encryption. An upper bound on the probability of these aborts is given by

$$|P_0 - P_1| \leq q_P(q_{IC} + q_P) \left(\frac{1}{|\mathcal{R}|} + \frac{1}{|\mathcal{R}| \cdot |\mathcal{M}|} \right) \approx \frac{q_P(q_{IC} + q_P)}{|\mathcal{R}|} \tag{8}$$

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on query $\mathsf{m2F.Enc}(pw, M)$:

$r \xleftarrow{r} \{0,1\}^n$
if $r \in \mathsf{usedR}_{pw}$ abort
$T \leftarrow M/\mathsf{H}(pw, r)$
$k \leftarrow \mathsf{H}'(pw, T)$
$s \leftarrow \mathsf{BC.Enc}(k, r)$
assign tag $\mathsf{m2F}$ to r in the set usedR_{pw}
return (s, T)

on query $\mathsf{m2F.Dec}(pw, (s, T))$:

$k \leftarrow \mathsf{H}'(pw, T)$
$r \leftarrow \mathsf{BC.Dec}(k, s)$
$M \leftarrow \mathsf{H}(pw, r) \cdot T$
if $\mathsf{m2F.Dec}$ query was fresh, add tag $\mathsf{m2F}$ to $r \in \mathsf{usedR}_{pw}$
return M

on query $\mathsf{H}(pw, r)$

if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$, abort, else add r to usedR_{pw}
if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$:
  $h \xleftarrow{r} \mathbb{G}$
  add (pw, r, h) to TH
return h

on query $\mathsf{H}'(pw, T)$

if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
  $k \xleftarrow{r} \{0,1\}^\mu$
  if $\exists$ $(\hat{pw}, \hat{T})$ s.t. $(\hat{pw}, \hat{T}, k) \in \mathsf{TH}'$ then abort (col.abort)
  if $\exists$ $(\hat{r}, \hat{s})$ s.t. $(k, \hat{r}, \hat{s}) \in \mathsf{TBC}$ then abort (bckey.abort)
  add (pw, T, k) to TH'
return k

on query $\mathsf{BC.Enc}(k, r)$

if $k = \mathsf{TH}'(pw, T)$:
  if $r \in \mathsf{usedR}_{pw}$ is tagged $\mathsf{m2F}$, abort, else add $r \in \mathsf{usedR}_{pw}$
if $\nexists$ s s.t. $(k, r, s) \in \mathsf{TBC}$:
  $s \xleftarrow{r} \{s \in \{0,1\}^n : \nexists \hat{r}$ s.t. $(k, \hat{r}, s) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
return s

on query $\mathsf{BC.Dec}(k, s)$

if $\nexists$ r s.t. $(k, r, s) \in \mathsf{TBC}$:
  if $\exists$ (pw, T) s.t. $(pw, T, k) \in \mathsf{TH}'$:
    $r \xleftarrow{r} \{0,1\}^n$
    if $r \in \mathsf{usedR}_{pw}$ abort, else add r to it
  else:
    $r \xleftarrow{r} \{r \in \{0,1\}^n : \nexists \hat{s}$ s.t. $(k, r, \hat{s}) \in \mathsf{TBC}\}$
  add (k, r, s) to TBC
remove tag $\mathsf{m2F}$ from record $r \in \mathsf{usedR}_{pw}$ if $k = \mathsf{TH}'(pw, T)$
return r

Fig. 14: Full description of Game 8: one step away from the real-world

Initialization

Let TH be a set of tuples in $\{0,1\}^* \times \{0,1\}^n \times \mathbb{G}$,
TH' be a set of tuples in $\{0,1\}^* \times \mathbb{G} \times \{0,1\}^\mu$,
and TBC be a set of triples in $\{0,1\}^\mu \times \{0,1\}^n \times \{0,1\}^n$.

on query $\mathsf{m2F.Enc}(pw, M)$:

$r \xleftarrow{r} \{0,1\}^n$
$T \leftarrow M/\mathsf{H}(pw, r)$
$k \leftarrow \mathsf{H}'(pw, T)$
$s \leftarrow \mathsf{BC.Enc}(k, r)$
return (s, T)

on query $\mathsf{m2F.Dec}(pw, (s, T))$:

$k \leftarrow \mathsf{H}'(pw, T)$
$r \leftarrow \mathsf{BC.Dec}(k, s)$
$M \leftarrow \mathsf{H}(pw, r) \cdot T$
return M

on query $\mathsf{H}(pw, r)$

if $\nexists h$ s.t. $(pw, r, h) \in \mathsf{TH}$:
    $h \xleftarrow{r} \mathbb{G}$
    add (pw, r, h) to TH
return h

on query $\mathsf{H}'(pw, T)$

if $\nexists k$ s.t. $(pw, T, k) \in \mathsf{TH}'$:
    $k \xleftarrow{r} \{0,1\}^\mu$
    add (pw, T, k) to TH'
return k

on query $\mathsf{BC.Enc}(k, r)$

if $\nexists s$ s.t. $(k, r, s) \in \mathsf{TBC}$:
    $s \xleftarrow{r} \{\hat{s} \in \{0,1\}^n : \nexists \hat{r} \text{ s.t. } (k, \hat{r}, \hat{s}) \in \mathsf{TBC}\}$
    add (k, r, s) to TBC
return s

on query $\mathsf{BC.Dec}(k, s)$

if $\nexists r$ s.t. $(k, r, s) \in \mathsf{TBC}$:
    $r \xleftarrow{r} \{\hat{r} \in \{0,1\}^n : \nexists \hat{s} \text{ s.t. } (k, \hat{r}, \hat{s}) \in \mathsf{TBC}\}$
    add (k, r, s) to TBC
return r

Fig. 15: Game 9: the real-world interaction between $\mathcal{Z}$ and $\mathsf{m2F}$

Game 2 (binding adversarial ciphertexts to passwords): We add two changes in the processing of both Enc and AdvEnc queries for any key $\hat{pw}$: If the game responds to either query by picking a new ciphertext c, it (1) aborts if this ciphertext is already in set Cset, (2) otherwise it proceeds but also sets $\mathsf{c2pw}[c] \leftarrow \hat{pw}$. (Initially $\mathsf{c2pw}[c] = \perp$ for all inputs.) The second change is purely syntactic, but the first one introduces a difference upper-bounded by the probability of encountering such collisions. Since Enc picks ciphertext c at random in the space of c's not used for a given key $\hat{pw}$, while AdvEnc picks only the s part of the ciphertext c at random from the space of unused s for a given T and key $\hat{pw}$, the upper-bound on these collisions comes from AdvEnc queries which implies

$$|P_1 - P_2| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}|} \tag{9}$$

Game 3 (embedding KA messages in decryption queries): In this game we embed KA messages into every (fresh) adversarial decryption query $\mathsf{AdvDec}(\hat{pw}, c)$ to $\mathcal{F}_{\mathsf{HIC}}$, and we save the local state generated with this KA message associated with $(c, \hat{pw})$. This change can be thought of as done in two sub-steps: First we change the decryption so it picks (r, M) at random in $\mathcal{R} \times \mathcal{G}$ and aborts if $((r, M), *)$ is in table THIC. (Note that before a decryption query picks (r, M) according to $\mathsf{THIC}_{\hat{pw}}.\mathsf{m}$, i.e. among pairs which are not yet in the table.) The difference this introduces is the probability of encountering this abort, which can be upper-bounded as $(q_{IC}+q_P)^2/(|\mathcal{R}|\cdot|\mathcal{M}|)$. The second sub-step is that we pick M according to KA message generation algorithm $\mathsf{KA.msg}$, and we save local state x generated together with M in record $(\mathsf{backdoor}, c, \hat{pw}, x)$. This second change can be reduced to an attack on the random-message property of scheme KA. The argument hybridizes over all decryption queries, where each consecutive hybrid differs by one more decryption query on which M is generated via $\mathsf{KA.msg}$ instead of uniform in $\mathcal{G}$. By a reduction to the random-message property of our scheme KA the total difference this change introduces can be upper-bound as $(q_{IC}+q_P)\cdot\varepsilon_{\mathsf{KA.rand}}$. We conclude that:

$$|P_2 - P_3| \leq \frac{(q_{IC}+q_P)^2}{|\mathcal{R}|\cdot|\mathcal{M}|} + (q_{IC}+q_P)\cdot\varepsilon_{\mathsf{KA.rand}} \tag{10}$$

Game 4 (delegating KA message generation to AdvDec): We make a syntactic change in processing $\mathsf{NewSession}$: Rather than picking a random KA message M, random r, a random ciphertext c, and defining $((r, M), c)$ as an IC pair for key $\hat{pw}$, we pick only random c and define M via a decryption query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$. Since in Game 3 a decryption query sets (r, M) in the same way this is only a syntactic change, hence $P_3 = P_4$.

Game 5 (extracting passwords and randomizing session keys on "wrong" passwords): We change how $\mathsf{P}^{\mathsf{sid}}$ reacts to a received ciphertext $\hat{c}$. First of all we introduce a special processing in case $\hat{c}$ is sent by a matching session $\mathsf{CP}^{\mathsf{sid}}$ (i.e. a session that uses the same sid, same pw, and matching P, CP values) that received the honest message c sent out by $\mathsf{P}^{\mathsf{sid}}$. In this case we short-cut all processing and simply set the session key output by $\mathsf{P}^{\mathsf{sid}}$ to the one which was output by $\mathsf{CP}^{\mathsf{sid}}$. Note this corresponds to the case where the environment does not interfere in the communication between $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$. This introduces no change because such sessions compute the same session keys in all previous games. Secondly, for all other $\hat{c}$ cases, instead of using $\mathcal{F}_{\mathsf{HIC}}$ to decrypt $\hat{c}$ under stored $\hat{pw}$, and using the decrypted plaintext $\hat{M}$ to compute the session key as $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, we set $\hat{pw} = \mathsf{c2pw}[\hat{c}]$ and consider two cases: If $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$ then Game 5 computes K in the same way as in Game 4, except that we render the decryption query as a retrieval from table $\mathsf{THIC}_{\hat{pw}}$ instead of as a query to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$, but this is only a notational change; In any other case, Game 5 shortcuts this decryption and key-computation process and outputs a random key $K \xleftarrow{r} \{0,1\}^\kappa$.

The argument that Game 4 is indistinguishable from Game 5 is a hybrid argument which changes the view in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ invoked by $\mathcal{Z}$. Note that the only case where there is a difference between the two games is the last one we described, i.e. if $\mathsf{c2pw}[\hat{c}]$ contains an entry $\hat{pw} \neq (\mathsf{fullsid}, \neg b, pw)$.[13] This corresponds to two sub-cases: (a) $\hat{c}$ was created via an adversarial encryption query on some key $\hat{pw}$ which does not match the decryption key $(\mathsf{fullsid}, \neg b, pw)$ that $\mathsf{P}^{\mathsf{sid}}$ would use in Game 4 to decrypt this ciphertext (note that this $\hat{pw}$ is unique because of an abort in the case two

[13] This includes the case of $\hat{pw} = \perp$.

encryption queries ever create the same ciphertext); and (b) $\hat{c}$ was not created in any encryption query. In either of these two sub-cases Game 4 would compute $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ for $\hat{M} \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$, and since in either case $\hat{c}$ was not inserted in table $\mathsf{THIC}_{(\mathsf{fullsid}, \neg b, pw)}$ via an encryption query, this AdvDec query will embed a random KA message into the decrypted plaintext $\hat{M}$.

We will argue that the existence of an adversary who distinguishes with non-negligible advantage between Games 4 and 5 implies an attack on the security property of the key exchange scheme KA. Since message M created by $\mathsf{P}^{\mathsf{sid}}$ is a random KA message, and we argued above that $\hat{M} = \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, \neg b, pw), \hat{c})$ is a random KA message as well, the session key K which $\mathsf{P}^{\mathsf{sid}}$ outputs in this case in Game 4 is a KA output on an exchange involving two random KA messages, M and $\hat{M}$. It can thus be replaced by a random string by a reduction to KA security done separately for each sid, in two sub-steps corresponding to the (at most) two sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ which run on this particular sid. Consider the argument for a fixed session $\mathsf{P}^{\mathsf{sid}}$ with corresponding counterparty session $\mathsf{CP}^{\mathsf{sid}}$ and $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$: Given the KA security challenge $(M, \hat{M}, K)$, the reduction does the following: First, it uses challenge value M when computing the outgoing message c of $\mathsf{P}^{\mathsf{sid}}$, i.e. it uses M from the challenge when processing query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ in Game 4. Second, it guesses an index $i \xleftarrow{r} [1, ..., q_{IC} + q_P]$ of a query to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$ using key $(\mathsf{fullsid}, \neg b, pw)$ and embeds challenge value $\hat{M}$ into the decrypted plaintext. (Note that by Game 4 each $\mathsf{NewSession}$ query also uses AdvDec. Notice also that $\hat{c}$ in this AdvDec query could be equal to ciphertext c' generated by session $\mathsf{CP}^{\mathsf{sid}}$, this corresponds to the adversary passively connecting two sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ which run on matching inputs). Third, if the guess is right and the adversary sends ciphertext $\hat{c}$ used in this i-th query to $\mathsf{P}^{\mathsf{sid}}$, the reduction embeds the K challenge value into the session key output by $\mathsf{P}^{\mathsf{sid}}$ (if the guess is not right the reduction aborts). If the guess is right and case (b) occurs, the reduction reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 4 if K is the real key corresponding to KA instance $(M, \hat{M})$, and it reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 5 if K is random. Since the right guess occurs with probability $1/(q_{IC} + q_P)$ and the identity of the index i does not affect the view the reduction produces before the abort, and the argument goes by a hybrid over all honest party sessions, we arrive at the following upper-bound:

$$|P_4 - P_5| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}| \cdot |\mathcal{M}|} + q_P(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KA.sec}} \tag{11}$$

GAME 6 (delaying password usage): In this game we delay using the password pw of session $\mathsf{P}^{\mathsf{sid}}$ to decrypt (and consequently embed the backdoor into) its honest outgoing message c to the moment $\mathsf{P}^{\mathsf{sid}}$ receives an incoming message $\hat{c}$. Moreover, we perform this decryption only in the case adversary created $\hat{c}$ via encryption under key $(\mathsf{fullsid}, \neg b, pw)$. Since Game 5 does not use the decrypted value M and the associated trapdoor x until this exact situation occurs, postponing this decryption does not matter as long as item $(*, c)$ is not written into table $\mathsf{THIC}_{(\mathsf{fullsid},b,pw)}$ via an encryption query. However, the latter cannot happen in Game 5 because each NewSession and each encryption queries generate disjoint ciphertexts (a collision in the ciphertexts created by any of these queries leads to an abort), which implies that $P_5 = P_6$.

GAME 7 (ideal-world game implied by $\mathcal{F}_{\mathsf{pwKE}}$ and SIM): This is the ideal-world game induced by functionality $\mathcal{F}_{\mathsf{pwKE}}$ interacting with simulator SIM of Figure 8. In that interaction $\mathcal{F}_{\mathsf{pwKE}}$ creates a session record with password pw in it, but $\mathcal{F}_{\mathsf{pwKE}}$ does not pass pw to SIM. However, SIM picks $\mathsf{P}^{\mathsf{sid}}$'s c at random and aborts if $c \in \mathsf{Cset}$, which is how NewSession processing is done in Game 6. Note also that SIM replies to Enc or AdvEnc queries in a way which matches processing of these queries starting from Game 2, and that it replies to AdvDec queries in a way which matches processing of these queries starting from Game 3. Finally, when the environment sends $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$, we have the following cases:

1. Message $\hat{c}$ was sent by counterparty session $\mathsf{CP}^{\mathsf{sid}}$ which matches session $\mathsf{P}^{\mathsf{sid}}$ in session identifier sid and party identifiers $(\mathsf{P}, \mathsf{CP})$. This case is detected by the simulator SIM who can check if identifiers $(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ of the two sessions match, and it corresponds to step 1 in SIM's processing of $\hat{c}$. In this case SIM sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$ in which case $\mathcal{F}_{\mathsf{pwKE}}$, since this NewKey was *not* proceeded by a TestPwd so session $\mathsf{P}^{\mathsf{sid}}$ is marked fresh, does either of the following two things: (case 1) if the two sessions run on the same password and $\mathsf{CP}^{\mathsf{sid}}$ completed while marked fresh, which happens only if the adversary sent (to $\mathsf{CP}^{\mathsf{sid}}$) the unmodified ciphertext c output by $\mathsf{P}^{\mathsf{sid}}$, then $\mathcal{F}_{\mathsf{pwKE}}$ makes key K output by $\mathsf{P}^{\mathsf{sid}}$ equal to key $\hat{K}$ output by $\mathsf{CP}^{\mathsf{sid}}$; and (case 2) in any other case $\mathcal{F}_{\mathsf{pwKE}}$ picks key K output by $\mathsf{P}^{\mathsf{sid}}$ at random.
   Note that this is exactly how Game 6 processes delivery of $\hat{c}$ output by $\mathsf{CP}^{\mathsf{sid}}$ as well. Case 1 corresponds to the first check performed by $\hat{c}$-delivery processing code of Game 6 which assigns $K \leftarrow \hat{K}$ if all inputs of $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ match and the adversary delivered the ciphertext output by $\mathsf{P}^{\mathsf{sid}}$ to $\mathsf{CP}^{\mathsf{sid}}$. Case 2 means that the $\hat{c}$-delivery processing code of Game 6 will recover $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and check if $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$. In case 2, where $\hat{c}$ is output by $\mathsf{CP}^{\mathsf{sid}}$ value $\mathsf{c2pw}[\hat{c}]$ is guaranteed to be $\perp$ because Game 6, just like the ideal-world interaction, does not allow collisions between ciphertexts output by honest sessions and ciphertexts output via Enc or AdvEnc queries. Therefore $\hat{c}$-delivery processing code of Game 6 will jump to the second "else" clause and set $K \xleftarrow{r} \{0,1\}^{\kappa}$, matching the behavior of the ideal-world interaction.
2. If message $\hat{c}$ was *not* sent by counterparty session $\mathsf{CP}^{\mathsf{sid}}$ which matches session $\mathsf{P}^{\mathsf{sid}}$ in its session+party identifiers inputs, i.e. if $\hat{c}$ is a ciphertext created by the adversary (or output by any other session than the intended counterparty of $\mathsf{P}^{\mathsf{sid}}$), this corresponds to 2. in SIM's definition of its processing of $\hat{c}$, which has two sub-cases based on the value $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$:
   (a) In (a) SIM processes the case when $\hat{pw} = \perp$ or $\hat{pw} \neq \perp$ but $\hat{pw}$ does not have the form $(\mathsf{fullsid}, \neg b, pw^*)$ for any password pw^* (which means that $\hat{pw}$ is guaranteed not to match the key $\mathsf{P}^{\mathsf{sid}}$ would use to decrypt $\hat{c}$ regardless of the password $\mathsf{P}^{\mathsf{sid}}$ uses). In that case SIM sends

$(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$ before sending $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$, which means that $\mathcal{F}_{\mathsf{pwKE}}$ marks this session as $\mathsf{interrupted}$ and sets its key as $K \xleftarrow{r} \{0,1\}^{\kappa}$.

Observe that in this case Game 6 will set K in the same way as in the above SIM+$\mathcal{F}_{\mathsf{pwKE}}$ interaction, because $\hat{pw} \neq (\mathsf{fullsid}, \neg b, pw^*)$ for any pw^* including pw held by $\mathsf{P}^{\mathsf{sid}}$, so the $\hat{c}$-delivery processing code of Game 6 will go to the second "else" clause and set $K \xleftarrow{r} \{0,1\}^{\kappa}$.

(b) In (b) SIM processes the case when $\hat{pw} = (\mathsf{fullsid}, \neg b, pw^*)$ for some pw^*, which might or might not be equal to the password input pw of $\mathsf{P}^{\mathsf{sid}}$. In this case SIM retrieves $((\hat{r}, \hat{M}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$, services query $(\mathsf{AdvDec}, (\mathsf{fullsid}, b, pw^*), c)$, retrieves $(\mathsf{backdoor}, c, (\mathsf{fullsid}, b, pw^*), x)$, sets $K \leftarrow \mathsf{KA.key}(x, \hat{M})$, and sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$. Consider two sub-cases depending on $\mathsf{P}^{\mathsf{sid}}$'s input pw:

i. If $pw^* \neq pw$ then $\mathcal{F}_{\mathsf{pwKE}}$ will mark session $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{interrupted}$ in response to the above $\mathsf{TestPwd}$ query, and consequently $\mathcal{F}_{\mathsf{pwKE}}$ will ignore the value K which SIM sends in the NewKey query, and it will pick the session key output by $\mathsf{P}^{\mathsf{sid}}$ uniformly from $\{0,1\}^{\kappa}$.

   This is also how Game 6 $\hat{c}$-delivery code will process this case, because it corresponds to the case when $\hat{pw}$ retrieved from $\mathsf{c2pw}[\hat{c}]$ is not equal to $(\mathsf{fullsid}, \neg b, pw)$.

ii. If $pw^* = pw$ then $\mathcal{F}_{\mathsf{pwKE}}$ will mark session $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{compromised}$ in response to $\mathsf{TestPwd}$ and in response to NewKey it will make $\mathsf{P}^{\mathsf{sid}}$ output the key K computed by SIM.

   This is also how Game 6 will behave in this case, because it corresponds to the case $\hat{pw} = (\mathsf{fullsid}, \neg b, pw)$, in which case Game 6 retrieves backdoor x as the KA state corresponding to the decryption of c under key $(\mathsf{fullsid}, b, pw)$, and it sets $\mathsf{P}^{\mathsf{sid}}$'s output as $K \leftarrow \mathsf{KA.key}(x, \hat{M})$ for $((\hat{r}, \hat{M}), \hat{c})$ retrieved from $\mathsf{THIC}_{\hat{pw}}$, exactly like SIM does above.

Since Game 6 matches the ideal-world interaction of Game 7 exactly we conclude that $P_6 = P_7$, which completes the proof of Theorem 2.

C Security Proof for protocol **EKE-KEM** of Section 5.1

In this section we prove Theorem 3 from Section 5.1, i.e. the $\mathsf{EKE\text{-}KEM}$ protocol shown in Figure 10 is a UC PAKE. Following the blueprint of EKE proof in Section 5, we argue security of $\mathsf{EKE\text{-}KEM}$ in the $\mathcal{F}_{\mathsf{HIC}}$-hybrid model. We adopt definitions of q_{IC}, q_P in EKE proof, and additionally let $\varepsilon_{\mathsf{KEM.sec}}$ be the upper-bound on the distinguishing advantage against the security property of KEM, let $\varepsilon_{\mathsf{KEM.randpk}}$ and $\varepsilon_{\mathsf{KEM.anonymity}}$ be the upper-bounds on the distinguishing advantage against the uniform public keys and anonymity properties of KEM, respectively. Also we let $\varepsilon_{\mathsf{prf}}$ be the upper-bound of distinguishing advantage against pseudorandomness of prf. We define notation

$c[pk^*]$ as a shortcut for generating e from $(e, K^*) \leftarrow \mathsf{KEM.enc}(pk^*)$, where pk^* is a random public key.[14]

Proof. GAME 0 (real-world game): This is the real world constructed by parties following the protocol, functionality $\mathcal{F}_{\mathsf{HIC}}$, and the adversary who can query $\mathcal{F}_{\mathsf{HIC}}$ through interfaces $\mathsf{Enc}, \mathsf{AdvEnc}, \mathsf{AdvDec}$. We also record all generated ciphertexts in set Cset which is syntactic.

GAME 1 (randomizing first message): We change the ciphertext c sent by $\mathsf{P}^{\mathsf{sid}}$ to be purely random, and abort if the corresponding plaintext (r, M) occurred in table $\mathsf{THIC}_{\hat{pw}}$ or ciphertext c was output by any encryption. As in the case of EKE proof:

$$|P_0 - P_1| \leq \frac{q_P(q_{IC} + q_P)}{|\mathcal{R}|} \tag{12}$$

GAME 2 (binding adversarial ciphertexts to passwords): We add a change in any processing of ciphertext generation, where we abort if any new generation is already in Cset. This includes ciphertexts generated (for any key $\hat{pw}$) by Enc, AdvEnc queries, and by $\mathsf{P}^{\mathsf{sid}}$ mentioned in Game 1. We also add a syntactic change where we record $\mathsf{c2pw}[c] \leftarrow \hat{pw}$ in Enc or AdvEnc queries. As in EKE proof we have:

$$|P_1 - P_2| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}|} \tag{13}$$

GAME 3 (embedding public key in decryption queries): In this game we embed public key into every fresh adversarial $\mathsf{AdvDec}(\hat{pw}, c)$ query to $\mathcal{F}_{\mathsf{HIC}}$, and we save the corresponding sk associated with $(c, \hat{pw})$. This change can be done in two sub-steps: First we change the decryption so it picks (r, pk) randomly in $\mathcal{R} \times \mathcal{PK}$, and aborts if $((r, pk), *)$ is in table $\mathsf{THIC}_{\hat{pw}}$, whereas before, a decryption query picks (r, pk) according to $\mathsf{THIC}_{\hat{pw}}.\mathsf{m}$, i.e. among pairs which are not yet in the table. The probability of encountering this abort can be upper-bounded by $(q_{IC} + q_P)^2/(|\mathcal{R}| \cdot |\mathcal{PK}|)$. The second sub-step is that we generate key pair (sk, pk) according to KEM key generation algorithm kg, and we save sk in record $(\mathsf{backdoor}, c, \hat{pw}, sk)$. This second change can be reduced to an attack on the uniform public keys property 4 of KEM. The argument hybridizes over all decryption queries, where each consecutive hybrid differs by one more decryption query on which pk is generated via kg instead of uniform in $\mathcal{PK}$. By a reduction to the uniform public keys property of KEM the total difference this change introduces can be upper-bound as $(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.randpk}}$. We conclude that:

$$|P_2 - P_3| \leq \frac{(q_{IC} + q_P)^2}{|\mathcal{R}| \cdot |\mathcal{PK}|} + (q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.randpk}} \tag{14}$$

[14] The proof below assumes a version of the protocol which uses $\mathsf{prf}(K, \cdot)$ to derive the authenticator τ and the session key, *and* the authenticator computation does not take the KEM public key pk as an input. This version of the protocol requires an additional assumption on KEM. We will update the proof shortly to reflect the modified protocol and get rid of the additional assumption.

SIM interacts with environment $\mathcal{Z}$'s "adversary" interface $\mathcal{A}$, and with PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$. Let $\mathcal{PK}$ be the public key space and $\mathcal{C}$ be the ciphertext space of KEM, and let $\mathcal{D} = \mathcal{R} \times \mathcal{PK}$ be the domain of Half-Ideal Cipher HIC. Let prf be a pseudorandom function. Without loss of generality we assume that $\mathcal{A}$ uses AdvDec interface to implement a Dec query to $\mathcal{F}_{\mathsf{HIC}}$.

Initialization: Setup $\mathsf{Cset} = \{\}$, set $\mathsf{THIC}_{\hat{pw}}$ and $\mathsf{T_H}$ as empty table, and $\mathsf{c2pw}[c] := \perp$ for all $\mathsf{fullsid}, pw, c$. Pick a random $pk^* \xleftarrow{r} \mathcal{PK}$.

Notation:

Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$ be a shortcut for set $\{s \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (s, T)) \in \mathsf{THIC}_{\hat{pw}}\}$.
Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{c}$ be a shortcut for set $\{c \in \mathcal{D} : \nexists \hat{m} \text{ s.t. } (\hat{m}, c) \in \mathsf{THIC}_{\hat{pw}}\}$.
Let $c[pk^*]$ be a shortcut for generating e from $(e, K^*) \leftarrow \mathsf{KEM.enc}(pk^*)$.

On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}_{\mathsf{pwKE}}$:
Set $\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$.

1. If $b = 0$, pick $c \xleftarrow{r} \mathcal{D}$($abort$ if $c \in \mathsf{Cset}$), add c to Cset, send c to $\mathcal{A}$ as a message from $\mathsf{P}^{\mathsf{sid}}$ and record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 0, \mathsf{fullsid}, c)$.
2. If $b = 1$, record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \perp, \perp, \perp)$.

Emulating $\mathcal{F}_{\mathsf{HIC}}$ (for $\mathsf{fullsid} = \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ for any P, CP)

- On $\mathcal{A}$'s query $(\mathsf{Enc}, \hat{pw}, M)$ to $\mathcal{F}_{\mathsf{HIC}}$: Set $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, M)$. If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return c; Else pick $c \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{c}$($abort$ if $c \in \mathsf{Cset}$), set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvEnc}, \hat{pw}, m, T)$ to $\mathcal{F}_{\mathsf{HIC}}$: If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return c; Else pick $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$($abort$ if $c \in \mathsf{Cset}$), $\mathsf{c2pw}[c] \leftarrow \hat{pw}$, add c to Cset and (m, c) to $\mathsf{THIC}_{\hat{pw}}$, return c.
- On $\mathcal{A}$'s query $(\mathsf{AdvDec}, \hat{pw}, c)$ to $\mathcal{F}_{\mathsf{HIC}}$: If $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return m; Else pick $r \xleftarrow{r} \mathcal{R}$ and $(pk, sk) \xleftarrow{r} \mathsf{KEM.kg}$, set $m \leftarrow (r, pk)$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$ ($abort$ if $\exists\ \hat{c} \neq c$ s.t. $(m, \hat{c}) \in \mathsf{THIC}_{\hat{pw}}$), save $(\mathsf{backdoor}, c, \hat{pw}, sk)$, return m.

On $\mathcal{A}$'s message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only one such message)
Retrieve record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \perp, \perp, \perp)$ and:

- If there is record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, 0, \mathsf{fullsid}, \hat{c})$ then set $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$. Update record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \hat{c}, e, \tau)$ and send (e, τ) to $\mathcal{A}$. Send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$.
- Otherwise set $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$ and do the following:
  1. If $\hat{pw} = \perp$ or $\hat{pw} = (\mathsf{fullsid}', \cdot)$ for $\mathsf{fullsid}' \neq \mathsf{fullsid}$: send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$; pick $e \leftarrow c[pk^*], \tau \xleftarrow{r} \{0,1\}^\kappa$ and send (e, τ) to $\mathcal{A}$
  2. If $\hat{pw} = (\mathsf{fullsid}, pw^*)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and:
     (a) if answer is "incorrect" then set $e \xleftarrow{r} c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, send (e, τ) to $\mathcal{A}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$
     (b) if answer is "correct" then service $\mathcal{F}_{\mathsf{HIC}}$'s query $(\mathsf{AdvDec}, \hat{pw}, \hat{c})$, retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$, set $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk}), \tau \leftarrow \mathsf{prf}(K^*, 1)$, send (e, τ) to $\mathcal{A}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \mathsf{prf}(K^*, 2))$ to $\mathcal{F}_{\mathsf{pwKE}}$

  Update record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 1, \mathsf{fullsid}, \hat{c}, e, \tau)$

On $\mathcal{A}$'s message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$: (accept only one such message)
Retrieve record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, 0, \mathsf{fullsid}, c)$ and:

- If there is record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, 1, \mathsf{fullsid}, c, \hat{e}, \hat{\tau})$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$.
- Else if $\exists\ pw^*$ s.t. $\exists(\mathsf{backdoor}, c, (\mathsf{fullsid}, pw^*), sk)$ and $\hat{\tau} = \mathsf{prf}(K^*, 1)$ for $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$(abort if find multiple pw^* satisfy above), send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \mathsf{prf}(K^*, 2))$ to $\mathcal{F}_{\mathsf{pwKE}}$
- Otherwise send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, \perp)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$

On H query $(\mathsf{fullsid}, K, pk)$:
if $\exists \langle(\mathsf{fullsid}, K, pk), \tau\rangle$ in $\mathsf{T_H}$ then output τ
else pick $\tau \xleftarrow{r} \{0,1\}^\kappa$, add $\langle(\mathsf{fullsid}, K, pk), \tau\rangle$ to $\mathsf{T_H}$, and output τ

Fig. 16: Simulator SIM for the proof of Theorem 3, Section 5.1

Game 4 (delegating key generation to $\mathsf{IC.Dec}$): We make a syntactic change in processing $\mathsf{NewSession}$ for the party who sends out message c: Rather than generating a random key pair (sk, pk), random r, a random ciphertext c, and defining $((r, pk), c)$ as an IC pair for key $\hat{pw}$, we pick only random c and define (sk, pk) via a decryption query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$. Since in Game 3 a decryption query sets (r, pk) in the same way, this is only a syntactic change, hence $P_3 = P_4$.

Game 5 (randomizing second message and session keys in passive cases):

As in the EKE proof, we change how honest parties react to received messages. First we change the passive case, where $\mathsf{P}^{\mathsf{sid}}$ receives the ciphertext $\hat{c}$ sent by a matching session $\mathsf{CP}^{\mathsf{sid}}$. In this case we shortcut all processing and simply let $\mathsf{P}^{\mathsf{sid}}$ output e as a random element in its space, and output τ and session key K as random elements in $\{0, 1\}^{\kappa}$. Furthermore, if $\mathsf{CP}^{\mathsf{sid}}$ receives this (e, τ) then we shortcut and set the session key of $\mathsf{CP}^{\mathsf{sid}}$ to K output by $\mathsf{P}^{\mathsf{sid}}$.

Recall that Game 4 would compute pk by querying $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), c)$, and then compute $(e, K^*) \leftarrow \mathsf{enc}(pk)$, $\tau \leftarrow \mathsf{prf}(K^*, 1)$ and $K \leftarrow \mathsf{prf}(K^*, 2)$. The change in passive cases can be done in 3 substeps: Firstly we randomize K^* from $\mathsf{KEM.enc}(pk)$ output, and further generate τ and session key by running prf on this random K^*. Meanwhile if $\mathsf{CP}^{\mathsf{sid}}$ receives this (e, τ) passively passed by $\mathcal{A}$, we shortcut KEM decapsulation and set K^* as the one $\mathsf{P}^{\mathsf{sid}}$ generates, the other parts are exact same as Game 4. We argue that an adversary who distinguishes this change with non-negligible advantage implies an attack on the security property of the KEM scheme. Consider the argument for session $\mathsf{P}^{\mathsf{sid}}$, with the other session is denoted $\mathsf{CP}^{\mathsf{sid}}$: Given the KEM security challenge (pk^*, e^*, K^*), the reduction does the following: it guesses an index $i \xleftarrow{r} [1, ..., q_{IC} + q_P]$ of a query to $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$ using key $(\mathsf{fullsid}, pw)$ and embeds challenge value pk^* into the decrypted plaintext.

Note that by Game 4 each $\mathsf{NewSession}$ query also uses AdvDec and $\hat{c}$ in this AdvDec query could be equal to ciphertext c generated by session $\mathsf{CP}^{\mathsf{sid}}$, which corresponds to the adversary passively connecting two sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ which run on matching inputs. If the guess is right and the adversary sends ciphertext $\hat{c}$ used in this i-th query to $\mathsf{P}^{\mathsf{sid}}$, the reduction embeds the (e^*, K^*) challenge value into the $\mathsf{KEM.enc}$ output by $\mathsf{P}^{\mathsf{sid}}$. (If the guess is not right the reduction aborts.) If the guess is right, the reduction reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 4 if K^* is the real key corresponding to KEM instance (pk^*, e^*, K^*), and it reproduces how $\mathsf{P}^{\mathsf{sid}}$ acts in Game 5 if K^* is random. Since the right guess occurs with probability $1/(q_{IC} + q_P)$ and the identity of the index i does not affect the view the reduction produces before the abort, and the argument goes by a hybrid over all honest party sessions, this change is upper-bounded by $q_P/(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.sec}}$.

Secondly we remove the usage of H and prf and directly generate τ, K as random bitstrings. The change on session keys introduces no difference because such sessions compute same session keys in all previous games. The change on τ can be reduced to the security of prf and is bounded by $q_P \cdot \varepsilon_{\mathsf{prf}}$.

Thirdly, we change e to be generated via a KEM encapsulation on a random public key, instead of via the public key output by decryption query. This change can be reduced to an attack on the anonymity property 5 of KEM. The argument is hybrid and changes in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ who received this passive $\hat{c}$. Each consecutive differs by one more e generation, where e is picked via $\mathsf{KEM.enc}(pk^*)$ for a random pk^* picked by SIM, instead of generated via $\mathsf{KEM.enc}(pk)$ for $pk \leftarrow \mathcal{F}_{\mathsf{HIC}}(\hat{pw}, c)$ where $\hat{pw}$ refers to the pw which $\mathsf{P}^{\mathsf{sid}}$ holds. By a reduction to the anonymity property of KEM, the total difference introduced by this change can be upper-bounded as $(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.anonymity}}$.

Game 0: real-world interaction

Initialize $\mathsf{Cset} = \{\}$ and empty table $\mathsf{THIC}_{\hat{pw}}$ for all $\hat{pw}$;
on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:

$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$
if $b = 0$: $(sk, pk) \xleftarrow{r} \mathsf{kg}$, $c \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, pk)$
  save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, c, \perp)$
  $\boxed{\text{output } c}$
if $b = 1$: save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \perp, \perp, \perp, \perp)$

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one):
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \perp, \perp, \perp, \perp)$:
  $(\hat{r}, \hat{pk}) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$
  $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{prf}(K^*, 1)$
  $K \leftarrow \mathsf{prf}(K^*, 2)$
  reset $\mathsf{Rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$
  $\boxed{\text{output } (e, \tau) \text{ and } (\mathsf{sid}, \mathsf{P}, K)}$

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$ (accept only one):
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, c, \perp)$:
  $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$
    if $\tau = \mathsf{prf}(K^*, 1)$ then set $K' \leftarrow \mathsf{prf}(K^*, 2)$ and $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K')}$
  else $\boxed{\text{output } K' \xleftarrow{r} \{0,1\}^\kappa}$

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, M)$ (assuming $M \in \mathcal{PK}$):
$r \xleftarrow{r} \mathcal{R}$, set $m \leftarrow (r, M)$
If $\exists$ c s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return c
else:
  pick $c \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{c}$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$ and c to Cset
  return c

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$: (assuming $m \in \mathcal{R} \times \mathcal{PK}$ and $T \in \mathcal{PK}$):
if $\exists$ c s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return c
else:
  $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, set $c \leftarrow (s, T)$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$ and c to Cset
  return c

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$ (assuming $c \in \mathcal{R} \times \mathcal{PK}$):
if $\exists$ m s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$:
  return m
else:
  $m \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{m}$, add (m, c) to $\mathsf{THIC}_{\hat{pw}}$
return m

Game 1: randomizing first message

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$
if $b = 0$:
  set $(sk, pk) \xleftarrow{r} \mathsf{kg}$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, pk)$, $c \xleftarrow{r} \mathcal{D}$
  abort if $(m, *) \in \mathsf{THIC}_{\hat{pw}}$ or $c \in \mathsf{Cset}$
  add (m, c) to $\mathsf{THIC}_{\hat{pw}}$
  save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, c, \perp)$
  $\boxed{\text{output } c}$

Game 2: binding adversarial ciphertexts to passwords

on $\mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}(\hat{pw}, M)$ or $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}(\hat{pw}, m, T)$:
Before either process adds c to Cset, do the following:
  abort if $c \in \mathsf{Cset}$
  set $\mathsf{c2pw}[c] \leftarrow \hat{pw}$

Game 3: embedding public key in decryption queries

on query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$:

if $\exists m$ s.t. $(m, c) \in \mathsf{THIC}_{\hat{pw}}$ return m, otherwise:
  set $(sk, pk) \xleftarrow{r} \mathsf{kg}$, $r \xleftarrow{r} \mathcal{R}$, $m \leftarrow (r, pk)$
  abort if $(m, *) \in \mathsf{THIC}_{\hat{pw}}$
  add (m, c) to $\mathsf{THIC}_{\hat{pw}}$, save $(\mathsf{backdoor}, c, \hat{pw}, sk)$, return (r, pk)

Game 4: delegating key generation to IC.Dec

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$, $\hat{pw} \leftarrow (\mathsf{fullsid}, pw)$
if $b = 0$: $c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
  query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}(\hat{pw}, c)$, retrieve $(\mathsf{backdoor}, c, \hat{pw}, sk)$
  save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, sk, c, \perp)$ and $\boxed{\text{output } c}$

Game 5: randomizing second message and session keys

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $\mathsf{rec} = (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \perp, \perp, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 0, pw, sk, \hat{c}, \perp)$:
    pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \leftarrow \{0,1\}^\kappa$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, pw)$ then retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$ and
    set $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{prf}(K^*, 1)$, $K \leftarrow \mathsf{prf}(K^*, 2)$
    else:
      pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$
  $\boxed{\text{output } (e, \tau), (\mathsf{sid}, \mathsf{P}, K)}$

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, sk, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 1, pw, c, \hat{e}, \hat{\tau}, K)$ then set $K' \leftarrow K$,
  $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K')}$
  else:
    set $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$
      if $\hat{\tau} = \mathsf{prf}(K^*, 1)$ then set $K' \leftarrow \mathsf{prf}(K^*, 2)$ and $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K')}$
    otherwise $\boxed{\text{output } K' \xleftarrow{r} \{0,1\}^\kappa}$

Game 6: delaying password usage

on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pw)$ to P:
$\mathsf{fullsid} \leftarrow \mathsf{order}(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$, $b \leftarrow \mathsf{bit}(\mathsf{P}, \mathsf{CP})$
if $b = 0$: $c \xleftarrow{r} \mathcal{D}$, *abort* if $c \in \mathsf{Cset}$, otherwise add c to Cset
  save $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, b, pw, \perp, c, \perp)$ and $\boxed{\text{output } c}$

on message $\hat{c}$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \perp, \perp, \perp, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 0, pw, \perp, \hat{c}, \perp)$:
    pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$
  else:
    $\hat{pw} \leftarrow \mathsf{c2pw}[\hat{c}]$
    if $\hat{pw} = (\mathsf{fullsid}, pw)$:
      query $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$, retrieve $((\hat{r}, \hat{pk}), \hat{c})$ from $\mathsf{THIC}_{\hat{pw}}$
      set $(e, K^*) \leftarrow \mathsf{KEM.enc}(\hat{pk})$, $\tau \leftarrow \mathsf{prf}(K^*, 1)$, $K \leftarrow \mathsf{prf}(K^*, 2)$
    else:
      pick $e \leftarrow c[pk^*]$, $\tau \xleftarrow{r} \{0,1\}^\kappa$, $K \xleftarrow{r} \{0,1\}^\kappa$
  reset $\mathsf{rec} \leftarrow (\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 1, pw, \hat{c}, e, \tau, K)$
  $\boxed{\text{output } (e, \tau) \text{ and } (\mathsf{sid}, \mathsf{P}, K)}$

on message $(\hat{e}, \hat{\tau})$ to session $\mathsf{P}^{\mathsf{sid}}$:
if $\exists$ record $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{fullsid}, 0, pw, \perp, c, \perp)$:
  if $\exists$ record $(\mathsf{sid}, \mathsf{CP}, \mathsf{P}, \mathsf{fullsid}, 1, pw, c, \hat{e}, \hat{\tau}, K)$ then set $K' \leftarrow K$, and $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K')}$
  else if $\exists$ pw s.t. $\exists$ $(\mathsf{backdoor}, c, (\mathsf{fullsid}, pw), sk)$
    set $K^* \leftarrow \mathsf{KEM.dec}(sk, \hat{e})$
      if $\hat{\tau} = \mathsf{prf}(K^*, 1)$ then set $K' \leftarrow \mathsf{prf}(K^*, 2)$ and $\boxed{\text{output } (\mathsf{sid}, \mathsf{P}, K')}$
  otherwise $\boxed{\text{output } K' \xleftarrow{r} \{0,1\}^\kappa}$

Fig. 17: Game changes for the proof of Theorem 3

Now that e, τ, K are independent of pk, we can remove the usage of pk and the corresponding decryption query to $\mathcal{F}_{\mathsf{HIC}}(\hat{pw}, c)$ in passive case.

Next we change how $\mathsf{P}^{\mathsf{sid}}$ reacts for all other $\hat{c}$ cases: instead of querying $\mathcal{F}_{\mathsf{HIC}}(\hat{pw}, \hat{c})$ to get $\hat{pk}$ and generate corresponding e, τ and K, we set $\hat{pw} = \mathsf{c2pw}[\hat{c}]$ and consider two cases: If $\hat{pw} = (\mathsf{fullsid}, pw)$ (case 1) then Game 5 computes K in the same way as in Game 4, except we render the decryption query as retrieval from table $\mathsf{THIC}_{\hat{pw}}$, which is just a notational change; In any other case (case 2), Game 5 shortcuts the decryption and key-computation process and outputs a random e, with τ, K as random elements in $\{0,1\}^{\kappa}$. On the other side, $\mathsf{CP}^{\mathsf{sid}}$ in Game 5 computes K in the same way as in Game 4.

We argue that the change introduced in case 2 is negligible, i.e. if $\mathsf{c2pw}[\hat{c}]$ contains an entry $\hat{pw} \neq (\mathsf{fullsid}, pw)$, including $\hat{pw} = \perp$. The argument is hybrid and changes the view in q_P substeps, for each $\mathcal{F}_{\mathsf{pwKE}}$ session $\mathsf{P}^{\mathsf{sid}}$ invoked by $\mathcal{Z}$. We consider two sub-cases, (case 2a) where $\hat{c}$ was created via an adversarial encryption query on some key $\hat{pw}$, which does not match the decryption key $(\mathsf{fullsid}, pw)$ that $\mathsf{P}^{\mathsf{sid}}$ would use in Game 4 to decrypt this ciphertext (note that this $\hat{pw}$ is unique because of an abort in the case two encryption queries ever create the same ciphertext), and (case 2b) where $\hat{c}$ was not created in any encryption query. In either of these two sub-cases Game 4 would compute $K \leftarrow \mathsf{prf}(K^*, 2)$ for $(\hat{r}, \hat{pk}) \leftarrow \mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}((\mathsf{fullsid}, pw), \hat{c})$ and $(e, K^*) \leftarrow \mathsf{enc}(\hat{pk})$, and since in either case $\hat{c}$ was not inserted in table $\mathsf{THIC}_{(\mathsf{fullsid}, pw)}$ via an encryption query, this AdvDec query will embed a random pk into the decrypted plaintext. We make same change as in the passive case. i.e. randomize K^* from $\mathsf{KEM.enc}(pk)$, remove usage of prf, change e to be generated via a random public key picked by SIM, and remove query to $\mathcal{F}_{\mathsf{HIC}}$. Note that process on $\mathsf{CP}^{\mathsf{sid}}$ side remains unchanged in case 2, where the τ checking always fails and $\mathsf{CP}^{\mathsf{sid}}$ always outputs a random session key as in the previous game. We conclude that:

$$|P_4 - P_5| \leq 2(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.anonymity}} + 2q_P \cdot \varepsilon_{\mathsf{prf}} + q_P/(q_{IC} + q_P) \cdot \varepsilon_{\mathsf{KEM.sec}} \quad (15)$$

Game 6 (delaying password usage): In this game we delay using the password pw of session $\mathsf{P}^{\mathsf{sid}}$ to decrypt its outgoing message c to the moment when $\mathsf{P}^{\mathsf{sid}}$ receives an incoming message $(\hat{e}, \hat{\tau})$ in the case of an active attack, where $\mathcal{A}$ made a decryption query on c using correct password. In this case $\mathsf{P}^{\mathsf{sid}}$ will go through the list of backdoor records based on pw and c to retrieve sk for KEM decapsulation, whereas Game 5 retrieves sk from $\mathsf{P}^{\mathsf{sid}}$'s record. This is just a notational change. We also change how $\mathsf{CP}^{\mathsf{sid}}$ reacts to an incoming message $\hat{c}$, and we perform the decryption query only in the case adversary created $\hat{c}$ via encryption under correct key $(\mathsf{fullsid}, pw)$. Since Game 5 does not use the decrypted value (r, pk) and the associated trapdoor sk until the exact same case occurs, postponing this decryption does not matter as long as item $(*, c)$ is not written into table $\mathsf{THIC}_{(\mathsf{fullsid}, pw)}$ via an encryption query. However, the latter cannot happen in Game 5 because each $\mathsf{NewSession}$ and each encryption queries generate disjoint ciphertexts (a collision in the ciphertexts created by any of these queries leads to an abort). Both cases imply that $P_5 = P_6$.

GAME 7 (ideal-world game implied by $\mathcal{F}_{\mathsf{pwKE}}$ and SIM): This is the ideal-world game induced by functionality $\mathcal{F}_{\mathsf{pwKE}}$ interacting with simulator SIM of Figure 16. Since Game 6 matches the ideal-world interaction of Game 7 exactly we conclude that $P_6 = P_7$, which completes the proof.

□

D Security Proof for Theorem 4

KHAPE[41] is an aPAKE protocol which gives a generic compiler from any "UC key-hiding AKE". In [41] the key-hiding AKE is realized by several efficient single-flow protocols including 3DH, HMQV, and SKEME. In the case of HMQV it requires only 2 exponentiations per party, the resulting aPAKE has 4 flows and it is minimal in computational cost because it matches the "2 exponentiations" cost of unauthenticated Diffie-Hellman key exchange (see [41]).

- Half-Ideal Cipher $\mathsf{HIC} = (\mathsf{Enc}, \mathsf{Dec})$ on space of private and public keys
- pseudorandom function kdf

Password File Initialization on S's input $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$:

S generates two AKE key pairs (a, A) and (b, B), sets $e \leftarrow \mathsf{HIC}.\mathsf{Enc}(pw, (a, B))$, stores $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e, (b, A))$, and discards all other values

C on $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$		S on $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$
$(a, B) \leftarrow \mathsf{HIC}.\mathsf{Dec}(pw, e)$	$\xleftarrow{e}$	$(e, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$
$\xrightarrow{(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)}$	Key-Hiding AKE	$\xleftarrow{(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)}$
$\xleftarrow{k_1}$		$\xrightarrow{k_2}$
$\tau \leftarrow \mathsf{kdf}(k_1, 1)$	$\xrightarrow{\tau}$	$\gamma \leftarrow \perp$ if $\tau \neq \mathsf{kdf}(k_2, 1)$ else $\gamma \leftarrow \mathsf{kdf}(k_2, 2)$
	$\xleftarrow{\gamma}$	
$K_1 \leftarrow \perp$ if $\gamma \neq \mathsf{kdf}(k_1, 2)$ else $K_1 \leftarrow \mathsf{kdf}(k_1, 0)$ output K_1		$K_2 \leftarrow \perp$ if $\tau \neq \mathsf{kdf}(k_2, 1)$ else $K_2 \leftarrow \mathsf{kdf}(k_2, 0)$ output K_2

Fig. 18: aPAKE protocol KHAPE using Half-Ideal Cipher (changes from [41] marked so)

Here we claim that KHAPE remains a UC aPAKE[15] if the Ideal Cipher used to encrypt the private and public AKE keys in protocol KHAPE is replaced by a Half-Ideal Cipher. The benefit of replacing IC^* implementation of the ideal cipher on a group in [41] with a HIC is that, as we show with the m2F construction of HIC, the latter can be implemented over any group which admits an RO-indifferentiable random oracle hash onto a group, requires only one such hash to both encrypt and decrypt, and it has bandwidth overhead of 2κ bits. By contrast, the IC^* implementations of an ideal cipher on a group suggested in [41] work only for restricted elliptic curve groups and/or require more bandwidth and more computation in encryption and decryption. The same change can also benefit protocol OKAPE[37], which improves the round efficiency of KHAPE, and the change should be done similarly to KHAPE.

We show the KHAPE protocol using Half-Ideal Cipher for password-encryption of keys in Figure 18. Intuitively Half-Ideal Cipher works because: in KHAPE the attacker can attack client by sending an arbitrary ciphertext of his choice, but with the credential encryption implemented using an ideal cipher, the ciphertext commits the attacker to only one choice of key/password, for which he can decide the plaintext. And for all other keys the decrypted plaintext will be random, i.e. there are two requirements: (1) ciphertext $c = \mathsf{Enc}(k, m)$ is an encryption of some unique (k, m). HIC satisfies since for every (k, m) HIC.Enc and HIC.AdvEnc outputs $c = (s, T)$ which has no collisions on s part; (2) $\mathsf{Dec}(k', c)$ for $k' \neq k$ outputs random M, which is defined as in HIC.AdvDec and HIC.Dec.

Proof. We describe how the security proof for KHAPE should be adapted to the case using HIC. We specify how we deal with the HIC-specific differences when they occur and mark them in gray. We show that the environment's view of the *real-world* security game, denoted Game 0, i.e. an interaction between the real-world adversary and honest parties who follow protocol KHAPE, is indistinguishable from the environment's view of the *ideal-world* game, denoted Game 7, i.e. an interaction between simulator SIM[16] of Figures 20 and functionality $\mathcal{F}_{\mathsf{aPAKE}}$24. As before, we use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G7}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let $q_{\mathsf{pw}}, q_{\mathsf{HIC}}$, and q_{ses} be the upper-bounds on the number of resp. password files, HIC queries, and online S or C aPAKE sessions. Let $\epsilon^{\mathcal{Z}}_{\text{kdf}}(\mathsf{SIM}_{\mathsf{AKE}})$ and $\epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$ be the advantages of an environment who uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{HIC}} + q_{\text{ses}} + q_{\mathsf{pw}})$ exponentiations in $\mathbb{G}$ in resp. breaking the PRF security of kdf, and in distinguishing between the real-world AKE protocol and its ideal-world emulation of $\mathsf{SIM}_{\mathsf{AKE}}$ interacting with $\mathcal{F}_{\mathsf{khAKE}}$. Let $X' = Y = \mathcal{R} \times \mathcal{G}$ be the domain and range of the Half-Ideal Cipher HIC

[15] The UC asymmetric PAKE functionality, adapted to the case of explicit C-to-S authentication implemented by protocol KHAPE, is shown in Section F.

[16] here we only attach part of the simulator since the rest, i.e. the "Respodning to AKE messages" part, is same as in [41]

Initialize empty table THIC; (Notation $\mathsf{THIC}_{\hat{pw}}.X'$, $\mathsf{THIC}_{\hat{pw}}.Y$ and $\mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$ as in Fig. 20)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Generate keys (a, A), (b, B), set $e_{\mathsf{S}}^{\mathsf{uid}} \leftarrow \mathsf{Enc}(pw_{\mathsf{S}}^{\mathsf{uid}}, (a, B))$, and $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$
- On new (pw, x) to Enc: pick $r \xleftarrow{r} \mathcal{R}$, set $x' \leftarrow (r, x)$, output $y \xleftarrow{r} Y \setminus \mathsf{THIC}_{\hat{pw}}.Y$, add (pw, x', y) to THIC
- On new (pw, x', T) to AdvEnc: pick $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, $y \leftarrow (s, T)$, add (pw, x', y) to THIC, and output y
- On new (pw, y) to AdvDec: Output $x' \xleftarrow{r} X' \setminus \mathsf{THIC}_{\hat{pw}}.X'$, add (pw, x', y) to THIC
- On new (pw, y) to Dec: Query $(r, x) \leftarrow \mathsf{AdvDec}(pw, y)$, output x
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Set $(e_{\mathsf{S}}^{\mathsf{uid}}, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, send $e_{\mathsf{S}}^{\mathsf{uid}}$ and start AKE session $\mathsf{S}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)$, set k_2 to $\mathsf{S}^{\mathsf{sid}}$ output;
  If $\mathcal{Z}$ sends $\tau' = \mathsf{kdf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set K_2, γ as $\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2)$, else as $\perp, \perp$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message e' to C: Set $(a, B) \leftarrow (\mathsf{Dec}(pw, e'))$, and start AKE session $\mathsf{C}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$, set k_1 to $\mathsf{C}^{\mathsf{sid}}$ output, send $\tau = \mathsf{kdf}(k_1, 1)$ to $\mathcal{Z}$;
  If $\mathcal{Z}$ sends $\gamma' = \mathsf{kdf}(k_1, 2)$ to $\mathsf{C}^{\mathsf{sid}}$, set $K_1 = \mathsf{kdf}(k_1, 0)$ else $K_1 = \perp$

Fig. 19: Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE

used, let X be the domain of (private,public) keys in AKE(e.g. for both 3DH and HMQV we have $X = \mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p). Whereas [41] defined a mapping from groups to bitstrings and used a "bitstring" IC on the result, here we directly show a (randomized) IC on groups, precisely so that it can be used directly for public key systems where public keys live in groups[17], which is the case for all public keys we give as our examples (either DH-based or Lattice-based).

Game 0 (real world): This is the interaction, shown in Figure 19, of environment $\mathcal{Z}$ with the real-world protocol KHAPE, except that the symmetric encryption scheme is idealized as a Half-Ideal Cipher oracle. (Technically, this is a hybrid world where each party has access to the Half-Ideal Cipher functionality $\mathcal{F}_{\mathsf{HIC}}$.)

Game 1 (embedding random keys in $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvDec}$ outputs): We modify processing of $\mathcal{Z}$'s query (pw, y) to AdvDec[18] for any $y \notin \mathsf{THIC}_{\hat{pw}}.Y$, i.e. y for which $\mathsf{AdvDec}(pw, y)$ has not been yet defined. On such query Game 1 pick a random r, generates fresh key pairs (a, A) and (b, B),

[17] the secret keys in our cases are either $\mathbb{Z}_p$ elements or bitstrings which are in groups
[18] all the Enc,AdvEnc,AdvDec notation refers to oracles defined by $\mathcal{F}_{\mathsf{HIC}}$

Initialization

Initialize simulator $\mathsf{SIM_{AKE}}$, an empty table THIC, empty lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$

Notation: $\mathsf{THIC}_{\hat{pw}}.X' = \{x' \mid \exists y \ (pw, x', y) \in \mathsf{THIC}\}$, $\mathsf{THIC}_{\hat{pw}}.Y = \{y \mid \exists x' \ (pw, x', y) \in \mathsf{THIC}\}$ Let $\mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$ be a shortcut for set $\{s \in \mathcal{R} : \nexists \hat{m} \text{ s.t. } (\hat{m}, (s, T)) \in \mathsf{THIC}_{\hat{pw}}\}$.

Convention: First call to $\mathsf{SvrSession}$ or $\mathsf{StealPwdFile}$ for $(\mathsf{S}, \mathsf{uid})$ sets $e_{\mathsf{S}}^{\mathsf{uid}} \overset{r}{\leftarrow} Y$.Without loss of generality we assume that $\mathcal{A}$ uses AdvDec interface to implement a Dec query to $\mathcal{F}_{\mathsf{HIC}}$

Half-Ideal Cipher queries

- On query (Enc, pw, x) to $\mathcal{F}_{\mathsf{HIC}}$, send back y if $(pw, (r, x), y) \in \mathsf{THIC}$ for some r, otherwise pick $r \overset{r}{\leftarrow} \mathcal{R}$, $y \overset{r}{\leftarrow} Y \setminus \mathsf{THIC}_{\hat{pw}}.Y$, add $(pw, (r, x), y)$ to THIC, send back y
- On query $(\mathsf{AdvEnc}, pw, x', T)$ to $\mathcal{F}_{\mathsf{HIC}}$, send back y if $(pw, x', y) \in \mathsf{THIC}$, otherwise pick $s \overset{r}{\leftarrow} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, $y \leftarrow (s, T)$, add (pw, x', y) to THIC, and send back y
- On query (AdvDec, pw, y) to $\mathcal{F}_{\mathsf{HIC}}$, send back x' if $(pw, x', y) \in \mathsf{THIC}$, otherwise do:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \overset{r}{\leftarrow} X' \setminus \mathsf{THIC}_{\hat{pw}}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and:
     (a) If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" then set $(r, A, B) \leftarrow (r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     (b) Otherwise pick $r \overset{r}{\leftarrow} \mathcal{R}$, initialize keys A and B via two Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_{C} and B to PK_{S}

     Set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (r, A, B)$, send query $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define a as $\mathsf{SIM_{AKE}}$'s response, add A to CPK, set $x' \leftarrow (r, (a, B))$

  In either case add (pw, x', y) to THIC and send back x'

Stealing Password Data

On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$. If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "no password file," pass it to $\mathcal{A}$, otherwise declare $(\mathsf{S}, \mathsf{uid})$ $\mathsf{compromised}$ and:

1. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns no value then pick $r \overset{r}{\leftarrow} \mathcal{R}$, initialize keys A and B via two Init calls to $\mathsf{SIM_{AKE}}$, add A to PK_{C} and B to PK_{S}
2. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns pw then set $(r, A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw)$

Send $(\mathsf{Compromise}, B)$ to $\mathsf{SIM_{AKE}}$, define b as $\mathsf{SIM_{AKE}}$'s response, add B to CPK, set $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r, A, B)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$ to $\mathcal{A}$.

Starting AKE sessions

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$, initialize random function $R_{\mathsf{S}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $e_{\mathsf{S}}^{\mathsf{uid}}$ to $\mathcal{A}$ as a message from $\mathsf{S}^{\mathsf{sid}}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$ to $\mathsf{SIM_{AKE}}$.

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$ and message e' sent by $\mathcal{A}$ to $\mathsf{C}^{\mathsf{sid}}$, initialize random function $R_{\mathsf{C}}^{\mathsf{sid}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, and:

1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM_{AKE}}$
2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ check if e' was output by $\mathcal{F}_{\mathsf{HIC}}.\mathsf{Enc}$ on some (pw, x) or $\mathcal{F}_{\mathsf{HIC}}.\mathsf{AdvEnc}$ on some $(pw, (r, x))$, and:
   (a) If there is no such query then send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM_{AKE}}$
   (b) Otherwise define (pw, x) (resp.$(pw, (r, x))$) as the first such query(abort others) which outputted e', send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, and:
      i. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "wrong guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM_{AKE}}$
      ii. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess" then set $(a, B) \leftarrow x$ and run the AKE protocol on behalf of $\mathsf{C}^{\mathsf{sid}}$ on inputs $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$; When $\mathsf{C}^{\mathsf{sid}}$ terminates with key k then send $\tau \leftarrow \mathsf{kdf}(k, 1)$ to $\mathcal{A}$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{kdf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$

Fig. 20: Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$

sets $x' \leftarrow (r,(a,B))$, and if $x' \notin \mathsf{THIC}_{\hat{pw}}.X'$ then it sets $\mathsf{AdvDec}(pw,y) \leftarrow x'$. If $x' \in \mathsf{THIC}_{\hat{pw}}.X'$, i.e. x' is already generated by $\mathsf{AdvEnc}(pw,\cdot,\cdot)$ or $\mathsf{Enc}(pw,\cdot)$, Game 1 aborts. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S},\mathsf{uid})$ then the game also sets $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (r,A,B)$.

The divergence this game introduces is due to the probability $(q_{\mathsf{HIC}})^2/2^n$ of ever encountering an abort [19], which leads to $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (q_{\mathsf{HIC}})^2/2^n$.

Game 2 (random $e_{\mathsf{S}}^{\mathsf{uid}}$ in the password file): We change $\mathsf{StorePwdFile}$ processing by picking ciphertext $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random element in $\{0,1\}^n \times \mathbb{G}$ instead of via query to Enc, then we pick two key pairs (a,A), (b,B), pick a random r and define $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r,A,B)$, set $x' \leftarrow (r,(a,B))$. If $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{THIC}_{\hat{pw}}.Y$ for *any* pw, not necessarily $pw_{\mathsf{S}}^{\mathsf{uid}}$, the game aborts. The game also aborts if $x' \in \mathsf{THIC}_{\hat{pw}}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$. Otherwise the game sets $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow x'$ and $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r,A,B)$. The divergence this game introduces is due to the probability of abort occuring in either case, which leads to $|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq 2q_{\mathsf{pw}}q_{\mathsf{HIC}}/2^n$.

Game 3 (abort on ambiguous ciphertexts): In[41] to eliminate the possibility of ambiguous ciphertexts we introduce an abort if $\mathsf{IC.Enc}$ oracle picks the same ciphertext for any two queries containing pair (pw_1, x_1) and (pw_2, x_2). Now this ambiguous case is already considered and avoided in definition of Enc and AdvEnc in $\mathcal{F}_{\mathsf{HIC}}$. so we have $\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$.

Taking stock of the game. Let us review how Game 3 operates: The initialization of password file $\mathsf{file}[\mathsf{uid},\mathsf{S}]$ on password $pw_{\mathsf{S}}^{\mathsf{uid}}$ picks a random r and fresh keys (a,A), (b,B), keeps them as $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) = (r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) = (r,A,B)$, picks $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random string, and programs $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ to $(r,(a,B))$. Oracle AdvDec on inputs (pw',y) for which decryption is undefined, picks some random r' and fresh key pairs (a',A') and (b',B'), and programs $\mathsf{AdvDec}(pw',y)$ to $(r',(a',B'))$. In addition, if $y = e_{\mathsf{S}}^{\mathsf{uid}}$ then it assigns $pk_{\mathsf{S}}^{\mathsf{uid}}(pw') \leftarrow (r',(a',B'))$. Finally, encryption is now unambiguous, i.e. every ciphertext e can be output by Enc or AdvEnc on only one pair (pw,x').

This is already very close to how simulator SIM operates as well. The crucial difference between the ideal-world interaction and Game 3, is that in Game 3, $r_{\mathsf{S}}^{\mathsf{uid}}$ and keys $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ are generated at the time of password file initialization, and $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set to $(r_{\mathsf{S}}^{\mathsf{uid}}, (a_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}))$ at the same time. In the ideal-world game these values are undefined until password compromise, and $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set only after offline dictionary attack succeeds in finding $pw_{\mathsf{S}}^{\mathsf{uid}}$. This delayed generation of the keys in $\mathsf{file}[\mathsf{uid},\mathsf{S}]$ is possible because AKE sessions which S and C run on these keys can be simulated without knowledge of these keys, a key-hiding AKE functionality allows precisely for such simulation, as we show next. Delayed r generation is also okay because it's not used in AKE sessions.

[19] the probability of collission comes from the n-bit string $r$is is at most $(q_{\mathsf{HIC}})^2/2^n$

Initialize simulator $\mathsf{SIM}_{\mathsf{AKE}}$, empty table THIC and $\mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, and lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Pick $e_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} Y$, mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ as fresh
- On new$^{(!)}$ (pw, x) to Enc: Pick $r \xleftarrow{r} \mathcal{R}$, set $x' \leftarrow (r, x)$, output $y \xleftarrow{r} Y \setminus \mathsf{THIC}_{\hat{pw}}.Y$, add (pw, x', y) to THIC
- On new$^{(!)}$ (pw, x', T) to AdvEnc: Pick $s \xleftarrow{r} \mathsf{THIC}_{\hat{pw}}.\mathsf{s}[T]$, $y \leftarrow (s, T)$, add (pw, x', y) to THIC, and output y
- On new$^{(!)}$ (pw, y) to AdvDec:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \xleftarrow{r} X' \setminus \mathsf{THIC}_{\hat{pw}}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is fresh or $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$ then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$, pick $r \xleftarrow{r} \mathcal{R}$, initialize A and B via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}
     (b) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$&$pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, set $(r, A, B) \leftarrow (r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     In both cases (a) and (b), set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (r, A, B)$, define a as $\mathsf{SIM}_{\mathsf{AKE}}$'s response to $(\mathsf{Compromise}, A)$, add A to CPK, and set $x' \leftarrow (r, (a, B))$

  Add (pw, x', y) to THIC and send back x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ $\mathsf{compromised}$ and:
  If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$ then set $(r, A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$;
  Else pick $r \xleftarrow{r} \mathcal{R}$, initialize A and B via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S};
  In either case, set $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (r, A, B)$, define b as $\mathsf{SIM}_{\mathsf{AKE}}$'s response to $(\mathsf{Compromise}, B)$, add B to CPK, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_{\mathsf{S}}^{\mathsf{sid}}$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, output $e_{\mathsf{S}}^{\mathsf{uid}}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and:
  1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ then: (1) set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, otherwise set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$; (2) send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
  2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ then:
     (a) If e' was not output by Enc or AdvEnc or it was output on $(pw', \cdot)$ for $pw' \neq pw$, then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathsf{SIM}_{\mathsf{AKE}}$
     (b) If e' was output by Enc on (pw, x) or AdvEnc on $(pw, (\cdot, x), \cdot)$ then set $(a, B) \leftarrow x$, run $\mathsf{C}^{\mathsf{sid}}$ of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If $\mathsf{C}^{\mathsf{sid}}$ terminates with k, output $\tau \leftarrow \mathsf{kdf}(k, 1)$ and $K_1 \leftarrow \mathsf{kdf}(k, 0)$

<u>Responding to AKE messages:</u>

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$, otherwise $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ set $k_1 \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}, \alpha)$, output $\tau \leftarrow \mathsf{kdf}(k_1, 1)$
  2. Otherwise output $\tau \xleftarrow{r} \{0,1\}^{\kappa}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\tau' = \mathsf{kdf}(k_2, 1)$ for $k_2 = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, \alpha)$ where $(\cdot, (A, B)) = pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$, then output $(K_2, \gamma) \leftarrow (\mathsf{kdf}(k_2, 0), \mathsf{kdf}(k_2, 2))$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by $\mathsf{C}^{\mathsf{sid}}$ where $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, then output $K_2 \xleftarrow{r} \{0,1\}^{\kappa}$ and $\gamma \xleftarrow{r} \{0,1\}^{\kappa}$
  3. In all other cases output $(K_2, \gamma) \leftarrow (\perp, \perp)$
- On γ' to $\mathsf{C}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ and $\gamma' = \mathsf{kdf}(k_1, 2)$, output $K_1 \leftarrow \mathsf{kdf}(k_1, 0))$
  2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ and γ' was generated by $\mathsf{S}^{\mathsf{sid}}$ for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, output K_1 equal to the key K_2 output by $\mathsf{S}^{\mathsf{sid}}$
  3. In all other cases output $K_1 \leftarrow \perp$
- On $(\mathsf{ComputeKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ if $pk \in PK_{\mathsf{P}}$, $pk' \in CPK$

Fig. 21: KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 7)

Game 4 (Using $\mathsf{SIM}_{\mathsf{AKE}}$ for AKE's on honestly-generated keys): In Game 4 we modify Game 3 by replacing all honest parties that run AKE instances on keys A, B generated either in password file initialization or by oracle AdvDec, with a simulation of these AKE instances via simulator $\mathsf{SIM}_{\mathsf{AKE}}$. For notational brevity we say that query (pw, x) to Enc(resp. (pw, x', T) to AdvEnc) or (pw, y) to AdvDec are new$^{(!)}$ as a shortcut for saying that table THIC includes no prior tuple corresponding to these inputs. If such tuple exists then Enc, AdvEnc and AdvDec oracles use the retrieved (key,input,output) tuple to answer the according query. We also omit the possibilities of the game aborts, because such aborts happen only with negligible probability. These aborts occur in three places, all marked $^{(*)}$: (1) When $e_{\mathsf{S}}^{\mathsf{uid}}$ is chosen in $\mathsf{StorePwdFile}$ the game aborts if $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{THIC}_{\hat{pw}}.Y$ for any pw (not necessarily $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$); (2) When x' is then set as $x' \leftarrow (r, (a, B))$, the game aborts if $x' \in \mathsf{THIC}_{\hat{pw}}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$; (3) When $x' \leftarrow (r, (a, B))$ is set in AdvDec query (pw, y) the game aborts also if $x' \in \mathsf{THIC}_{\hat{pw}}.X'$.

Game 4 operates like Game 3, except that it outsources AKE key generation in $\mathsf{StorePwdFile}$ and AdvDec to $\mathsf{SIM}_{\mathsf{AKE}}$, and whenever $\mathsf{S}^{\mathsf{sid}}$ or $\mathsf{C}^{\mathsf{sid}}$ runs AKE on such keys these executions are outsourced to $\mathsf{SIM}_{\mathsf{AKE}}$, while the game emulates what $\mathcal{F}_{\mathsf{khAKE}}$ would do in response to $\mathsf{SIM}_{\mathsf{AKE}}$'s actions. In particular, Game 4 initializes random function $R_{\mathsf{P}}^{\mathsf{sid}}$ for every AKE session $\mathsf{P}^{\mathsf{sid}}$ invoked by emulated $\mathcal{F}_{\mathsf{khAKE}}$. Whenever C and S run an AKE instance under keys generated by AKE key generation the game, playing $\mathcal{F}_{\mathsf{khAKE}}$, triggers $\mathsf{SIM}_{\mathsf{AKE}}$ with messages resp. $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C})$. When $\mathsf{SIM}_{\mathsf{AKE}}$ translates the real-world adversary's behavior into $\mathsf{Interfere}$ actions on these sessions, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by marking these sessions as actively attacked. If $\mathsf{SIM}_{\mathsf{AKE}}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ on activey attacked session, its output key k is set to $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the keys this session runs under, which are $(B_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}})$ for S, and keys (A, B) defined by $\mathsf{AdvDec}(pw, e')$ for C. The game must also emulate $\mathsf{ComputeKey}$ interface of $\mathcal{F}_{\mathsf{khAKE}}$ and let $\mathsf{SIM}_{\mathsf{AKE}}$ evaluate $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ for any $pk \in PK_{\mathsf{P}}$ and any $pk' \in CPK$. (Note that all sessions emulated by $\mathsf{SIM}_{\mathsf{AKE}}$ run on public keys pk' which are created by the Init interface.) Set PK_{S} contains only one key, $B_{\mathsf{S}}^{\mathsf{uid}}$, while set PK_{C} contains $A_{\mathsf{S}}^{\mathsf{uid}}$ and all keys A' created by AdvDec queries. Set CPK consists of $A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}$, because these were compromised in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ initialization, which used the corresponding private keys, and all client-side keys A' generated in AdvDec queries, because each AdvDec query creates and immediately compromises key A', since it needs to embed the corresponding private key a' into AdvDec output. Finally, if $\mathsf{SIM}_{\mathsf{AKE}}$ sends NewKey on non-attacked session, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by issuing random keys to such sessions except if $\mathsf{C}^{\mathsf{sid}}$ runs under key pair $(A', B') = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, which matches the key pair used by $\mathsf{S}^{\mathsf{sid}}$, in which case the game copies the key output by the session which terminates first into the key output by the session which terminates second. The rest of the code is as in Game 3: C uses its key k_1 to compute authenticator $\tau = \mathsf{kdf}(k_1, 1)$ and its local output $K_1 = \mathsf{kdf}(k_1, 0)$, while S uses its key k_2 to verify the incoming authenticator τ' and outputs $K_2 = \mathsf{kdf}(k_2, 0)$ if $\tau' = \mathsf{kdf}(k_2, 1)$ and $K_2 = \perp$ otherwise.

The one case where a party might not run AKE on keys generated via a call to $\mathsf{SIM}_{\mathsf{AKE}}$ is client session C which receives e' which was output by $\mathsf{Enc}(pw, x)$ or $\mathsf{AdvEnc}(pw, (\cdot, x), \cdot)$ for some x and pw matching the password input to $\mathsf{C}^{\mathsf{sid}}$. In this case $\mathsf{C}^{\mathsf{sid}}$ runs AKE on $(a, B) = x$, and since wlog these keys are chosen by the adversary and not by $\mathsf{SIM}_{\mathsf{AKE}}$, we cannot outsource that execution to $\mathsf{SIM}_{\mathsf{AKE}}$. As we said above, functionality $\mathcal{F}_{\mathsf{khAKE}}$ does not admit honest parties running AKE on arbitrary private keys a, hence $\mathsf{SIM}_{\mathsf{AKE}}$ does not have an interface to simulate such executions. In Game 4 such AKE instances are executed as in Game 3.

Since Game 4 and Game 3 are identical except for replacing real-world AKE executions with the game emulating functionality $\mathcal{F}_{\mathsf{khAKE}}$ interacting with $\mathsf{SIM}_{\mathsf{AKE}}$, it follows that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$

Game 5 (delay $r^{\mathsf{uid}}_{\mathsf{S}}$, $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ generation until password compromise): In Game 4, $r^{\mathsf{uid}}_{\mathsf{S}}$ and keys $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ are initialized and compromised in $\mathsf{StorePwdFile}$, in Game 5 we postpone these steps until password compromise. This change can be done in several steps.

Denote first step as Game 5(a), we remove compromising $B^{\mathsf{uid}}_{\mathsf{S}}$, adding it to CPK and setting $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$. $\mathcal{Z}$ cannot notice this change because in Game 4, only $\mathsf{StealPwdFile}$ will need $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$, and compromising $B^{\mathsf{uid}}_{\mathsf{S}}$ to get $b^{\mathsf{uid}}_{\mathsf{S}}$ is not needed anywhere else except when generating $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$.

In Game 5(b) we make a change in AdvDec, that if $y \neq e^{\mathsf{uid}}_{\mathsf{S}}$ then set $x' \xleftarrow{\text{r}} X' \setminus \mathsf{THIC}_{\hat{pw}}.X'$, while in Game 4 we set $x' \leftarrow (r, (a, B))$ for randomly initialized $(r, (a, B))$, with restriction that this x' hasn't been set before. This is just a notational change.

Then in Game 5(c) we remove compromising $A^{\mathsf{uid}}_{\mathsf{S}}$, adding it to CPK, setting x' and adding $(pw^{\mathsf{uid}}_{\mathsf{S}}, x', e^{\mathsf{uid}}_{\mathsf{S}})$ to THIC in $\mathsf{StorePwdFile}$, and delay them to new$^{(!)}$ (pw, y) to AdvDec. After this change, in $\mathsf{StorePwdFile}$ we now only initialize $r^{\mathsf{uid}}_{\mathsf{S}}$ and $(A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, add them to PK and pick $e^{\mathsf{uid}}_{\mathsf{S}}$. Since $(pw^{\mathsf{uid}}_{\mathsf{S}}, x', e^{\mathsf{uid}}_{\mathsf{S}})$ is no longer added to THIC in $\mathsf{StorePwdFile}$, query $(pw^{\mathsf{uid}}_{\mathsf{S}}, e^{\mathsf{uid}}_{\mathsf{S}})$ is now new$^{(!)}$ to AdvDec, and we add that in this case AdvDec responds by retrieving $(r^{\mathsf{uid}}_{\mathsf{S}}, A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, compromising $A^{\mathsf{uid}}_{\mathsf{S}}$, setting corresponding x' and adding $(pw^{\mathsf{uid}}_{\mathsf{S}}, x', e^{\mathsf{uid}}_{\mathsf{S}})$ to THIC. For any other queries, AdvDec reacts same as in Game 5(b). Game 5(c) and Game 5(b) is identical since we only postpone executing those steps removed from $\mathsf{StorePwdFile}$.

In Game 5(d) we further remove usage of $(A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$ when responding to AKE messages, except for input to $R^{\mathsf{sid}}_{\mathsf{P}}$ in actively attacked sessions. We change $\mathsf{hbc}(A, B)$ in Game 5(c) to $\mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$ if $(A, B) = (A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, and rnd otherwise. Similarly we change $\mathsf{act}(A, B)$ in Game 5(c) to $\mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}$ if $(A, B) = (A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, which corresponds to active attack, otherwise set to rnd and derive corresponding k_1 from random element of $\{0, 1\}^{\kappa}$ instead of $R^{\mathsf{sid}}_{\mathsf{C}}(A, B, \alpha)$, from randomness of $R^{\mathsf{sid}}_{\mathsf{C}}$ this change makes indistinguishable difference to $\mathcal{Z}$. Since

these are only notational changes and $\mathcal{Z}$ cannot notice them, Game 5(d) and Game 5(c) are identical to $\mathcal{Z}$.
Finally, in Game 5(e) we remove steps of picking $r_{\mathsf{S}}^{\mathsf{uid}}$ and initializing $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via $\mathsf{SIM}_{\mathsf{AKE}}$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$ or $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, depending on which happens first. In order to set $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ only after $\mathcal{A}$ finds $pw_{\mathsf{S}}^{\mathsf{uid}}$ via successful offline dictionary attack, we first mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ fresh in $\mathsf{StorePwdFile}$, and mark it $\mathsf{compromised}$ anytime $\mathcal{A}$ runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$.
If $\mathcal{A}$ first runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, we pick $r_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \mathcal{R}$, initialize $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add $A_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{C} and $B_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{S}, and later upon query $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is already marked $\mathsf{compromised}$, we simply retrieve $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, then compromise $A_{\mathsf{S}}^{\mathsf{uid}}$ and set x' as in Game 5(d). In the other case, if $\mathsf{AdvDec}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ runs first, which means at this moment $pw_{\mathsf{S}}^{\mathsf{uid}}$ must be fresh, we treat it same way as before, and just like any other $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$, where we pick $r_{\mathsf{S}}^{\mathsf{uid}} \xleftarrow{r} \mathcal{R}$, init $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via $\mathsf{SIM}_{\mathsf{AKE}}$, add them to PK and save $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ into $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ for future retrieval. We also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, and later if $\mathcal{A}$ runs $\mathsf{StealPwdFile}$ and there exists record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, then just directly retrieve $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ from $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$ and skip initialization. In addition we also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ upon query $\mathsf{AdvDec}(pw, e_{\mathsf{S}}^{\mathsf{uid}})$ even if $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. Game 5(e) is identical to Game 5(d) since we only postpone $(r_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ initialization. Thus we conclude: $\mathsf{G5} = \mathsf{G4}$

Game 6 (replace kdf output with random string in passive sessions): In Game 5, in passive sessions, i.e. any sessions except actively attacked sessions, τ, γ are all derived from kdf of k_1 or k_2. In Game 6 in these sessions we remove usage of kdf and directly assign random elements of $\{0,1\}^\kappa$ to these values. Also we replace verifying τ', γ' via checking $\tau' = \mathsf{kdf}(k_2, 1), \gamma' = \mathsf{kdf}(k_1, 2)$ with checking whether they're generated by corresponding hbc parties, since these two checking methods are actually equal. In addition, we further remove usage of k_1 and k_2 in passive sessions, and instead set $K_2 \leftarrow \{0,1\}^\kappa$, and in matching sessions we copy K_2 to K_1, as Game 5 copy k_1 to k_2 or vice versa in such sessions. Since there're at most q_{ses} such sessions, and from security of kdf, the difference between Game 5 and Game 6 is negligble to $\mathcal{Z}$, i.e. $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}} \epsilon_{\text{kdf}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$

Game 7 (Ideal-world game): This is the ideal-world interaction, i.e. an interaction of environment $\mathcal{Z}$ with simulator SIM and functionality $\mathcal{F}_{\mathsf{aPAKE}}$, shown in Figure 21.

Observe that Game 6 is identical to the ideal-world Game 7. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof. $\square$

E Lattice-Based UC PAKE from EKE and Saber KEM

We argue that the CPA-secure Key Encapsulation Mechanism (KEM) at the heart of the Saber [28] public key encryption, whose security is based on the

Module-LWR problem, achieves also the *anonymity* and *uniform public keys*property, see Section 2, under the same Module-LWR assumption. In Figure 22 we show the EKE-KEM construction, which is Figure 10 instantiated with Saber KEM. Note that Theorem 3 implies that the resulting protocol is a UC PAKE under the Module-LWR assumption.

Saber Cryptosystem. We define the notation needed to introduce Saber. Let $\mathbb{Z}_q$ be the ring of integers modulo q represented in $[-q/2+1, q/2]$ and R_q a polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$, where n is a power of 2 and a security parameter (and a length of the session key output by Saber). Let $R_q^{l_1 \times l_2}$ be the ring of l_1 by l_2 matrices over R_q. (Below we use uppercase bold font to denote matrices and lowercase bold font to denote vectors.) Let $\mathcal{U}(R_q)$ be a uniform distribution over R_q and let $\chi_\mu(R_q)$ be a distribution where each polynomial coefficient is chosen from a binomial distribution centered at 0 with parameter μ (and standard deviation $\sqrt{\mu/2}$). When these distributions are taken over a matrix space $R_q^{l_1 \times l_2}$ instead of R_q, this stands for choosing each matrix entry (or vector if $l_2 = 1$) according to that distribution.

- Parameters l, μ, moduli $q = 2^{\epsilon_q}, p = 2^{\epsilon_p}, T = 2^{\epsilon_T}$, for $\epsilon_q > \epsilon_p > (\epsilon_T + 1)$
- Half-Ideal Cipher HIC on domain $\mathcal{R} \times \mathcal{PK}$ for $\mathcal{PK} = R_p^{l \times 1} \times \{0,1\}^{256}$
- Random oracle hash H onto $\{0,1\}^\kappa$

P_0 on $\mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1, pw_0)$		P_1 on $\mathsf{NewSession}(\mathsf{sid}, \mathsf{P}_1, \mathsf{P}_0, pw_1)$
	(Assume $\mathsf{P}_0 \leq_{lex} \mathsf{P}_1$ & $\mathsf{fullsid} = (\mathsf{sid}, \mathsf{P}_0, \mathsf{P}_1)$)	
$seed_{\mathbf{A}} \xleftarrow{r} \{0,1\}^{256}$		
$\mathbf{A} \leftarrow \mathsf{gen}_{\mathbf{A}}(seed_{\mathbf{A}}) \in R_q^{l \times l}$		
$\mathbf{s} \leftarrow \chi_\mu(R_q^{l \times 1})$		$\mathbf{s}' \leftarrow \chi_\mu(R_q^{l \times 1})$
$\mathbf{b} = \lfloor \mathbf{A}^T \mathbf{s} + \mathbf{h} \rceil_{q \to p}$		
$c \leftarrow \mathsf{HIC.Enc}((\mathsf{fullsid}, pw_0), (\mathbf{b}, seed_{\mathbf{A}}))$	$\xrightarrow{c}$	
		$(\mathbf{b}, seed_{\mathbf{A}}) \leftarrow \mathsf{HIC.Dec}((\mathsf{fullsid}, pw_1), c_0)$
		$\mathbf{A} \leftarrow \mathsf{gen}_{\mathbf{A}}(seed_{\mathbf{A}})$
		$\mathbf{b}' = \lfloor \mathbf{A}\mathbf{s}' + \mathbf{h} \rceil_{q \to p}$
		$v' = \mathbf{b}^T [\mathbf{s}']_p + h_1$
		$k' = \lfloor v' \rceil_{p \to 2}$
		$c = \lfloor v' \rceil_{p \to T} \bmod T/2$
$v = \mathbf{b}'^T [\mathbf{s}]_p$	$\xleftarrow{(\mathbf{b}', c), \tau}$	$\tau \leftarrow \mathsf{H}(k', (\mathbf{b}, seed_{\mathbf{A}}))$
$k = \lfloor v - \lfloor c \rceil_{T \to p} + h_2 \rceil_{p \to 2}$		output $K_1 \leftarrow \mathsf{H}(k')$
if $\tau = \mathsf{H}(k, (\mathbf{b}, seed_{\mathbf{A}}))$ then output $K_0 \leftarrow \mathsf{H}(k)$		
else output $K_0 \xleftarrow{r} \{0,1\}^\kappa$		

Fig. 22: Protocol EKE-KEM of Section 5.1 instantiated with Saber KEM

Denote $\lfloor\cdot\rfloor$ as flooring to the nearest lower integer and $\lfloor\cdot\rceil$ as rounding to the nearest integer. The operation $\lfloor\cdot\rfloor_{q\to p}$ takes an integer $x \in \mathbb{Z}_q$ as input and outputs $\lfloor p/q \cdot x\rfloor \in \mathbb{Z}_p$, and similarly $\lfloor x\rceil_{q\to p} = \lfloor p/q \cdot x\rceil \in \mathbb{Z}_p$. We use $[\cdot]_p$ to denote mod p operation. Saber uses moduli $q = 2^{\epsilon_q}, p = 2^{\epsilon_p}, T = 2^{\epsilon_T}$ with $q > p > T$, and the constants added in $\lfloor\cdot\rfloor$ in Figure 22 are set as $h_1 = \frac{q}{2p} \in R_p, h_2 = \frac{p}{4} - \frac{p}{2T} + \frac{q}{2p} \in R_p$ and $\mathbf{h} = \frac{q}{2p} \in R_q^{l\times 1}$. (Saber NIST proposal [28] suggests parameters $\epsilon_q = 13, \epsilon_p = 10, \epsilon_T = 4$.)

Security of Saber relies on the hardness of the Module Learning with Rounding problem (Mod-LWR)[6], defined as a variant of the Learning with Errors (LWE) problem where the error is implicitly generated by the integer rounding operation. The advantage of a polynimal-time adversary $\mathcal{A}$ against the generalized Mod-LWR problem is defined as follows for parameters m, l, p, q, μ s.t. $p < q$:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{Mod-LWR}}_{m,l,q,p,\mu}(\mathcal{A}) = \Big|\ &\Pr[\ 1 \leftarrow \mathcal{A}(\mathbf{A}, \lfloor\mathbf{As}\rceil_{q\to p})\ :\ \mathbf{s} \xleftarrow{r} \chi_\mu(R_q^{l\times 1}); \mathbf{A} \xleftarrow{r} \mathcal{U}(R_q^{m\times l})\ ]\\ &-\ \Pr[\ 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{u})\ :\ \mathbf{A} \xleftarrow{r} \mathcal{U}(R_q^{m\times l}); \mathbf{u} \leftarrow \mathcal{U}(R_p^{m\times 1})\ ]\ \Big|\end{aligned}$$

EKE instantiated with Saber. In Figure 22 we show the EKE-KEM protocol of Section 5.1 instantiated with Saber KEM. The resulting protocol is essentially a Saber key exchange protocol but with the initiator's public key encrypted using Half-Ideal Cipher, and with the responder attaching a key-and-password confirmation message.

The following theorem, proven in [28], states the CPA security of Saber under the Mod-LWR assumption:

Theorem 5. *Assuming* $\mathsf{gen}_{\boldsymbol{A}}$ *to be a random oracle. For any adversary* $\mathcal{A}$*, there exists two adversaries* B_1 *and* B_2*, such that:*

$$\boldsymbol{Adv}^{\mathsf{IND\text{-}CPA}}_{Saber}(\mathcal{A}) \leq \boldsymbol{Adv}^{\mathsf{mod-lwr}}_{l,l,\mu,q,p}(B_1) + \boldsymbol{Adv}^{\mathsf{mod-lwr}}_{l+1,l,\mu,q,p}(B_2) \quad \text{if} \quad q/p \leq p/T.$$

The two further KEM properties needed in the EKE-KEM protocol of Section 5.1 are ciphertext anonymity and uniform public keys (see Section 2.2 for definition of these notions), but Saber satisfies these properties under the same Mod-LWR assumption:

Theorem 6. *Saber KEM satisfies the* uniform public keys *property on domain* $\mathcal{PK}$ *and the* anonymity *property under Module-LWR assumption.*

Proof. Below we sketch the proof of Theorem 6. The uniform public keys property which requires the public key generated to be indistinguishable from uniform, is by definition of Module-LWR problem and proved in Game 2 in the same proof of Theorem 5, where $\mathbf{b}$ is replaced with a uniform value. The anonymity property, which requires that given two different public keys and a ciphertext $(\mathbf{b}', c)$ generated by one of them, it's computationally hard to distinguish the correct key, is also satisfied by Saber since without information about secret $\mathbf{s}'$, LWR samples $(\mathbf{A}, \mathbf{b'})$ and $(\mathbf{b}, v')$ are both indistinguishable from random elements by definition of LWR. The full proof is given in [51]. □

Comparison with prior lattice-based PAKEs. We recall prior work on lattice-based PAKE's to compare it to the $\mathsf{EKE\text{-}KEM}$(Saber) protocol shown in Figure 22. The short summary is that $\mathsf{EKE\text{-}KEM}$(Saber) appears to be the first UC PAKE from lattice assumption, and it also forms a two-round PAKE which has the smallest bandwidth among prior lattice-based PAKE proposals. Indeed, its bandwidth is minimal because it adds only 3κ bits to the underlying (plain) Key Exchange implemented by KEM.

The first lattice-based PAKE was shown by Katz and Vaikuntanathan [47], where both parties send a CCA-encrypted ciphertext to each other, compute Approximate Smooth Projective Hash (ASPH) values on ciphertexts, and conduct a key reconciliation subprotocol to derive a session key. This protocol needs three rounds and the underlying CCA-encrypted ciphertext actually contains n CPA-encrypted ciphertexts, which is costly to compute. KV is further optimized by Zhang and Yu [60], who proposed a 2-round PAKE with a new ASPH based on a "splittable CCA-secure encryption". Following the same track, Benhamouda [11] adapts Groce and Katz [40] framework using KV's realization of ASPH and as result, gets new 3-round and 2-round PAKEs in standard model, and they further optimize the protocol to one round, using the same SS-NIZK approach as in [60]. However, construction of lattice-based SS-NIZK in standard model appears to be still an open question. Moreover, all of these works rely on standard-model CCA-secure encryption which appears expensive to realize. We refer for more details to [46], who explain the effiency challenges in this line of work.

[46] is the first to construct a lattice-based PAKE in the standard model which only requires CPA-secure encryption, and it's significantly more efficient compared compared to the PAKEs which use CCA-secure lattice-based encryption. Ding et al. [31] proposed a still much more efficient scheme assuming ROM. Their scheme appears to be a lattice-based counterpart to the PPK protocol of Boyko et al. [18], and thus also to EKE. The significant difference, however, is that in PPK hashed password is used as a one-time mask on the KE messages, wheres in EKE it is used as key that encrypts the KE messages using an ideal cipher. Consequently, Ding et al. [31] analyze the security of their PAKE in the "BMP" model of [18], whereas we analyze our proposal in the UC PAKE model. (We note, however, that the BMP model for PAKE is mostly likely equivalent to the recently proposed UC *relaxed PAKE* model [1].) Apart of this difference in analysis, the fact that our analysis uses KEM as a black-box allows instant reuse of efficiency improvements in lattice-based KEMs. Indeed, Saber uses a much smaller field modulus $q = 2^{13}$ compared to $2^{32} - 1$ in [31], which reduces the size of both the KEM public key and the ciphertext (and these sizes are further reduced by rounding operations).[20]. We benchmark the bandwidth for the last three lattice PAKEs

[20] Saber[28] authors argue that this more aggressive parameter suffices in their construction, and while using large prime moduli can possibly adopt Number Theoretic Transformation (NTT) to speedup polynomial multiplications, [28] using power-of-two moduli has its own advantages including: (1) avoiding modular

discussed above, which seem to form the most efficient proposals. For security parameter $\kappa = 128$, the total bandwidth is 207 KB for [46], 8.32 KB for [31] and 1.376 KB for $\mathsf{EKE\text{-}KEM}$(Saber).

Table E provides a detailed comparison on efficiency of these last three lattice PAKEs.

Scheme	Bndw (KB)	Rounds	Assum	Security	Model
JGHNW[46]	207	3	(R)LWE	BPR	Standard
Ding17[31]	8.320	2	(R)LWE PairWE	Bokyo[18]	ROM
EKE-KEM (Saber)	1.376	2	LWR	UC	ROM

Table 1: Comparison of lattice-based PAKE protocols based on bandwidth, rounds, security assumptions, security claims, and security model

F PAKE and aPAKE functionalities

In Figure 23 we recall a symmetric PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ of Canetti et al[21], adapted to the multi-session setting. $\mathcal{F}_{\mathsf{pwKE}}$ is used in Section 5 to argue that protocol EKE 7 and EKE-KEM 10 are UC-secure PAKE. In Figure 24 we recall a asymmetric PAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ of [39] adapted to the case of explicit C-to-S authentication in [41]. $\mathcal{F}_{\mathsf{aPAKE}}$ is used in Section 6 to prove that KHAPE remains a UC aPAKE after replacing IC with HIC.

reduction and rejection sampling; (2) the use of LWR halves the amount of randomness required compared to LWE-based schemes, and thus reduces bandwidth; (3) the module structure provides flexibility by reusing one core component for multiple security levels. See more details in [28]

Notation: κ is the security parameter, P, P' are arbitrary parties, $\mathcal{A}$ is the ideal-world adversary

On query $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ from party P:

If this is the first $\mathsf{NewSession}$ query for this sid, or it is the second one and the previous one was $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw')$, then record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked fresh and forward $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}')$ to $\mathcal{A}$.

On query $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ from adversary $\mathcal{A}$:

If there is record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked fresh then:

- If $pw^* = pw$ then mark this record $\mathsf{compromised}$ and reply "correct" to S
- If $pw^* \neq pw$ then mark this record $\mathsf{interrupted}$ and reply "incorrect" to S

On query $(\mathsf{NewKey}, \mathsf{sid}, P, K^*)$ from adversary $\mathcal{A}$:

If there is record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ marked $\mathsf{flag} \neq \mathsf{completed}$ then:

- If $\mathsf{flag} = \mathsf{compromised}$ then set $K \leftarrow K^*$;
- If $\mathsf{flag} = \mathsf{fresh}$, there is a record $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw)$, and $\mathcal{F}_{\mathsf{pwKE}}$ sent (sid, K') to P' when record $(\mathsf{sid}, \mathsf{P}', \mathsf{P}, pw)$ was fresh, then set $K \leftarrow K'$;
- In any other case set $K \xleftarrow{\mathrm{r}} \{0,1\}^\kappa$.

Mark record $(\mathsf{sid}, \mathsf{P}, \mathsf{P}', pw)$ as $\mathsf{completed}$ and send (sid, K) to P.

Fig. 23: $\mathcal{F}_{\mathsf{pwKE}}$: UC symmetric PAKE functionality (multi-session version)

Password Registration

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$ from S create record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked fresh.

Stealing Password Data

- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is no record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, return "no password file". Otherwise mark this record $\mathsf{compromised}$, and if there is a record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ then send pw to $\mathcal{A}$.
- On $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ from $\mathcal{A}$, then do:
  - If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$, do the following: If $pw^* = pw$ then return "correct guess" to $\mathcal{A}$ else return "wrong guess."
  - Else record $\langle \mathsf{offline}, \mathsf{S}, uid, pw^* \rangle$

Password Authentication

- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ from C, if there is no record $\langle \mathsf{sid}, \mathsf{C}, ... \rangle$ then record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh and send $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$.
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ from S, if there is no record $\langle \mathsf{sid}, \mathsf{S}, ... \rangle$ then retrieve record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, and if it exists then create record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, pw, 1 \rangle$ marked fresh and send $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ to $\mathcal{A}$.

Active Session Attacks

- On $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ marked fresh, then do: If $pw^* = pw$ then mark it $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."
- On $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh, then do: If there is a record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$ then mark $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if there is a record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ not marked $\mathsf{completed}$, then do:
  - If rec is marked $\mathsf{compromised}$ set $K \leftarrow K^*$;
  - Else if $\mathsf{role} = 0$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 1 \rangle$ s.t. $\mathcal{F}_{\mathsf{aPAKE}}$ sent (sid, K') to P' while that record was marked fresh, set $K \leftarrow K'$;
  - Else if $\mathsf{role} = 1$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 0 \rangle$ which is marked fresh, pick $K \xleftarrow{\mathrm{r}} \{0,1\}^\ell$;
  - Else set $K \leftarrow \perp$.

  Finally, mark rec as $\mathsf{completed}$ and send output (sid, K) to P.

Fig. 24: $\mathcal{F}_{\mathsf{aPAKE}}$: UC asymmetric PAKE with explicit C-to-S authentication