## **Efficient Asymmetric PAKE Compiler from KEM and AE**

You Lyu<sup>1,2</sup><sup>0</sup>, Shengli Liu<sup>1,2</sup><sup>( $\otimes$ </sup>)<sub>2</sub>, and Shuai Han<sup>2,3</sup>

<span id="page-0-0"></span><sup>1</sup> Department of Computer Science and Engineering Shanghai Jiao Tong University, Shanghai 200240, China <sup>2</sup> State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China <sup>3</sup> School of Cyber S[cien](https://orcid.org/0000-0002-8148-3643)ce and Engineerin[g, S](https://orcid.org/0000-0003-1366-8256)hanghai Jiao Tong [U](https://orcid.org/0000-0002-8156-7089)niversity, Shanghai 200240, China {vergil,slliu,dalen17}@sjtu.edu.cn

**Abstract.** Password Authenticated Key Exchange (PAKE) allows two parties to establish a secure session key with a shared low-entropy password *pw*. Asymmetric PAKE (aPAKE) extends PAKE in the clientserver setting, and the server only stores a password file instead of the plain password so as to provide additional security guarantee when the server is compromised.

In this paper, we propose a novel generic compiler from PAKE to aPAKE in the Universal Composable (UC) framework by making use of Key Encapsulation Mechanism (KEM) and Authenticated Encryption  $(AE)$ .

- **–** Our compiler admits efficient instantiations from lattice to yield lattice-based post-quantum secure aPAKE protocols. When instantiated with Kyber (the standardized KEM algorithm by the NIST), the performances of our compiler outperform other lattice-based compilers (Gentry et al. CRYPTO 2006) in all aspects, hence yielding the most efficient aPAKE compiler from lattice. In particular, when applying our compiler to the UC-secure PAKE schemes (Santos et al. EUROCRYPT 2023, Beguinet et al. ACNS 2023), we obtain the most efficient UC-secure aPAKE schemes from lattice.
- **–** Moreover, the instantiation of our compiler from the tightly-secure matrix DDH (MDDH)-based KEM (Pan et al. CRYPTO 2023) can compile the tightly-secure PAKE scheme (Liu et al. PKC 2023) to a tightly-secure MDDH-based aPAKE, which serves as the first tightly UC-secure aPAKE scheme.

**Keywords:** Password authenticated key exchange; asymmetric password authenticated key exchange; lattice; post-quantum security

## **1 Introduction**

<span id="page-0-1"></span>Password Authenticated Key Exchange (PAKE) facilitates a secure establishment of session keys between two parties, say a client and a server, over a public network using a low entropy password *pw*. These session keys can then be used to set up secure communication channels for the client and server. In contrast to Authenticated Key Exchange (AKE) protocols, PAKE operates with easily memorable passwords and eliminates the need for reliance on Public Key Infrastructure (PKI), hence offering a more convenient deployments of secure channels.

Note that the adversary can always implement online invocations of the PAKE protocol with guessing passwords. The low entropy of the passwords might make such online attacks succeed with non-negligible probability. So the security of PAKE requires these online attacks be the only efficient attacks.

For PAKE, a password must be shared between the client and server. Generally, the password is memorable so it is not necessary for the client to store it. However, the server has to store passwords for all the clients. If the server is compromised, the adversary can snatch the password *pw* from the server so as to impersonate the client. To address this vulnerability, asymmetric PAKE (aPAKE) [15] was proposed. With an aPAKE, the server only has to store a password file, which is a one-way function value *H*(*pw*) of the password *pw*. Now if the server is compromised, the adversary only obtains the password file instead of the plain password. Therefore, aPAKE is deemed preferable to PAKE in the clie[nt-s](#page-30-0)erver setting. In this paper, we focus on aPAKE.

For aPAKE, if the adversary obtains  $H(pw)$ , it is possible for him/her to implement a so-called offline attack: compute or pre-compute  $(pw', H(pw'))$ pairs to check the correctness of guessing password *pw′* by testing whether  $H(pw') = H(pw)$ . Clearly, offline attack is inevitable after  $H(pw)$  is obtained by the adversary. As such, the security of aPAKE requires these offline attacks on  $H(pw)$  be the only efficient attacks that may work (beyond the online attacks).

**Security Models for aPAKE.** There are two types of security notions for aPAKE, the game-based security in the Indistinguishability model (IND security) [3,4] and the simulation-based security under the Universally Composable framework (UC security) [15]. The UC framework is preferable to the IND model in the following important aspects.

- T[h](#page-29-0)[e U](#page-29-1)C framework permits arbitrary correlations and distributions for passwords, while the IN[D m](#page-30-0)odel typically assumes passwords uniformly distributed over a dictionary for the sake of security proofs.
- Protocols that are proven secure in the UC framework enable a smooth and secure composition with other UC-secure protocols, thanks to the universal composition theorem [8].

Therefore, UC framework offers a more robust security guarantee and facilitates modular design of complex protocols, making it a better security model than the IND model.

**aPAKE from Post-Quantum Assumptions.** Now there is a trend to migrate cryptographic algorithms to post-quantum ones. In fact, NIST has already determined Kyber [7] as the Post-Quantum Cryptography (PQC) winner for Key Encapsulation Mechanism (KEM), and Dilithium [11], Falcon [14], Sphincs<sup>+</sup>[5] for digital signature schemes. Recently, the Crypto Forum Research Group (CFRG) in IETF [31] initiated a selection process for both PAKE and aPAKE protocols. As far as we know, none of existing candidates are based on post-quantum assumptions.

There do exist a few aPAKE compilers instantiable from post-quantum assumption[s. O](#page-31-0)ne compiler is the  $\Omega$ -method proposed by Gentry et al. [15], which can compile any UC-secure PAKE protocol into a UC-secure aPAKE protocol with the help of a signature scheme. Accordingly, the existing UC-secure PAKE protocol from lattices [29,2] and post-quantum secure signature schemes (like Dilithium [11], Falcon [14], Sphincs<sup>+</sup> [5]) admit UC-secure aPAKE [pr](#page-30-0)otocols. Another compiler is due to McQuoid and Xu [24], which can compile any UCsecure PAKE protocol into a strong aPAKE protocol. This strong aPAKE is secure against pre-com[put](#page-31-1)[at](#page-29-2)ion attacks (in idealized generic group action model [12]), but t[heir](#page-29-3) compiler [can](#page-29-4) only be inst[an](#page-29-5)tiable from group actions (e.g. isogenybased CSIDH) rather than lattices. Note that [the](#page-30-1)re also exists an aPAKE compiler from UC-secure Key-Hiding AKE protocols [17,30,29]. However, no such key-hiding AKE protocols is instantiable from post-quantum assumptions. So [up](#page-29-6) to now this compiler does not lead to post-quantum secure aPAKE yet.

In summary, the two UC-secure aPAKE compilers in [15,24] seem to be the only available approaches to aPAKE achieving post[-qu](#page-30-2)[an](#page-31-2)[tum](#page-31-1) security. However, either compiler has its own limitations.

- The compiler in [24] can only be instantiated from i[sog](#page-30-0)[eny](#page-30-1)-based assumptions, rather than the mainstream lattice-based assumptions. Besides, the isogeny-based group action does not offer good computational efficiency under the current parameter configuration.
- The compiler in [[15\]](#page-30-1) can be instantiated from lattice-based assumptions with the help of lattice-based signature schemes. However, it is not very efficient since current lattice-based signature schemes (like Dilithium [11] in the NIST PQC winner) will lead to much higher computational overhead compared to their KEM coun[terp](#page-30-0)arts (like Kyber [7] in the NIST PQC winner) under the same security level. Technically speaking, this might be mainly due to the time-consuming process of trapdoor sampling or rejection sa[mp](#page-29-3)ling involved in most lattice-based signature schemes, dating back to the seminal works of [16,23].

This motivates us to seek a new aPAKE compiler instantiable from lattice to answer the following question:

> *[H](#page-30-3)ow to construct an efficient UC-secure aPAKE protocol from lattices, without using signatures?*

**Our Contribution.** In this paper, we address the aforementioned question through the following contributions.

*•* **UC-secure aPAKE compiler without using signatures.** We propose a novel generic compiler that can compile any UC-secure PAKE protocol to a UC-secure aPAKE protocol, with the help of weak primitives of KEM and

Authenticated Encryption (AE). KEM is only required to have One-Wayness under Plaintext-Checkable Attacks (OW-PCA)[25], weaker than the standard Indistinguishability under Chosen-Ciphertext Attacks (IND-CCA). AE is only required to have one-time CCA security and one-time authenticity, which has efficient information-theoretical instantiations. In particular, our compiler does not rely on signature schemes.

*•* **Mutual explicit authentication & good round efficiency.** In addition to its excellent efficiency, our compiler enjoys other desirable features. By using our compiler, the resulting aPAKE enjoys mutual explicit authentication even if the underlying PAKE does not. This is similar to the compiler in [15] but in sharp contrast to [24], which completely relies on the underlying PAKE to achieve mutual explicit authentication.

Moreover, our compiler only augments two additional rounds to the underlying PAKE. If the last round message of PAKE is sent from the server, [the](#page-30-0)n the first additional rou[nd](#page-30-1) of our compiler can be merged to the last round of PAKE, and hence only one additional round is needed for aPAKE. This is also similar to the compiler in  $[15]$  and comparable to  $[24]$ <sup>4</sup>

- *•* **The most efficient UC-secure aPAKE from lattice.** By instantiating the KEM in our compiler with Kyber, the NIST PQC KEM winner, we obtain a lattice-based compiler. The performance evaluations show that our Kyber-based compiler outperforms the [sig](#page-30-0)nature-based compile[r in](#page-30-1) [\[](#page-3-0)15] from lattice in all aspects: ours save at least 61.1%-63.8% computing time and 41.5%-84.3% communication cost. Our Kyber-based compiler turns out to be the most efficient aPAKE compiler from lattice, and also results in *the most efficient UC-secure aPAKE schemes from lattice* when applyi[ng t](#page-30-0)o the UC-secure PAKE schemes in [29,2].
- *•* **The first tightly UC-secure aPAKE.** By instantiating the KEM in our compiler with the tightly-secure matrix DDH (MDDH)-based KEM scheme proposed in [26], we obtain a tightness-preserving compiler, which yields *the first tightly UC-secure aPAK[E s](#page-31-1)[ch](#page-29-2)eme* when applying the compiler to the tightly-secure PAKE scheme in [22].

**Technique Ov[erv](#page-30-4)iew.** Our aPAKE compiler mainly relies on a KEM with OW-PCA security to compile a PAKE to an aPAKE protocol. Here OW-PCA [25] is weaker than IND-CCA securi[ty](#page-30-5) and asks one-wayness of ciphertext even if the adversary has access to a "plaintext-checking" oracle which on input a ciphertext-key pair  $(c, K)$  determines whether *c* is an encapsulation of *K*.

Our compiler also requires a one-time secure Authenticated Encryption (AE) [and](#page-30-6) five hash function  $H_0, \cdots, H_4$ . Here one-time security of AE includes onetime authenticity and one-time CCA security. We note that one-time secure AE has very efficient information-theoretical instantiations, e.g., from one-time pad and one-time secure message authentication code (MAC). On the other hand,

<span id="page-3-0"></span><sup>&</sup>lt;sup>4</sup> More precisely, the compiler in [24] incurs an additional round from server to client before PAKE. Since it is usually the client who starts the protocol, actually two additional rounds are needed before PAKE in [24].

the hash functions will be modelled as Random Oracles (RO) during the security proof.

We refer to Fig. 1 for a high-level description of our compiler. More precisely, our compiler works as follows.

<span id="page-4-0"></span>

Fig. 1: High level description of our aPAKE compiler from KEM and AE.

The server's password file contains a hash value  $H_0(pw)$  of the password  $pw$ and a KEM public key *pk*, which is derived from the password *pw* by using *H*<sub>1</sub>(*pw*) as the randomness of key generation, i.e., (*pk*, *sk*) ← KeyGen(*H*<sub>1</sub>(*pw*)).

To establish a session key sKey, the client and server equip the underlying PAKE with password  $rw = H_0(pw)$ , and run the PAKE protocol to derive a PAKE session key  $K_0$ .

Next, the server will use KEM to distribute a key  $K_1$  to the client with the help of *pk* (retrieved from its password file): it first invokes  $(c, K_1) \leftarrow \text{Encap}(pk)$ but it does not send *c* directly to the client. Instead, the server will use the PAKE session key  $K_0$  to derive a key  $H_2(K_0)$  for AE. Then with the help of AE, the server further encrypts the encapsulation *c* to obtain an AE ciphertext  $\psi$  via  $\psi \leftarrow$  AE.Enc( $H_2(K_0), c$ ), and sends  $\psi$  to the client. In this way, the (one-time) authenticity of AE helps the server authenticate itself to the client since only the server and client can compute  $K_0$  via PAKE and generate valid ciphertext  $\psi$ (i.e., the decryption of which does not lead to rejection) under AE key  $H_2(K_0)$ .

After receiving  $\psi$ , the client checks the validity of  $\psi$  with the AE key  $H_2(K_0)$ . If  $\psi$  is valid, i.e.,  $\mathsf{AE}.\mathsf{Dec}(H_2(K_0), \psi) \neq \bot$ , the client will set the session key sKey :=  $H_4(K_0)$ . It also recovers the KEM secret key *sk* from *pw* via  $(pk, sk) \leftarrow$ KeyGen( $H_1(pw)$ ), and then double decrypts  $\psi$  with AE key  $H_2(K_0)$  and KEM secret key *sk* to obtain  $K_1$ . Finally, the client computes an authenticator  $\sigma$  :=  $H_3(K_0, K_1, Trans|\psi)$  where *Trans* is the transcription of PAKE, and sends  $\sigma$  to the server. Note that only the server and client can compute  $K_0$  and  $K_1$ . In this way, the confidentiality of KEM and AE (i.e., OW-PCA of KEM and one-time CCA of AE) help the client authenticate itself to the server.

Finally, the server checks the validity of  $\sigma$  with  $(K_0, K_1)$  by testing whether  $\sigma = H_3(K_0, K_1, Trans|\psi)$ , where  $Trans, \psi, \sigma$  constitute the transcription seen by the server. If  $\sigma$  is valid, the server's session key is set to sKey :=  $H_4(K_0)$ .

Roughly speaking, our aPAKE compiler is designed in a "PAKE plus passwordbased mutual authentication" manner. PAKE helps the client and server share an AE key  $H_2(K_0)$  and then AE helps the authentication of the server. Meanwhile, the server's message  $\psi$  can be viewed as a "challenge" to the client. Only the client knowing the correct password *pw* is able to derive the KEM secret key *sk* to recover  $K_1$  and generate the correct "response"  $\sigma$  with  $(K_0, K_1)$ . Putting differently,  $\sigma$  can be viewed as a proof of knowledge of  $pw$  for the client. As long as adversary *A* does not obtain *pw* through offline or online attacks, *A* cannot impersonate the client even if *A* compromises the server and obtains (*rw, pk*).

**UC security in RO model.** To prove the UC security for our aPAKE compiler in the random oracle (RO) model, we have to construct a simulator Sim to simulate all the interaction transcripts and session keys for both passive and active attacks implemented by *A*, without the knowledge of password *pw* but accessing to the ideal functionality  $\mathcal{F}_{apake}$  for aPAKE (see Fig. 4 in Sect. 2.3).

The aPAKE protocol has two parts: the PAKE part in which  $rw = H_0(pw)$ is used to generate PAKE key  $K_0$ , and the last two rounds in which  $K_0$  and  $(pk, sk, rw)$  are used to generate  $\psi$ ,  $\sigma$  and the session keys skey. Recall that PAKE has UC security, so the PAKE part can emulate the id[ea](#page-12-0)l functio[nali](#page-11-0)ty  $\mathcal{F}_{\text{pake}}$  (see Fig. 3 in Sect. 2.3 for the description of  $\mathcal{F}_{\text{pake}}$ ).

Firstly, let us consider *A*'s attacks on the PAKE part. The task of Sim is to simulate the correctly distributed PAKE key  $K_0$  without password  $pw$ . Note that if *A* did not use the correct PAKE password  $rw := H_0(pw)$  in the PAKE part, *A*'s activ[e](#page-11-1) attack o[n t](#page-11-0)he PAKE part hardly succeeds, due to the ideal functionality  $\mathcal{F}_{\text{pake}}$  of PAKE. We consider the following three cases.

- **Case I:** *A***'s successful active attacks on the PAKE part with the correct password** *pw***.** In this case, *A* successful implements either an online attack or an offline attack, in both of which *A* must have queried  $H_0(pw)$ to obtain *rw*. For *A*'s such an attack with *rw*, Sim can search the hash list to find  $pw'$  such that  $rw = H_0(pw')$ , and then justify the correctness of  $pw'$ with the help of Testpw interface of  $\mathcal{F}_{apake}$ . In this way, Sim is able to extract the correct password  $pw$  from  $\mathcal{A}$ 's successful active attacks. Then with  $pw$ , it can simulate the PAKE part to generate PAKE key  $K_0$  for  $A$ .
- **Case II:** *A***'s successful active attacks on the PAKE part with the stolen PAKE password** *rw***.** In this case, *A* must have stolen the password file (*rw, pk*) and can use *rw* to impersonate client or server in the PAKE part of the aPAKE protocol. Sim can simulate the generation of password file for *A* with random elements. More precisely, Sim picks *rw* and *r* randomly, sets  $H_0(?) := rw$  and  $H_1(?) := r$  respectively with  $pw = ?$  undetermined, and generates  $(pk, sk) \leftarrow \text{KeyGen}(r)$ . If later *A* attacks with correct *pw*, Sim can extract it (see Case I) and then re-program the hash function with  $H_0(pw) := rw$  and  $H_1(pw) := r$ . With the knowledge of *rw*,

it can simulate  $\mathcal{F}_{\text{pake}}$  to generate PAKE key  $K_0$  for  $\mathcal{A}$ . We stress that Sim generates (*pk, sk, rw*) without *pw* in this case.

- **Case III: the other cases.** If neither Case I nor Case II happens, then either *A*'s active attack fails or *A* implements a passive attack.
	- **Case III.1: Failed active attack on the PAKE part.** In this case, either the PAKE part outputs *⊥* to abort the protocol (PAKE with explicit authentication), or the PAKE part outputs different PAKE session keys  $K_0$  and  $K'_0$  for Client and Server respectively (PAKE with implicit authentication), where  $K_0$  and  $K'_0$  must be uniform to  $A$  according to the UC security of PAKE.
	- **Case III.2: Passive attack on the PAKE part.** In this case, the PAKE part generates the shared PAKE session key  $K_0$  for Client and Server. According to the UC security,  $K_0$  must be uniform to  $A$ .
	- In this case, Sim can simulate  $K_0$  (also  $K'_0$ ) with a randomly chosen one.

Next, let us consider *A*'s attacks on the last two rounds of the aPAKE protocol. We follow the above three cases.

- **Case I happened.** In this case, Sim has extracted the correct password *pw* and also knows PAKE key  $K_0$ . With the knowledge of  $pw$  and  $K_0$ , Sim can give a perfect simulation for *A*'s active attacks (here passive attack is impossible due to the previous active attack on the PAKE part) with round message *ψ ′* or *σ ′* . Note that Sim can use the correct *pw* to invoke *Fapake* to assign the session key sKey determined by *A* to both client and server.
- **Case II happened.** In this case, Sim has the knowledge of (*pk, sk, rw*) itself (generated without  $pw$ ) and the PAKE key  $K_0$ . also uniformly distributed.
	- For *A*'s active attack with  $\psi'$ , Sim can determine the validity of  $\psi'$  with *K*<sub>0</sub>. It rejects  $\psi'$  and aborts the protocol for invalid  $\psi'$ , and for valid  $\psi'$ it invokes *Fapake* to assign the session key sKey determined by *A* to both client and server (dealing with the case that *A* stole the file (*pk, rw*) and successfully impersonated server to accomplish the aPAKE protocol).
	- For *A*'s active attack with *σ ′* , Sim can determine the validity of *σ ′* with  $K_0$  and  $K_1$ . If  $\sigma'$  is valid and  $A$  has obtained the correct password pw via an offline attack, it invokes *Fapake* to assign the session key sKey determined by *A* to both client and server. If  $\sigma'$  is invalid, it rejects  $\sigma'$ and aborts the protocol. Define Bad as the event that  $\sigma'$  is valid but *A* did not obtain *pw* from offline attack. If Bad happens, it rejects *σ ′* as well. However in the real experiment, in case of Bad,  $\sigma'$  should be accepted. This imperfect simulation does not annoy us since Bad can hardly happen, which is guaranteed by the OW-PCA security of KEM. The reason is as follows. Without *pw*, *A* cannot derive *sk*, then *A* cannot obtain  $K_1$  due to the OW-PCA security, hence  $H_3(K_0, K_1, Trans|\psi)$  is uniform to *A* and  $\sigma' = H_3(K_0, K_1, Trans|\psi)$  hardly holds.
- **Case III happened.** In this case, Sim has the knowledge of  $K_0$  which is uniformly distributed in A's view. Then to A, the hash values of  $H_2(K_0)$ ,  $H_3(K_0)$ ,  $K_1, Trans|\psi)$ ,  $H_4(K_0)$  are also uniformly distributed.
- For *A*'s passive attacks on server or client, Sim can simulate  $\psi$  with  $\psi \leftarrow$  AE.Enc( $H_2(K_0)$ , dummy message) and simulate  $\sigma$  with a uniform string, and invoke  $\mathcal{F}_{apake}$  to generate random session key sKey for server or client. Given uniform AE key  $H_2(K_0)$ , AE encryption of a dummy message is indistinguishable from AE encryption of the real message *c*, due to the one-time CCA security of AE. Meanwhile random oracles guarantee the uniformity of  $\sigma := H_3(K_0, K_1, Trans|\psi)$  and sKey :=  $H_4(K_0)$ . Moreover, Sim invokes  $\mathcal{F}_{apake}$  to generate a uniform session key sKey for both client and server. Therefore, Sim's simulation is perfect for these passive attacks.
- For  $\mathcal{A}$ 's active attacks on client with  $\psi'$ , Sim directly rejects and aborts the protocol. Without the knowledge of AE key  $H_2(K_0)$ ,  $\mathcal{A}$ 's forgery of  $\psi'$ will result in  $\mathsf{AE}\text{-}\mathsf{Dec}(H_2(K_0), \psi') = \bot$ , due to the one-time authenticity of AE. So Sim's simulation of dealing with  $\psi'$  is perfect (except with negligible probability).
- For *A*'s active attacks on client with *σ ′* , Sim directly rejects and aborts the protocol. Without the knowledge of KEM's secret key  $sk$ ,  $K_0$ ,  $A$ 's  $\text{forgery of } \sigma' \text{ can hardly collide with the valid } \sigma := H_3(K_0, K_1, Trans|\psi),$ due to uniformity of the RO outputs. So Sim's simulation of dealing with  $\psi'$  is perfect (except with negligible probability).

**Comparison.** In Table 1, we instantiate our aPAKE compiler with the NIST PQC KEM winner algorithm Kyber, and compare it with the signature-based compiler [15], which is the only known aPAKE compiler instantiable from lattices prior to our work. The performance results in Table 1 is obtained by running experiments on Intel(R) [C](#page-8-0)ore(TM)  $i7-1068NG7$  CPU  $@$  2.30GHz with 4 cores under macOS 13.3.1. When the compiler [15] is instantiated with the NIST PQC lattice-ba[sed](#page-30-0) signature scheme Dilithium or Falcon, our compiler significantly outperforms theirs in all aspects including registrati[on](#page-8-0) time, computing time, server storage size, and communication cost. This suggests that our compiler is the most efficient aPAKE compiler fro[m la](#page-30-0)ttices. As far as we know, there is no UC-secure aPAKE scheme constructed directly from lattices, and the only approach to UC-secure aPAKE from lattices is via compiling a PAKE scheme to an aPAKE. Therefore, the good efficiency of our compiler suggests that applying our compiler to the available UC-secure PAKE schemes from lattices will yield the most efficient UC-secure aPAKE schemes from lattices.

We also compare our compiler with known aPAKE compilers [15,24] instantiated from other post-quantum assumptions (see Table 3 in Appendix  $D$ ). The comparison shows that our compiler overwhelms these compilers in terms of registration time and computing time, and has comparable server storage and communication cost.

In Table 2, we also compare the resulting MDDH-base[d](#page-38-0) aPAKE sche[me](#page-38-1) (from our MDDH-based compiler) with other aPAKE schemes in terms of tight UC security. The tight security of aPAKE in [22,30,32,19,24] is eliminated by the impossibility result in  $[22]$ , which essentially said that tightly UC-security is impossible t[o](#page-8-1) achieve if the secret derived from password by client is uniquely <span id="page-8-0"></span>Table 1: Efficiency comparison of our Kyber-based compiler with the other aPAKE compilers [15] from lattices, where Kyber is in its version of NIST security level 3. For computing and communication cost, we use the mode of "**Client + Server = Total**" to reflect the seperation of client/server computation/communication. The aPAKE compiler in [15] has two lattice-based instantiations, one i[s in](#page-30-0)stantiated with Dilithium in its version of NIST security level 3, and the other is instantiated with Falcon in its version of NIST security level 1 (due to its lack of level 3).

aPAKE Compiler	Registration Time (ms)	Computing Time (ms)	Server  Storage (KB)	Communication Cost (KB)	Assumption Type
$\vert$ [15] (Dilithium)	0.096	$0.245 + 0.126 = 0.371$	5.875	$3.215 + 3.938 = 7.153$	Lattice-based
$[15]$ (Falcon)	7.003	$0.271 + 0.074 = 0.345$	2.189	$0.642 + 1.282 = 1.924$ Lattice-based	
Ours $(Kyber)$	0.017	$ 0.051 + 0.083 = 0.134 $	1.188	$ 0.031 + 1.094 = 1.125 $ Lattice-based	

[Tab](#page-30-0)le 2: Comparison of our MDDH-based aPAKE scheme with other UC-secure aPAKE scheme in terms of tight UC security, where our aPAKE scheme is resulted from compiling the PAKE protocol [22] with our MDDH-based compiler. "?" denotes its tight UC security is unknown.

<span id="page-8-1"></span>

determined by the password file and moreover [the](#page-30-5) r[ela](#page-31-2)ti[on b](#page-31-3)e[twe](#page-30-7)e[n th](#page-30-1)e [sec](#page-30-0)re[t a](#page-30-2)nd file can be efficiently determined. In contrast to their schemes, the client secret (*rw, sk*) in our MDDH-based aPAKE evades the impossibility result since the key generation of MDDH-based KEM makes sure that multiple *sk* correspond to one *pk* and hence to one password file. On the other hand, the aPAKE schemes  $[15,17]$  in Table 2 are obtained by compiling PAKE + signature in  $[15]$  and AKE  $+$  ideal cipher in [17] respectively, but the security reduction is not tight, so it is unknown whether tight UC security can be achieved.

**The Necessity of AE in Our Compiler.** Note that AE plays an important [rol](#page-30-0)[e in](#page-30-2) our co[mp](#page-8-1)iler for aPAKE to achieve explicit authenticati[on.](#page-30-0) Moreover, AE is especially [nece](#page-30-2)ssary in our compiler when the underlying KEM has no anonymity under secret key leakage. No anonymity under secret key leakage means that given *sk*, one can efficiently decide whether a KEM ciphertext *c* is generated under *pk*. If we take off AE from the round messge  $\psi$ , the server will send the KEM ciphertext *c* directly, and then the resulting aPAKE will suffer from an offline attack: the adversary sees the encapsulation *c*, tries different password *pw*, generates  $(pk, sk) \leftarrow \mathsf{KeyGen}(H_1(pw))$ , and tests whether *c* is the KEM ciphertext generated under key pair (*pk, sk*).

As far as we know, many lattice-based KEMs have no anonymity under secret key leakage. Let us take Regev's encryption scheme [28] as an example. For a ciphertext *c* generated under *pk*, the decryption of *c* using *sk* will result in a value which is either very close to 0 or close to *q/*2 with *q* the modulus. However, for *c*

generated under another public key *pk′* , the decryption of *c* using *sk* may result in a value far from both 0 and  $q/2$ . In this way, one can in fact efficiently tell whether *c* is generated under *pk* (at least with noticeable probability). Consequently, AE is necessary for our lattice-based compiler.

On the other hand, if the underlying KEM does have anonymity under secret key, which means the adversary cannot distinguish a given ciphertext *c* is generated under  $(pk_1, sk_1)$  or  $(pk_2, sk_2)$ , the above offline attack fails. However, the price of getting rid of AE from our compiler is that we lose the explicit authentication of server-side. Actually, the CDH-based aPAKE compiler [19] can be seen as an instantiation of our compiler from ElGamal KEM without AE, and of course the CDH-based aPAKE does not achieve explicit mutual authentication.

**Hardness on Achieving UC-Secure aPAKE in QROM.** Note that our aPAKE compiler is proved under ROM instead of Quantum Ran[dom](#page-30-7) Oracle Model (QROM) [6]. QROM allows adversary *A* to perform superposition queries to random oracles, making it hard for simulator to extract preimages or reprogram random oracles. It is especially hard to give UC-secure aPAKE protocols in QROM since reprogrammable random oracles are unavoidable for the construction of UC-[se](#page-29-7)cure aPAKE protocols, as demonstrated in [18]. Therefore, it is a big challenge to construct UC-secure aPAKE in QROM, and we leave it as an interesting open problem.

## **2 Preliminary**

<span id="page-9-0"></span>Let  $\lambda \in \mathbb{N}$  denote the security parameter throughout the paper. If x is defined by *y* or the value of *y* is assigned to *x*, we write  $x := y$ . For  $\mu \in \mathbb{N}$ , define  $[\mu] := \{1, 2, \ldots, \mu\}$ . Denote by  $x \leftarrow s \mathcal{X}$  the procedure of sampling x from set  $\mathcal X$  uniformly at random. Let  $|\mathcal X|$  denote the number of elements in  $\mathcal X$ . All our algorithms are probabilistic unless stated otherwise. We use  $y \leftarrow \mathcal{A}(x)$  to define the random variable *y* obtained by executing algorithm  $A$  on input  $x$ . We also use  $y \leftarrow \mathcal{A}(x; r)$  to make explicit the random coins r used in the probabilistic computation. The notation "*≈s*" represents statistical indistinguishability, while "*≈c*" denotes computational indistinguishability. "PPT" abbreviates probabilistic polynomial-time. Denote by negl some negligible function.

In Subsect. 2.1, we recall the definitions of Key Encapsulation Mechanism (KEM), Authentication Encryption (AE) and their security notions. In Subsect. 2.2 and Subsect. 2.3, we present the ideal functionalities of random oracle and (a)PAKE, respectively.

#### **[2.1](#page-11-2) KEM and [AE](#page-11-0)**

**Definition 1 (KEM).** *A key encapsulation mechanism (KEM) scheme* KEM = (KeyGen*,* Encap*,* Decap) *consists of three algorithms:*

<span id="page-9-1"></span>*-* KeyGen(*r*) : *Taking as input a randomness r ∈ R , the key generation algorithm outputs a pair of public key and secret key* (*pk, sk*)*.*

- *-* Encap(*pk*) : *Taking as input a public key pk, the encapsulation algorithm outputs a pair of ciphertext*  $c \in \mathcal{CT}$  *and encapsulated key*  $K \in \mathcal{K}$ *.*
- *-* Decap(*sk, c*) : *Taking as input a secret key sk and a ciphertext c, the decapsulation algorithm outputs*  $K \in \mathcal{K}$ *.*

The correctness of KEM requires that

$$
\Pr\left[ \begin{matrix} r \leftarrow_{\$} \mathcal{R}, (pk, sk) \leftarrow \mathsf{KeyGen}(r) \text{ : } \mathsf{Decap}(sk, c) = K \end{matrix} \right] = 1 - \mathsf{negl}(\lambda).
$$

**Definition 2 (OW-PCA security for KEM).** *For a KEM scheme* KEM = (KeyGen*,* Encap*,* Decap)*, the advantage function of an adversary A is defined by*

$$
\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}}(\mathcal{A}) := \Pr\left[\begin{matrix}r \leftarrow_{\$} \mathcal{R}, (pk, sk) \leftarrow \mathsf{KeyGen}(r), \\ (c, K) \leftarrow \mathsf{Encap}(pk) \\ K' \leftarrow \mathcal{A}^{ \mathsf{CHECK}(sk, \cdot, \cdot)}(pk, c)\end{matrix}\right] , \right.\right.
$$

*where the oracle Check takes as input a ciphertext-key pair* (*c, K*) *and returns whether*  $Decap(sk, c) = K$  *or not. The OW-PCA security for KEM requires*  $\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow-pca}}(\mathcal{A}) = \mathsf{negl}(\lambda) \text{ for all } PPT \mathcal{A}.$ 

**Definition 3 (AE).** *An authenticated encryption (AE) scheme* AE = (Enc*,* Dec) *consists of two algorithms:*

- *-* Enc(*k, m*) : *Taking as input a key k ∈ K and a message m ∈ M, the encryption algorithm outputs a ciphertext c.*
- $\text{Dec}(k, c)$ : *Taking as input a key*  $k \in \mathcal{K}$  *and a ciphertext c, the decryption algorithm outputs a message*  $m \in \mathcal{M}$  *or a symbol*  $\perp$ *.*

The correctness of AE requires that  $\text{Dec}(k, \text{Enc}(k, m)) = m$  holds for all  $k \in \mathcal{K}$ and all  $m \in \mathcal{M}$ .

**Definition 4 (One-time authenticity for AE).** *For an AE scheme* AE = (Enc*,* Dec)*, the advantage function of an adversary A for one-time authentic-* $\int$  *ity is defined by*  $\mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot-auth}}(\mathcal{A}) := \Pr\left[k \leftarrow s \mathcal{K}, c \leftarrow \mathcal{A}^{\mathsf{ENC}(k, \cdot)} : \mathsf{Dec}(k, c) \neq \bot \land k$ *c is not the output of oracle*  $\mathsf{Enc}(k, \cdot)$ . *The one-time authenticity for AE requires*  $\mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot} \text{-} \mathsf{auth}}(\mathcal{A}) = \mathsf{negl}(\lambda) \text{ for all PPT } \mathcal{A} \text{ that query the oracle } \mathsf{ENC}(k, \cdot) \text{ at most }$ *once.*

**Definition 5 (One-time IND-CCA security for AE).** *For an AE scheme* AE = (Enc*,* Dec)*, the advantage function of an adversary A for one-time CCA se*curity is defined by  $\mathsf{Adv}^{\text{ot-cca}}_{\mathsf{AE}}(\mathcal{A}) := \big| \Pr \big[ k \leftarrow s \mathcal{K}, b \leftarrow s \{0, 1\}, (m_0, m_1) \leftarrow \mathcal{A}^{\text{DEC}(k, \cdot)},$  $c_b \leftarrow \text{Enc}(k, m_b), b' \leftarrow A^{Dec(k, \cdot)}(c_b) : b = b' \right] - 1/2$ . The one-time IND-CCA security requires  $\mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot}\text{-}\mathsf{cca}}(\mathcal{A}) = \mathsf{negl}(\lambda)$  *for all PPT A that query the oracle*  $Dec(k, \cdot)$  *at most once for a ciphertext different from*  $c_b$ .

Such an AE can be easily constructed from one-time pad and one-time secure Message Authentication Code (MAC), and hence it has a very efficient information-theoretical instantiation. See Sect. 4 for the detailed construction.

#### **2.2 Idealized Random Oracle Model**

<span id="page-11-2"></span>The idealized functionality of random oracle is shown in Fig. 2. The idealized functionality of random oracle can be perfectly simulated by a PPT simulator Sim: Sim maintains a list  $\mathcal L$  (initialized to be empty) to store the records. Upon receiving a query (Eval, x) from *P*, Sim checks whether there exists  $(x, y) \in \mathcal{L}$ . If yes, Sim returns *y*. Otherwise, Sim randomly samples  $y \leftarrow s \mathcal{Y}$  $y \leftarrow s \mathcal{Y}$  $y \leftarrow s \mathcal{Y}$ , records  $(x, y)$ in the list  $\mathcal{L}$ , and returns  $y$ .

<span id="page-11-3"></span>**Ideal Functionality** *FRO* The ideal functionality  $\mathcal{F}_{RO}$  is parameterized by a random map  $H: \{0,1\}^* \to \mathcal{Y}$ . **Upon receiving a query** (Eval*, x*) **from** *P***:** If  $H(x)$  is defined: return  $H(x)$ . Else:  $y \leftarrow \in \mathcal{Y}$ ,  $H(x) := y$ , return  $H(x)$ .

Fig. 2: The ideal functionality  $\mathcal{F}_{RO}$  for random oracle  $H$ .

#### **2.3 (Asymmetric) PAKE under UC Framework**

<span id="page-11-0"></span>In Fig. 3 and Fig. 4, we show the ideal functionalities of PAKE and aPAKE respectively. Here the functionalities mainly follows from [15] along with some modifications as did in [32]. The ideal functionality of aPAKE is extended from that of PAKE, so we only explain aPAKE below.

<span id="page-11-1"></span>

Functionality $\mathcal{F}_{nake}$				
The functionality $\mathcal{F}_{\text{packet}}$ is parameterized by a security parameter $\lambda$ . It interacts with an adversary A				
and a set of parties via the following queries:				
<b>Upon receiving a query</b> (NewClient, $C^{(i)}$ , <i>iid</i> , $S^{(j)}$ , <i>pw</i> ) from $C^{(i)}$ :				
Send (NewClient, $C^{(i)}$ , <i>iid</i> , $S^{(j)}$ ) to A. Record $(C^{(i)}, id, S^{(j)}, pw)$ and mark it as fresh.				
<b>Upon receiving a query</b> (NewServer, $S^{(j)}$ , <i>iid'</i> , $C^{(i)}$ , <i>pw</i> ) from $S^{(j)}$ :				
Send (NewServer, $S^{(j)}$ , iid', $C^{(i)}$ ) to A. Record $(S^{(j)}, \text{iid}', C^{(i)}, \text{pw})$ and mark it as fresh.				
<b>Upon receiving a query</b> (Testpw, P, iid, pw') from $\mathcal{A}$ :				
If there is a <b>fresh</b> record $(P, \text{iid}, \cdot, \text{pw})$ :				
If $pw' = pw$ , mark the record <b>compromised</b> and reply to A with "correct guess".				
If $pw' \neq pw$ , mark the record <b>interrupted</b> and reply A with "wrong guess".				
<b>Upon receiving a query</b> (NewKey, P, <i>iid</i> , <i>sid</i> , Key <sup>*</sup> ) from $A$ :				
If there is a compromised record $(P, iid, Q, pw)$ , send $(sid, Key^*)$ to P.				
If there is a fresh record $(P, id, Q, pw)$ , there is a <b>completed</b> record $(Q, id', P, pw')$ with $pw = pw'$ ,				
$\mathcal{F}_{\text{make}}$ has sent (sid, Key') to Q, and (Q, iid', P, pw') was fresh when (sid, Key') was sent, then				
output $(sid, \text{Key}')$ to P.				
In any other case, pick a new random key $\text{Key} \leftarrow \{0,1\}^{\lambda}$ and send $(sid, \text{Key})$ to P.				
In all cases, mark the record $(P, iid, Q, pw)$ as <b>completed</b> .				

Fig. 3: The ideal functionality  $\mathcal{F}_{\text{pake}}$  for PAKE.

**Security guarantees in the ideal world.**  $\mathcal{F}_{apake}$  in Fig. 4 formalizes the ideal functionality of asymmetric PAKE with mutual explicit authentication.

### **Functionality**  $\bar{F}$

 $\overline{\phantom{a}}$ 

<span id="page-12-0"></span>

$\boldsymbol{u}$ <i>and</i> $\boldsymbol{u}$ <i>apake</i> The functionality $\mathcal{F}_{apake}$ is parameterized by a security parameter $\lambda$ . It interacts with an adversary $\mathcal{A}$
and a set of parties via the following queries:
Password Storage
Upon receiving a query (StorePWFile, $C^{(i)}$ , $S^{(j)}$ , $pw$ ) from $S^{(j)}$ :
Record (file, $C^{(i)}$ , $S^{(j)}$ , pw), mark it as fresh, and send (StorePWFile, $C^{(i)}$ , $S^{(j)}$ ) to A.
<b>Stealing Password File</b>
Upon receiving a query (StealPWFile, $C^{(i)}$ , $S^{(j)}$ ) from A:
Mark record (file, $C^{(i)}$ , $S^{(j)}$ , pw) as <b>compromised</b> , and send (StealPWFile, $C^{(i)}$ , $S^{(j)}$ ) to A.
If there is a record (offline, $C^{(i)}, S^{(j)}, pw'$ ) with $pw' = pw$ : send pw and "correct guess" to A.
Upon receiving a query (OfflineTestPW, $C^{(i)}$ , $S^{(j)}$ , $pw'$ ) from A:
If there exists a record (file, $C^{(i)}$ , $S^{(j)}$ , pw) marked <b>compromised</b> :
If $pw' = pw$ , return "correct guess", else return "wrong guess".
Else: Store (offline, $C^{(i)}$ , $S^{(j)}$ , pw').
<b>Sessions</b>
Upon receiving a query (NewClient, $C^{(i)}$ , iid, $S^{(j)}$ ) from $C^{(i)}$ :
Retrieve the record (file, $C^{(i)}$ , $S^{(j)}$ , pw).
Send (NewClient, $C^{(i)}$ , iid, $S^{(j)}$ ) to A. Record $(C^{(i)},iid, S^{(j)}, pw)$ and mark it as fresh.
Upon receiving a query (NewServer, $S^{(j)}$ , iid', $C^{(i)}$ ) from $S^{(j)}$ :
Retrieve the record (file, $C^{(i)}$ , $S^{(j)}$ , $pw$ ). Send (NewServer, $S^{(j)}$ , $iid'$ , $C^{(i)}$ ) to A.
Record $(S^{(j)},iid, C^{(i)}, pw)$ and mark it as fresh.
<b>Active Session Attacks</b>
Upon receiving a query (Testpw, $P$ , <i>iid</i> , $pw'$ ) from $A$ :
If there is a fresh record $(P, id, \cdot, pw)$ :
If $pw' = pw$ , mark the record <b>compromised</b> and reply to A with "correct guess".
If $pw' \neq pw$ , mark the record <b>interrupted</b> and reply to A with "wrong guess".
Upon receiving a query (Impersonate, $C^{(i)}$ , iid) from A:
If there exists a record (file, $C^{(i)}$ , $S^{(j)}$ , pw) marked <b>compromised</b> and a record $(C^{(i)},iid, S^{(j)}, pw)$
marked fresh:
mark the record $(C^{(i)},iid, S^{(j)}, pw)$ compromised and return "correct guess" to A.
Otherwise, mark the record $(C^{(i)},iid, S^{(j)}, pw)$ interrupted and return "wrong guess" to A.
<b>Key Generation</b>
Upon receiving a query (FreshKey, $P$ , iid, sid) from $A$ :
If there is a fresh record $(P, id, Q, pw)$ and sid has never been assigned to P's any other instance:
Pick sKey $\leftarrow$ {0,1} <sup><math>\lambda</math></sup> , mark the record ( <i>P</i> , <i>iid</i> , <i>Q</i> , <i>pw</i> ) as <b>completed</b> , return ( <i>sid</i> , sKey) to <i>P</i> , and
record $(P, Q, sid, sKey)$ .
Upon receiving a query (CopyKey, $P$ , iid, sid) from $A$ :
If there is a fresh record $(P, id, Q, pw)$ , a <b>completed</b> record $(Q, id', P, pw)$ and sid has never
been assigned to $P$ 's any other instance:
If there exists record $(Q, P, sid, sKey)$ and the record $(Q, iid', P, pw)$ was fresh when Q receives
(sid, sKey): Mark the record $(P, id, Q, pw)$ as <b>completed</b> , and return (sid, sKey) to P.
Upon receiving a query (CorruptKey, $P$ , iid, sid, sKey) from $A$ :
If there is a compromised record $(P, id, Q, pw)$ and sid has never been assigned to P's any other
instance: Mark the record $(P, iid, Q, pw)$ as <b>completed</b> , and return (sid, sKey) to P. Upon receiving a query (Abort, $P$ , <i>iid</i> ) from $A$ .
Mark the record $(P, iid, \cdot, pw)$ as <b>abort</b> , and return $(\perp, \perp)$ to P.

Fig. 4: The ideal functionality  $\mathcal{F}_{apake}$  for a<br>PAKE.

Roughly speaking, the ideal functionally  $\mathcal{F}_{apake}$  in the UC model captures the following security guarantees in the ideal world.

- **– Passive attack and forward security.** For a passive attacker *A*, the session key is uniformly distributed (which is modeled by the FreshKey and CopyKey query), even if the adversary is given the password.
- **– Online guessing password attack.** In this case, the adversary *A* guesses a password *pw′* once for a session and tests whether his guess is correct, which is modeled by the Testpw query. This means that any active attack implemented by a real-world adversary  $A$  to the protocol can be translated to a single Testpw query in the ideal world. If the password-guess is correct, then the adversary can control the session key at its will (which is modeled by the CorruptKey query). Otherwise, the attacked party can detect the active attack and reject this session directly (which modeled by the Abort query).
- **– Online impersonation attack on server after stealing the password file.** In this case, the adversary *A* has obtained the password file stored in the server. Without loss of generality, we write the password file as  $H(pw)$ . With  $H(pw)$ ,  $A$  can impersonate a server perfectly to communicate with a client, which is modeled by the Impersonate query. Besides online attack, *A* can also implement offline attacks by choosing a password *pw′* and checking whether  $H(pw') = H(pw)$ , which is modeled by the OfflineTestPW query. Moreover, before *A* obtains the password file  $H(pw)$ , *A* can precompute a table of possible pairs (*pw′ , H*(*pw′* )). When *A* obtains the password file, he can search *pw* in the table quickly by comparing  $H(pw') = H(pw)$ . This is also captured by the OfflineTestPW. Note that in this case, *A* cannot impersonate a client unless *A* makes a correct online password guess, or *A* obtains the password via offline attacks, i.e. *A* gets a  $pw'$  s.t.  $H(pw') = H(pw)$ .

**UC security for aPAKE.** In the real world, the environment  $Z$  has all passwords for all users, controls the adversary  $A$ , and sees all the interactions over the channel and session keys derived from the protocol. In UC framework, we will construct a PPT simulator Sim which has access to the ideal functionality  $\mathcal{F}_{apake}$  and interacts with the environment  $\mathcal{Z}$ . If the view simulated by Sim for environment  $Z$  is indistinguishable to  $Z$ 's view in the real world, then UC security is achieved.

Difference between our  $\mathcal{F}_{\text{pake}}/\mathcal{F}_{\text{apake}}$  and the original  $\mathcal{F}_{\text{pake}}/\mathcal{F}_{\text{apake}}$ . The original ideal functionality  $\mathcal{F}_{\text{pake}}$  for PAKE was proposed in [9]. The ideal functionality  $\mathcal{F}_{apake}$  for aPAKE in [15] was built upon [9]. Shoup [32] improved and optimized both  $\mathcal{F}_{\text{pake}}$  and  $\mathcal{F}_{\text{apake}}$  in several ways.

- In the original ideal functionality  $\mathcal{F}_{\text{pake}}$  and  $\mathcal{F}_{\text{apake}}$ , each prot[oc](#page-29-8)ol instance must be identified by a global[ly](#page-30-0) unique session i[de](#page-29-8)ntifier an[d t](#page-31-3)he PAKE participants can successfully establish a joint session key only if they use the same session identifier *sid*. The requirements on *sid* are problematic for an implementer, as pointed out by [1]. Shoup's modified  $\mathcal{F}_{\text{pake}}$  addresses this issue by replacing globally unique session identifiers with locally unique

instance identifiers (*iid*), and session identifiers (*sid*) are seen as protocol outputs.

- The *Fapake* in [15] is flawed, as pointed out by [18]. In fact, Shoup also pointed it out, and he fixed these flaws with a modified  $\mathcal{F}_{apake}$ .
- *Fapake* in [32] models the explicit authentication (via fresh-key, copy-key, corrupt-key and abort interfaces), while the formulation of explicit authentication in [15] i[s fl](#page-30-0)awed, as pointed out by Remar[k 1](#page-30-8)0 in [32].

In our paper, [we](#page-31-3) adopt the optimized  $\mathcal{F}_{\text{pake}}$  and  $\mathcal{F}_{\text{apake}}$  provided in [32].

*Remark 1.* In this paper, we assume that each instance of a party (client or server) is initia[lize](#page-30-0)d with the different instance ID *iid*. This i[s r](#page-31-3)easonable and also taken in [32] (see Sect. 4.3). Another notation is that for simplicity, our ideal functionality  $\mathcal{F}_{apake}$  does not model the case that client mistypes its [pa](#page-31-3)ssword, but this can be easily taken into account with the same approach in [32] (See Remark 3 and Remark 4 in [32] for more details).

### **3 Our aPAKE Compiler from KEM and AE**

<span id="page-14-1"></span>In this section, we propose [an](#page-31-3) aPAKE compiler from KEM and AE, and show how to construct aPAKE from UC-secure PAKE with the help of our compiler in the UC framework. The detailed compiler (as well as the aPAKE construction) is shown in Fig. 5. Due to its UC security, PAKE can emulate its ideal functionality  $\mathcal{F}_{\text{pake}}$ , so we replace PAKE with  $\mathcal{F}_{\text{pake}}$  in the aPAKE construction.

Clearly, the resulting aPAKE scheme from our compiler is correct if the underlying PAKE, KEM and AE scheme are correct. If the underlying KEM has OW-PC[A](#page-15-0) security and AE has one-time authenticity and one-time CCA security, then our compiler is able to compile a UC-secure PAKE to a UC-secure aPAKE, as shown in the following theorem.

<span id="page-14-0"></span>**Theorem 1.** *If* KEM *is a key encapsulation mechanism with OW-PCA security,* AE *is an authenticated encryption scheme with one-time authenticity and onetime CCA security,*  $H_0, H_1, H_2, H_3, H_4$  *work as random oracles, then the aPAKE scheme in Fig.* 5 *securely emulates*  $\mathcal{F}_{apake}$ *, hence achieving UC security in the {Fpake,FRO}-hybrid model. More precisely, suppose there are at most N parties, ℓ sessions and q random oracle queries, then there exists a simulator* Sim *s.t.*

$$
\begin{aligned} \left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}\right]-\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}\right]\right| \leq & N^2\ell\cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}}(\mathcal{B}_{\mathsf{KEM}}) + \frac{q^2+q+1}{2^\lambda} \\ & + \ell\cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot\text{-}auth}}(\mathcal{B}_{\mathsf{AE}}) + \ell\cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot\text{-}ccc}}(\mathcal{B}_{\mathsf{AE}}). \end{aligned}
$$

Proof. The main objective of the proof is constructing a PPT simulator Sim to simulate an indistinguishable view for the environment *Z*. Sim is designed to have access to the ideal functionality  $\mathcal{F}_{apake}$  and interact with the environment *Z*, thereby emulating the real-world aPAKE protocol interactions involving the adversary  $A$ , the parties, and the environment  $Z$ . It is important to note that Sim *does not possess any password*.

<span id="page-15-0"></span>

Fig. 5: Construction of UC-secure aPAKE from UC-secure PAKE with our compiler.

Let  $\text{Real}_{\mathcal{Z},\mathcal{A}}$  represent the real-world experiment where the environment  $\mathcal{Z}$ interacts with the parties and adversary  $A$  who can access ideal functionality  $\mathcal{F}_{\text{pake}}$  via Testpw and NewKey. Let  $\text{Ideal}_{\mathcal{Z},\text{Sim}}$  represents the ideal experiment where  $\mathcal{Z}$  interacts with "dummy" parties and simulator Sim. By  $\text{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1$ , we mean  $\mathcal Z$  outputs 1 in  $\text{Real}_{\mathcal Z, \mathcal A}$ , and  $\text{Ideal}_{\mathcal Z, \text{Sim}} \Rightarrow 1$  is similarly defined.

Our goal is to show that  $|Pr[\text{Real}_{Z,A} \Rightarrow 1] - Pr[\text{Ideal}_{Z,\text{Sim}} \Rightarrow 1]|$  is negligible by employing a series of games, denoted as Game  $G_0-G_7$ . In this sequence,  $G_0$ corresponds to  $\text{Real}_{\mathcal{Z},\mathcal{A}}$ , while  $G_7$  corresponds to  $\text{Ideal}_{\mathcal{Z},\text{Sim}}$ . We aim to show that these adjacent games are indistinguishable from the view of *Z*. For simplicity, we write  $H_0(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$  and  $H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$  as  $H_0(pw)$  and  $H_1(pw)$  in the security proof.

**Game**  $G_0$ . This is the real experiment  $\text{Real}_{\mathcal{Z},\mathcal{A}}$ . In this experiment,  $\mathcal{Z}$  initializes a password for each client-server pair, sees the interactions among clients, servers, ideal functionality  $\mathcal{F}_{\text{pake}}$  and adversary  $\mathcal{A}$ , and also obtains the corresponding session keys of protocol instances. During the execution,  $\mathcal{F}_{\text{pake}}$  may be invoked to create records like "(*P, iid, Q, rw*)" which we call *inner records* so as to distinguish them from the records created by  $\mathcal{F}_{angle}$ . Here A may implement attacks like view, modify, insert, or drop messages over the network. In  $\mathsf{G}_0$ ,  $H_0, H_1, H_2, H_3, H_4$  works as random oracles. Each party will do the following.

 $-$  For a server  $S^{(j)}$  on input (StorePWFile,  $C^{(i)}, S^{(j)}, pw$ ) from  $\mathcal{Z}$ , it computes  $rw := H_0(pw)$  and  $(pk, sk) \leftarrow \mathsf{KeyGen}(H_1(pw))$  and then stores  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)},$ *rw, pk*) locally.

- $-$  For a server  $S^{(j)}$  on input (StealPWFile,  $C^{(i)}, S^{(j)})$  from *A*, it retrieves  $(C^{(i)}, S^{(j)})$  $(S^{(j)}, rw,pk)$  from local storage, and returns  $(rw,pk)$  to A.
- $-$  For a client instance  $(C^{(i)},iid)$  on input (NewClient,  $C^{(i)},iid, S^{(j)}, pw)$  from Z, it computes  $rw := H_0(pw)$  and issues "(NewClient,  $C^{(i)}$ , *iid*,  $S^{(j)}$ ,  $rw$ )" query to ideal functionality  $\mathcal{F}_{\text{pake}}$ . According to the specification of  $\mathcal{F}_{\text{pake}}$ , it will create a fresh inner record "(NewClient,  $C^{(i)}$ , *iid*,  $S^{(j)}$ ,  $rw$ )", and send "(NewClient,  $C^{(i)}, iid, S^{(j)})$ " to  $\mathcal{A}.$

If  ${\cal A}$  issues "(NewKey,  ${\sf C}^{(i)},iid, sid, \text{Key}^*)$ " to  $\mathcal{F}_{pake},$  then the instance  $({\sf C}^{(i)},iid)$ may receive " $(sid, \text{Key} = K_0)$ " from  $\mathcal{F}_{\text{pake}}$ .

 $-$  For a server instance  $(S^{(j)},iid')$  on input (NewServer,  $S^{(j)}, iid', C^{(i)}$ ) from  $\mathcal{Z},$ it retrieves  $(C^{(i)}, S^{(j)}, rw, pk)$  from its storage, and sends "(NewServer,  $S^{(j)}$ , *iid'*,  $C^{(i)}$ ,  $rw$ )" to  $\mathcal{F}_{\text{pake}}$ . According to the specification of  $\mathcal{F}_{\text{pake}}$ , it will create a fresh inner record " $(S^{(j)},iid', C^{(i)}, rw)$ ", and send "(NewServer, S<sup>(j)</sup>,  $iid', C^{(i)})$ " to  $\mathcal{A}.$ 

If  ${\cal A}$  issues "(NewKey,  ${\sf S}^{(j)},iid', sid, \text{Key}^*)$ " to  $\mathcal{F}_{pake},$  then the instance  $({\sf S}^{(j)},iid')$ may receive " $(sid, \text{Key} = K_0)$ " from  $\mathcal{F}_{\text{pake}}$ .

- **–** *A* can access *Fpake* via two interfaces (Testpw*, P, iid, pw*) and (NewKey*, P, iid, sid,*Key*∗* ).
	- *•* Upon *Fpake* receiving "(NewKey*, P, iid*, *sid,*Key*<sup>∗</sup>* )" from *A*, *Fpake* may create " $(sid, \text{Key} = K_0)$ " according to the specification in Fig. 3, mark the inner record " $(P, iid, Q, rw)$ " as completed, and send " $(sid, Key =$  $K_0$ )" to the instance " $(P, iid)$ ".
	- *•* Upon *Fpake* receiving "(Testpw*, P, iid, rw′* )" query from *A*, it checks whether  $rw' = H_0(pw)$ . If yes,  $\mathcal{F}_{\text{pake}}$  returns "correct guess" to *[A](#page-11-1)*. Otherwise,  $\mathcal{F}_{\text{pake}}$  returns "wrong guess" to A. Meanwhile,  $\mathcal{F}_{\text{pake}}$  marks the inner record " $(P, iid, Q, rw)$ " as compromised or interrupted accordingly (See Fig. 3).
- $-$  Upon a server instance  $(S^{(j)},iid')$  receiving message  $(sid, K_0)$  from  $\mathcal{F}_{pak}$ , it retrieves " $(C^{(i)}, S^{(j)}, rw,pk)$ " from its storage, and computes  $(c, K_1) \leftarrow$ Encap(*pk*). Then it computes  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), c)$  and sends  $\psi$  to A.
- $-$  Upon a clien[t i](#page-11-1)nstance  $(C^{(i)},iid)$  receiving message " $(sid, K_0)$ " from  $\mathcal{F}_{\textit{pake}}$ and message  $\psi$  from *A*, it first decrypts  $c \leftarrow$  AE.Dec( $H_2(K_0), \psi$ ). If  $c =$ *⊥*, then it rejects and sends *⊥* to *Z*. Otherwise, it computes (*pk, sk*) *←* KeyGen( $H_1(pw)$ ) and decrypts  $K_1 \leftarrow \mathsf{Decap}(sk, c)$ . Finally, it computes  $\sigma :=$  $H_3(K_0, K_1, sid|\psi)$ , sends  $\sigma$  to  $\mathcal{A}$ , sets sKey :=  $H_4(K_0)$  and returns sKey to *Z*.
- $-$  Upon a server instance  $(S^{(j)}, \text{iid}')$  receiving message  $\sigma$  from  $\mathcal{A}$ , if  $\sigma \neq H_3(K_0, \mathcal{A}')$ *K*<sub>1</sub>*, sid* $|\psi\rangle$ , then it rejects and sends  $\perp$  to *Z*. Otherwise, it sets sKey :=  $H_4(K_0)$  and returns sKey to  $\mathcal{Z}$ .

We have

$$
\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1].
$$

**Game**  $G_1$  (simulations for clients and servers with  $pw$ ). In this game, we introduce a simulator Sim who *additional knows passwords* and has access to the ideal functionality  $\mathcal{F}_{apake}$ . Now in Game  $\mathsf{G}_1$ , the client and server become "dummy party" and directly forward their inputs to the ideal functionality  $\mathcal{F}_{apake}$  defined in Fig. 4. Then Sim simulates the behaviors of clients and servers with the help of *pw* as follows.

- $-$  For a dummy server  $S^{(j)}$  on input (StorePWFile,  $C^{(i)}, S^{(j)},\, pw)$  from  $\mathcal{Z},$  it directly sends this query to  $\mathcal{F}_{apake}$ . Then  $\mathcal{F}_{apake}$  sends (StorePWFile,  $\mathsf{C}^{(i)}, \mathsf{S}^{(j)}$ ) to Sim. Then Sim [s](#page-12-0)imulates the password file  $(rw, pk)$  with  $rw := H_0(pw)$ ,  $(pk, sk) \leftarrow \mathsf{KeyGen}(H_1(pw)).$  Sim also stores a trapdoor record  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, rw,$  $pk, sk, H_1(pw)$  in its local storage.
- $-$  For a dummy server  $S^{(j)}$  on input (StealPWFile,  $C^{(i)}, S^{(j)})$  from *A*, Sim directly sends this query to  $\mathcal{F}_{apake}$ . Then  $\mathcal{F}_{apake}$  functions as described in Fig.  $4$  and sends (StealPWFile,  $C^{(i)}, S^{(j)}$ ) to Sim. Then Sim returns the password file  $(rw, pk)$  to  $\mathcal{A}$ .
- $-$  For a dummy client instance  $(C^{(i)},iid)$  on input (NewClient,  $C^{(i)},iid, S^{(j)}, pw$ ), it directly sends this query to  $\mathcal{F}_{apake}.$  Then  $\mathcal{F}_{apake}$  sends (NewClient,  $\mathsf{C}^{(i)},iid,$  $S^{(j)}$  $S^{(j)}$  to Sim and Sim simulates the behavior of the client instance  $(C^{(i)},iid)$ *with the password pw*, just like  $G_0$ .
- $-$  For a dummy server instance  $(S^{(j)},iid')$  on input (NewServer,  $S^{(j)},iid, C^{(i)}$ ), it directly sends this query to  $\mathcal{F}_{apake}.$  Then  $\mathcal{F}_{apake}$  sends (NewServer, S<sup>(j)</sup>, iid<sup>'</sup>,  $C^{(i)}$  to Sim and Sim simulates the behavior of the server instance  $(S^{(j)},iid')$ *with the PAKE password*  $rw$ , just like  $G_0$ .
- **–** For dummy client and server instances, the generations of *ψ*, *σ* and sKey are all simulated by Sim *with the password pw*, just like  $G_0$ .

With the knowledge of *passwords*, the simulations of the behaviors of all clients and servers are perfect.

Moreover, Sim also simulates random oracles  $H_i$  ( $i \in [0, 4]$ ) maintaining separate lists, namely  $\mathcal{L}_{H_i}$ . For a query *x* on  $H_i(\cdot)$ , if  $(x, y) \in \mathcal{L}_{H_i}$ , then Sim will return *y* as the reply. Otherwise, Sim will choose a random element *y*, record  $(x, y)$  in  $\mathcal{L}_{H_i}$ , and return *y* as the reply.

During the simulation, Sim additionally checks a bad event: if there exists two different random oracle queries to  $H_i$  such that  $H_i(pw) = H_i(pw')$ , then Sim will abort the game. By the ideal functionality of random oracles, Sim's simulations for oracles  $H_i$  are perfect except a collision occurs in the simulation of  $H_i$ . Suppose that the adversary issues  $q$  random oracle queries totally, by the union bound, we have

$$
\left|\Pr\left[\mathsf{G}_1 \Rightarrow 1\right] - \Pr\left[\mathsf{G}_0 \Rightarrow 1\right]\right| \le \frac{q^2}{2^\lambda}.
$$

The following games will change the simulations of Sim step by step in an indistinguishable way so that Sim can arrive at its final form in Fig. 6 and Fig. 7 in Appendix A, and accomplish the simulations in  $\text{Ideal}_{\mathcal{Z},\text{Sim}}$  without passwords *pw*.

**Game**  $G_2$  (simulation for ideal functionality  $\mathcal{F}_{\text{pake}}$  without  $pw$ ). In  $G_2$ , simulator Sim will simulate the ideal functionality *Fpake* itself, but [w](#page-32-0)ithout t[he](#page-33-0)

knowledge of *pw*. More precisely, Sim will maintain some inner records to simulate the output of  $\mathcal{F}_{\text{pake}}$  in the following way.

- $\bf{Upon receiving}$  (NewClient,  $C^{(i)},iid, S^{(j)})$   ${\bf from}$   $\mathcal{F}_{apake}$ : Sim sends (NewClient,  $(C^{(i)}, iid, S^{(j)})$  to *A*. Then it stores " $(C^{(i)}, iid, S^{(j)}, ?)$ " as an inner record and marks it as **fresh**.
- **– Upon receiving input** (NewServer*,* S (*j*) *, iid′ ,* C (*i*) ) **from** *Fapake*: Sim sends (NewServer,  $S^{(j)}$ , *iid'*,  $C^{(i)}$ ) to *A*. Then it stores " $(S^{(j)}, iid', C^{(i)}, ?)$ " as an inner record and marks it as **fresh**.
- **– Upon receiving a query** (Testpw*, P, iid, rw*) **from** *A*: If there exists a **fresh** inner record " $(P, iid, Q, ?)$ ", then do the following.
	- 1. If there exists  $(pw, rw) \in \mathcal{L}_{H_0}$ , then Sim sends (Testpw, P, iid, pw) to  $\mathcal{F}_{apake}$  and forwards  $\mathcal{F}_{apake}$ 's reply ("correct guess" or "wrong guess") to *A*.
	- 2. If *A* ever issued (StealPWFile,  $C^{(i)}$ ,  $S^{(j)}$ ) query before and Sim returned  $(rw', pk')$  to *A* with  $rw' = rw$ , and  $(P, iid)$  is an instance among the interaction of  $C^{(i)}$  and  $S^{(j)}$ , then Sim returns "correct guess" to A.
	- 3. In other cases, return "wrong guess".
	- 4. If Sim returns "correct guess" to  $A$ , it also replaces " $(P, \text{iid}, Q, ?)$ " with " $(P, \text{iid}, Q, rw)$ " and marks it as a **compromised** inner record. Otherwise, it marks "(*P, iid, Q,* ?)" as an **interrupted** inner record.
- **– Upon receiving a query** (NewKey*, P, iid, sid,* Key*∗* ) **from** *A*: The simulator replies the query just like  $\mathcal{F}_{\textit{pake}}$ .
	- 1. If *sid* has been assigned to *P*'s any other instance  $(P, iid')$ , return  $\perp$ .
	- 2. If there exists a **compromised** inner record " $(P, \text{iid}, Q, rw)$ ", then output (*sid,*Key*∗* ) to *P*.
	- 3. If there exists a **fresh** inner record "(*P, iid, Q,* ?)" and a **completed** inner record " $(Q, iid', P, ?)$ ", " $(sid, \text{Key}')$ " was sent to  $Q$  and  $(Q, iid', P, ?)$ was **fresh** at the time, then output "(*sid,*Key*′* )" to *P*.
	- 4. In any other case, pick a new random key  $Key \leftarrow s \{0,1\}^{\lambda}$  and send "(*sid,*Key)" to *P*.

Finally, mark the inner record " $(P, iid, Q, \cdot)$ " as **completed**.

It is easy to see the simulation of NewClient*,* NewServer*,* NewKey query is perfect. The only difference in  $G_1$  and  $G_2$  occurs in the simulation of Testpw query.

Note that in G1, (Testpw*, P, iid, rw*) returns "correct guess" if and only if *rw* is the PAKE password used in the instance (*P, iid*), which is the case that  $rw = H_0(pw)$  for the input *pw* to party *P*. Therefore,  $G_2$  and  $G_1$  differs only when *A* issues a (Testpw, *P*, *iid*, *rw*) query to  $\mathcal{F}_{\text{pake}}$ ,  $rw = H_0(pw)$  but *A* does not query  $H_0(pw)$ . By the ideal functionality of random oracle,  $H_0(pw)$  is uniformly distributed to *A* if *A* does not query it. So we have

$$
\left|\Pr\left[\mathsf{G}_2 \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1 \Rightarrow 1\right]\right| \leq \frac{1}{2^{\lambda}}.
$$

Game  $G_3$  (replace hash values of  $K_0$  with uniform ones unless  $A$  im**plements a successful on-line active attacks on**  $\mathcal{F}_{\text{pake}}$ **). We consider the**  following three cases, where Case I and Case II cover *A*'s successful on-line active attacks on  $\mathcal{F}_{\text{pake}}$ , and Case III covers the rest.

- **Case I.** *A* implements a (successful) on-line attack on  $\mathcal{F}_{\text{pake}}$  with the correct password *pw*. That is, *A* issues "(Testpw, *P*, *iid*,  $rw = H(pw)$ )" query, and then Sim obtains "correct guess" from  $\mathcal{F}_{apake}$  for its query "(Testpw, P, iid, pw)" to *Fapake*. In this case, Sim is able to extract the correct password *pw*.
- **Case II.** *A* implements a (successful) on-line attack on  $\mathcal{F}_{\text{pake}}$  by impersonating a party with the stolen PAKE password *rw*. That is *A* first issues a query (StealPWFile*, P*, *Q*), then issues "(Testpw*, P, iid, rw*)" or "(Testpw*, Q, iid, rw*)" query. In this case, *A* has stolen the PAKE password *rw* and successfully implements an on-line impersonation attack.
- **Case III.** Neither Case I nor Case II occurs.

If Case I or Case II happens,  $G_3$  is the same as  $G_2$ . But if Case III happens in G3, when *A* issues a "(NewKey*, P, iid,*Key*<sup>∗</sup>* )" query resulting in *K*0, then the hash values of  $H_2(K_0), H_3(K_0, K_1, c)$  and  $H_4(K_0)$  are replaced with independently and uniformly chosen elements, no matter whether *A* has ever queried any of them or not.

It is easy to see that  $G_3$  is the same as  $G_2$  unless  $A$  has ever queried  $H_2(K_0)$ ,  $H_3(K_0, K_1, c)$  or  $H_4(K_0)$  in Case III.

Note that Case III means that either there is no on-line attacks from  ${\mathcal A}$  or the on-line attack does not succeed. Upon *A* issuing a "(NewKey*, P, iid,*Key*<sup>∗</sup>* )" query to  $\mathcal{F}_{\text{make}}$  in Case III, then the resulting key  $K_0$  simulated by Sim is uniformly distributed to *A* (according to the specification of the simulation of ideal functionality  $\mathcal{F}_{\text{pake}}$ , the session key  $K_0$  should be uniform). Suppose the adversary totally issues  $q$  random oracle queries, then  $A$  ever issues hash query  $H_2(K_0), H_3(K_0, K_1, c)$  or  $H_4(K_0)$  with correct  $K_0$  with probability at most  $q/2^{\lambda}$ . So we have

$$
|\Pr\left[\mathsf{G}_3 \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2 \Rightarrow 1\right]| \le \frac{q}{2^{\lambda}}.
$$

Now in Case III, the AE key  $H_2(K_0)$ , the authenticator  $\sigma = H_3(K_0, K_1, c)$ and the session key  $sKey = H_4(K_0)$  will be uniformly distributed to A. Jumping ahead, the uniform AE key  $H_2(K_0)$  in Case III paves the way for the security reduction to the security of  $AE$  in  $G_4$  and  $G_5$ .

Now in G3, Sim does not need *pw* to simulate the generation of PAKE session key *K*0: upon *A*'s (NewKey*, P, iid,*Key*<sup>∗</sup>* ) query, Sim just sets *K*<sup>0</sup> := Key*<sup>∗</sup>* in Case I and Case II, and can choose *K*<sup>0</sup> uniformly in Case III. But Sim still needs *rw* to identify Case II and needs password *pw* to generate password file. Moreover, the generations of  $\psi$  and  $\sigma$  also needs  $pw: (pk, sk) \leftarrow \mathsf{KeyGen}(H_1(pw)), (c, K_1) \leftarrow$  $\mathsf{Encap}(pk)$ ,  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), c)$ , and  $\sigma = H_3(K_0, K_1 = \mathsf{Decap}(sk, c), sid|\psi)$ . Game  $G_4$  (Simulation of generating server's message  $\psi$  without  $pw$ ).

In  $G_4$ , Sim is the same as in  $G_3$ , except for Sim's generation of  $\psi$  for the server instance  $(S^{(j)},iid')$ . We describe  $Sim's$  simulations in the three cases (which are defined in  $G_3$ ) in  $G_4$ .

- **–** If Case I occurs to (S (*j*) *, iid′* ), then Sim does not need to know *pw* beforehand. Instead, Sim can extract the true password *pw* of server  $S^{(j)}$  from  $\mathcal{F}_{apake}$ . Then Sim can use *pw* to generate  $(pk, sk) \leftarrow \mathsf{KeyGen}(H_1(pw)), (c, K_1) \leftarrow$ Encap( $pk$ ) and  $\psi \leftarrow$  AE.Enc( $H_2(K_0), c$ ), exactly like  $G_3$ .
- **–** If Case II occurs to (S (*j*) *, iid′* ), then Sim does not need the knowledge of *pw* to generate  $\psi$ . In this case, Sim directly retrieves *pk* from its trapdoor record  $(C^{(i)}, S^{(j)}, rw,pk,sk,r)$ , and then compute  $(c, K_1) \leftarrow$  Encap $(pk)$  and generates  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), c)$ . Simulation of  $\psi$  is exactly the same as  $G_3$ .
- **–** If neither Case I nor Case II occurs to (S (*j*) *, iid′* ), then Sim does not need the knowledge of  $pw$  to generate  $\psi$  either. In this case, Sim directly retrieves *pk* from its trapdoor record  $(C^{(i)}, S^{(j)}, rw,pk,sk,r)$  and then computes  $(c, K_1) \leftarrow \text{Encap}(pk)$  and  $\psi \leftarrow \text{AE}.\text{Enc}(H_2(K_0), 0)$ . Accordingly, when its partnered client instance  $(C^{(i)},iid)$  receives this specific  $\psi$  (passive attack with  $\psi$ ), it directly uses  $K_1$  corresponding to  $\psi$  (for consistence) and computes  $\sigma := H_3(K_0, K_1, sid|\psi)$ . Recall that in  $\mathsf{G}_3, \psi \leftarrow \mathsf{A}\mathsf{E}.\mathsf{Enc}(H_2(K_0), c)$ rather than  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), 0).$

Finally, simulator Sim records  $(S^{(j)}, \text{iid}', K_1)$  in Case I and Case II and records  $(S^{(j)}, \text{iid}', \perp)$  in Case III.

The difference between  $G_3$  and  $G_4$  lies in the generation of  $\psi$  in Case III:  $ψ$  ← AE.Enc( $H_2(K_0), c$ ) in G<sub>3</sub> but  $ψ$  ← AE.Enc( $H_2(K_0), 0$ ) in G<sub>4</sub>. In Case III,  $H_2(K_0)$  is uniformly distributed and independent of  $\mathcal{A}$ 's view, as shown in  $\mathsf{G}_3$ . According to the one-time IND-CCA security of AE and hybrid arguments over the (at most) *ℓ* ciphertexts of AE, we have

$$
\left|\Pr\left[\mathsf{G}_4 \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3 \Rightarrow 1\right]\right| \leq \ell \cdot \mathsf{Adv}^{\text{ot-cca}}_{\mathsf{AE}}(\mathcal{B}_{\mathsf{AE}}).
$$

Note that the reduction needs to query the decryption oracle once to simulate the message  $\sigma$  when it receives  $\psi$  generated by *A*. This is why we need AK have one-time IND-CCA security.

Now in G4, Sim only needs *rw* to distinguish Case II and *pk* to simulate the generation of  $\psi$  in Case II. Besides Sim also uses *pk* to generate *c*,  $K_1$  in Case II, III, and uses *sk* to generate  $\sigma$  in Case II, III. But it still needs *pw* to generate (*rw, pk, sk*).

**Game** G<sup>5</sup> **(Simulation of client's message** *σ* **and session key** sKey **without** *pw*). In G<sub>5</sub>, Sim is the same as in G<sub>4</sub>, except for Sim's generation of  $\sigma$  and sKey for the client instance  $(C^{(i)},iid)$  when receiving  $\psi$ . We describe Sim's simulations in the three cases in  $G_5$ .

 $-$  **Case I** occurs to  $(C^{(i)},iid)$ : *A* successfully guesses *pw* by issuing "(Testpw,  $C^{(i)}$ , *iid, rw* =  $H_0(pw)$ )" **query to**  $\mathcal{F}_{pake}$ . In this case, Sim can extract the true password  $pw$  from  $\mathcal{F}_{apake}$  and simulate the generation of  $\sigma$  in the same way as  $G_4$ . More precisely, Sim first decrypts  $c \leftarrow \mathsf{AE}.\mathsf{Dec}(H_2(K_0), \psi)$ .  $−$  If  $c = \bot$ , then Sim issues (Abort, C<sup>(i)</sup>, iid) query to  $\mathcal{F}_{apake}$ . As a result,  $\mathcal{F}_{apake}$  returns  $(L, L)$  to  $(\mathsf{C}^{(i)}, iid)$  to reject the session. In this case the session is rejected in both  $\mathsf{G}_4$  and  $\mathsf{G}_5.$ 

– Otherwise, Sim generates (*pk, sk*) *←* KeyGen(*H*1(*pw*)), decrypts *K*<sup>1</sup> *←* Decap( $sk, c$ ) and computes  $\sigma := H_3(K_0, K_1, sid|\psi)$ . Finally, Sim issues

(CorruptKey,  $C^{(i)}$ , *iid*, *sid*| $\psi | \sigma$ ,  $H_4(K_0)$ ) query to  $\mathcal{F}_{apake}$  and  $\mathcal{F}_{apake}$  returns  $(\textit{sid}|\psi|\sigma,\textsf{sKey}) = H_4(K_0))$  to  $(\mathsf{C}^{(i)},\textit{iid})$ . In this case  $\mathsf{sKey} = H_4(K_0)$  and  $\sigma = H_3(K_0, K_1, sid|\psi)$  in both  $G_4$  and  $G_5$ .

 $-$  **Case II occurs to**  $(C^{(i)},iid)$ : *A* must have stolen server's password file and then impersonates party  $S^{(j)}$ . In this case, Sim first decrypts  $c \leftarrow$  AE.Dec( $H_2(K_0), \psi$ ).

– If  $c = \bot$ , then Sim issues (Abort, C<sup>(*i*)</sup>, *iid*) query to  $\mathcal{F}_{apake}$ . As a result,  $\mathcal{F}_{apake}$  returns  $(L, L)$  to  $(\mathsf{C}^{(i)}, iid)$  to reject the session. In this case the session is rejected in both  $G_4$  and  $G_5$ .

 $-$  If  $c \neq \bot$ , then Sim retrieves  $sk$  from its trapdoor record  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, rw,pk,sk,$ *r*) to decrypt  $K_1 \leftarrow \text{Decap}(sk, c)$  and computes  $\sigma := H_3(K_0, K_1, sid|\psi)$ . Then Sim issues (Impersonate,  $C^{(i)},$   $iid)$  query to  $\mathcal{F}_{apake},$  which indicates that Sim impersonates  $S^{(j)}$  to attack  $(C^{(i)},iid)$ . Finally, Sim issues (CorruptKey,  $C^{(i)}$ , *iid, sid* $|\psi|\sigma, H_4(K_0)$  query to  $\mathcal{F}_{apake}$  and then  $\mathcal{F}_{apake}$  must return  $(sid|\psi|\sigma,$  $\mathsf{skey} = H_4(K_0))$  to  $(\mathsf{C}^{(i)},iid)$ . In this case the session key is  $\mathsf{skey} = H_4(K_0)$ and  $\sigma = H_3(K_0, K_1, sid|\psi)$  in both  $G_4$  and  $G_5$ .

**– Case III: neither Case I nor Case II occurs to** (C (*i*) *, iid*)**.** In this case,  $K_0$ ,  $H_2(K_0)$ ,  $H_3(K_0, K_1, c)$  and  $H_4(K_0)$  are all simulated with uniform ones by Sim. We further consider the following two subcases according to passive or active attacks.

– **Passive Attack with**  $\psi$ **:** In this case,  $(C^{(i)},iid)$  and  $(S^{(j)}, iid')$  must have shared the same PAKE session key  $K_0$  and  $\psi$  must be generated by Sim for some instance  $(S^{(j)}, \text{iid}')$ . Sim randomly chosen  $\sigma \leftarrow \{0, 1\}^{\lambda}$  and issues (FreshKey,  $C^{(i)}$ , *iid*, *sid* $|\psi|\sigma$ ) query to  $\mathcal{F}_{apake}$ . Then  $\mathcal{F}_{apake}$  returns a uniform session key  $s$ Key  $\leftarrow$  s  $\{0,1\}^{\lambda}$  to  $(\textsf{C}^{(i)},iid)$ . Later if the server session  $(\textsf{S}^{(j)},iid')$ receives the same *σ* later, Sim will issue (CopyKey, S<sup>(*j*)</sup>, *iid', sid*| $ψ|σ)$  query to  $\mathcal{F}_{apake}$  directly. In this case,  $(\mathsf{C}^{(i)},iid)$  and  $(\mathsf{S}^{(j)},iid')$  share a same uniform session key chosen by *Fapake*.

– **Active Attack with**  $\psi$ : In this case  $\psi$  is not generated by Sim or instance  $(S^{(j)}, \text{iid}')$  and  $(C^{(i)}, \text{iid})$  do not share a same PAKE session key. Sim issues (Abort,  $C^{(i)}$ , *iid*) query to  $\mathcal{F}_{apake}$ . As a result,  $\mathcal{F}_{apake}$  returns  $(\perp, \perp)$  to  $(C^{(i)}, iid)$  to reject the session.

In both sub-cases, Sim does not need to retrieve *pk* to generate *c* via  $(c, K_1) \leftarrow$  Encap $(pk)$  for the server instance  $(S^{(j)}, iid')$ , and it does not need to retrieve *sk* to decrypt *c* for client instance  $(C^{(i)}, iid)$  any more.

If Case I or Case II occurs,  $G_4$  and  $G_5$  are the same, as analysed above. The difference lies in Case III.

Recall in  $G_4$ , if neither Case I nor Case II occurs, then  $K_0$  is uniform and independent of  $\mathcal{A}$ 's view. Consequently,  $H_3(K_0, K_1, sid|\psi)$ ,  $H_4(K_0), H_2(K_0)$  are all uniform to  $A$  and independent of  $A$ 's view. In the case of passive attack where  $\psi$  is generated by Sim, the client message  $\sigma := H_3(K_0, K_1, sid|\psi)$  and the client session key sKey :=  $H_4(K_0)$  are uniform to A. In  $\mathsf{G}_5$ ,  $\sigma$  and sKey are uniformly chosen. Therefore,  $\sigma$  and skey have the same distribution in  $G_4$  and  $G_5$ . In the case of active attack with  $\psi$ , Sim directly rejects  $\psi$  in  $G_5$ . But in  $G_4$ , Sim accepts if  $\psi$  is valid and rejects otherwise.  $G_5$  and  $G_4$  are the same unless the event **BadAuth** happens, where **BadAuth** is defined as

**Bad**<sub>Auth</sub>: Active attack with  $\psi$  results in AE.Dec( $H_2(K_0), \psi$ )  $\neq \bot$  in Case III.

With difference lemma, we know that  $|Pr(G_5 \Rightarrow 1] - Pr(G_4 \Rightarrow 1]| \leq Pr[\text{Bad}_{\text{Auth}}].$ 

Recall that  $H_2(K_0)$  is uniformly distributed in Case III. Thanks to the authenticity of AE, *A* only has negligible probability of generating a valid ciphertext  $\psi$  without the knowledge of  $H_2(K_0)$ .

Hence,  $\psi$  will always be rejected by  $(\mathsf{C}^{(i)},iid)$  except with a negligible probability. Given at most  $\ell$  sessions, we know that  $Pr[\mathbf{Bad_{Auth}}] \leq \ell \cdot \mathbf{Adv}_{AE}^{\mathsf{ot-auth}}(\mathcal{B}_{AE})$ . Consequently, we have

$$
\left|\Pr\left[\mathsf G_5 \Rightarrow 1\right] - \Pr\left[\mathsf G_4 \Rightarrow 1\right]\right| \leq \ell \cdot \mathsf{Adv}^{\mathsf{ot}\text{-}\mathsf{auth}}_{\mathsf{AE}}(\mathcal B_{\mathsf{AE}}).
$$

Note that  $A$  may additionally see a valid ciphertext  $\psi$  generated by Sim. This is why we need a one-time secure authenticity AE scheme.

Now in G5, Sim only needs *rw* to distinguish Case II, needs *pk* to simulate the generation of  $\psi$  for server instances in Case II, and needs *sk* to decrypt *c* so as to compute  $\sigma$  for client instances in Case II. But it still needs *pw* to generate (*rw, pk, sk*).

**Game**  $G_6$  (simulation for password file without  $pw$ ). In  $G_6$ , Sim behaves the same as  $G_5$  except for the simulation of generating the password file  $(rw, pk)$ .

Recall that in G5, Sim uses password *pw* to generate the password file (*rw, pk*) upon *A*'s StorePWFile query, but reveals the file to *A* upon *A*'s StealPWFile query. Sim also generates the trapdoor record along with the file but only uses the record for Case II in which StealPWFile must have happened. This fact suggests that Sim can delay the generation of password file (*rw, pk*) and the trapdoor  $\mathrm{record}\ (\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, rw,pk,sk,H_1(pw))$  until <code>StealPWFile</code> query. This is exactly <code>Sim</code> does without  $pw$  in  $\mathsf{G}_6$ , but with the help of re-programming technique of ROs.

In  $G_6$ , Sim will keep the password file and the trapdoor record empty until StealPWFile query from *A*. We consider three phases of the game.

**– Before receiving** (StealPWFile,  $C^{(i)}, S^{(j)}$ ) from A: In this phase, the simulations by  $Sim$  is exactly the same as that in  $G<sub>5</sub>$ . Besides,  $Sim$  also does the following. For any random oracle query to  $H_0(x)$  or  $H_1(x)$  from A, Sim will also issue a query (OfflineTestPW,  $C^{(i)}, S^{(j)}, x$ ) to  $\mathcal{F}_{apake}$  and  $\mathcal{F}_{apake}$  will store an offline-guess record.

Recall that only Case I or Case III happens in this phase. But neither *pw* nor the trapdoor record is needed by  $Sim$  for Case I and Case III in  $G_5$ . So the simulation without  $pw$  in  $G_6$  is sound in this phase.

 $-$  **When receiving** (StealPWFile,  $C^{(i)}$ ,  $S^{(j)}$ ) from  $\mathcal{A}$ : It must happen that  $\mathcal{A}$ has issued a StealPWFile query to a dummy server S (*j*) and then Sim issues a StealPWFile query to *Fapake*. According to the specification of *Fapake* in Fig. 4,  $\mathcal{F}_{apake}$  sends (StealPWFile,  $C^{(i)}, S^{(j)})$  to Sim.

Upon the output (StealPWFile,  $C^{(i)}, S^{(j)}$ ) from  $\mathcal{F}_{apake}$ ,

(1) If Sim ever issued a (OfflineTestPW,  $C^{(i)}$ ,  $S^{(j)}$ , x) query before such that  $x = pw$ ,  $\mathcal{F}_{apake}$  must have additionally output *pw* and "correct guess" to Sim. In this case, Sim obtains the correct password *pw*, and then it can invoke  $(pk, sk) \leftarrow \text{KeyGen}(H_1(pw))$ , set  $rw := H_0(pw)$ , return  $(rw, pk)$  to A, and set the trapdoor record as  $(C^{(i)}, S^{(j)}, rw,pk, sk, H_1(pw))$ . The simulations of password file and trapdoor record are exactly like G5.

(2) Otherwise, Sim randomly samples  $rw \leftarrow s \{0,1\}^{\lambda}$  and  $r \leftarrow s \mathcal{R}$ , invokes  $(pk, sk) \leftarrow \text{KeyGen}(r)$ , returns  $(rw, pk)$  to A, and stores record  $(\text{C}^{(i)}, \text{S}^{(j)},$  $rw, pk, sk, r$ ). In this case, *A* did not ever query  $H_0(pw)$  or  $H_1(pw)$ , so these hash values are uniform to *A*. Though Sim does not know the value of *pw*, it implicitly set  $H_0(pw) := rw$  and  $H_1(pw) = r$ . Therefore, in this case, the simulations of password file and trapdoor record are exactly the same from  $\mathcal{A}$ 's view no matter in in  $\mathsf{G}_6$  or  $\mathsf{G}_5$ .

 $-$  **After receiving** (StealPWFile,  $C^{(i)}, S^{(j)}$ ) from  $A$ : In this phase, Sim will use the trapdoor record for the simulations, exactly like in  $G_5$ . Besides, Sim also keeps an eye on random oracle queries: For each new random oracle query for  $H_0(x)$  or  $H_1(x)$ , Sim will issue a query (OfflineTestPW,  $\mathsf{C}^{(i)}, \mathsf{S}^{(j)},$ *x*) to  $\mathcal{F}_{apake}$  and check the reply. If  $\mathcal{F}_{apake}$  returns "correct guess", Sim will retrieve the record  $(C^{(i)}, S^{(j)}, rw,pk,sk,r)$  and *reprogram*  $H_0(x = pw) := rw$ and  $H_1(x = pw) := r$  by storing  $(x, rw)$  in list  $\mathcal{L}_{H_0}$  and  $(x, r)$  in list  $\mathcal{L}_{H_1}$ . As long as *A* issues oracle queries on *pw*, Sim will detect it and obtains the correct password *pw*. In this way, Sim keeps the consistence between *pw* and the trapdoor record. Even if *A*'s offline-attack succeeds, Sim still give a perfect simulation for  $\mathcal{A}$ , just like  $G_5$ .

Due to the simulation strategy and reprogramming technique, the distribution of trapdoor record  $(C^{(i)}, S^{(j)}, rw,pk,sk,r = H_1(pw))$  in  $G_6$  and  $G_5$  are exactly the same. Therefore, we have

$$
\Pr\left[\mathsf{G}_6 \Rightarrow 1\right] = \Pr\left[\mathsf{G}_5 \Rightarrow 1\right].
$$

**Game** G<sup>7</sup> **(Simulation of dealing with** *σ* **for server instances without**  $pw$ ). In  $G_7$ , Sim is the same as in  $G_6$ , except for the simulation of the server instance  $(S^{(j)}, \text{iid}')$  when receiving  $\sigma$ . We still consider Sim's simulations in the three cases in  $G_7$ .

First, Sim retrieves record  $(S^{(j)}, \text{iid}', K_1)$  (that was stored when  $\psi$  is simulated for  $(\mathsf{S}^{(j)},iid')$  and the corresponding PAKE key  $K_0$ .

**Case I occurs to**  $(S^{(j)},iid')$ . If  $\sigma \neq H_3(K_0, K_1, sid|\psi)$ , then Sim sends (Abort,  $(\mathsf{S}^{(j)},iid')$  to  $\mathcal{F}_{apake}$ . As a result,  $\mathcal{F}_{apake}$  returns  $(\perp, \perp)$  to  $(\mathsf{S}^{(j)},iid')$  to reject the session. If  $\sigma = H_3(K_0,K_1,sid|\psi),$  then Sim sends (CorruptKey, S<sup>(j)</sup>, iid',  $si d|\psi|\sigma, H_4(K_0)$  to  $\mathcal{F}_{apake}$ . Then  $\mathcal{F}_{apake}$  returns  $H_4(K_0)$  to  $(\mathsf{S}^{(j)},iid')$ . Recall in  $G_6$ , sKey =  $\perp$  if  $\sigma$  is invalid and sKey =  $H_4(K_0)$  if  $\sigma$  is valid. Therefore,  $\mathsf{G}_6$  and  $\mathsf{G}_7$  are the same in this case.

**Case II occurs to**  $(S^{(j)},iid')$ . If  $\sigma \neq H_3(K_0, K_1, sid|\psi)$  or  $\sigma = H_3(K_0, K_1, sid|\psi)$ but *Fapake* did not return "correct guess" to any of Sim's (OfflineTestPW*,*

 $C^{(i)}$ ,  $S^{(j)}$ ,  $pw'$ ) queries, then Sim sends (Abort,  $S^{(j)}$ , *iid'*) to  $\mathcal{F}_{apake}$ . As a result,  $\mathcal{F}_{apake}$  returns  $(\perp, \perp)$  to  $(\mathsf{S}^{(j)},iid')$  to reject the session.

If  $\sigma = H_3(K_0, K_1, sid|\psi)$  and  $\mathcal{F}_{apake}$  returned "correct guess" to one of Sim's (OfflineTestPW,  $C^{(i)}$ ,  $S^{(j)}$ ,  $pw'$ ) queries, then Sim sends (CorruptKey,  $S^{(j)}$ , *iid'*,  $sid|\psi|\sigma, H_4(K_0)$ ) to  $\mathcal{F}_{apake}$ . Accordingly  $\mathcal{F}_{apake}$  returns  $(sid|\psi|\sigma, sKey :=$  $H_4(K_0)$ ) to  $(S^{(j)}, \text{iid}')$ .

Recall in  $G_6$ , as long as Case II occurs, the simulator will set the session key sKey :=  $H_4(K_0)$  if  $\sigma = H_3(K_0, K_1, sid|\psi)$ , and set sKey :=  $\perp$  if  $\sigma \neq$  $H_3(K_0, K_1, sid|\psi)$ .

**Case III occurs to**  $(S^{(j)}, \text{iid}')$ . We further consider whether  $\sigma$  is from *A*' passive attack or active attack.

**Passive Attack with**  $\sigma$ **: In this case, there must exist some instance**  $(C^{(i)}, iid)$  which has agreed PAKE key  $K_0$  with  $(S^{(j)}, iid')$ , and  $\sigma$  must be generated by  $Sim$  for  $(C^{(i)},iid)$ . Moreover,  $Sim$  must also have issued (FreshKey,  $C^{(i)}$ , *iid*, *sid*| $\psi|\sigma$ ) query to  $\mathcal{F}_{apake}$  and  $\mathcal{F}_{apake}$  returns a uniform session key sKey to  $(\textsf{C}^{(i)},iid).$  Now Sim issues  $(\textsf{CopyKey}, \, \textsf{S}^{(j)}, iid', sid|\psi|\sigma)$ query to  $\mathcal{F}_{apake}$  directly. In this case,  $(\mathsf{C}^{(i)},iid)$  and  $(\mathsf{S}^{(j)},iid')$  share a same uniform session key sKey chosen by  $\mathcal{F}_{apake}$  and the session key sKey is simulated with the help of  $\mathcal{F}_{apake}$ , without the knowledge of *pw* or the trapdoor record, just like  $G_6$ .

– **Active Attack with** *ψ***:** Sim sends (Abort*,* S (*j*) *, iid′* ) to *Fapake*, and accordingly  $\mathcal{F}_{apake}$  returns ( $\bot$ , sKey  $:= \bot$ ) to ( $\mathsf{S}^{(j)},iid'$ ) to reject the session. Recall that in Case III of  $\mathsf{G}_6$ ,  $H_4(K_0, K_1, sid|\psi)$  is uniformly distributed and independent of  $A$ 's view, the simulator will reject the session by setting sKey := *⊥* except with negligible probability.

Therefore,  $G_6$  and  $G_7$  are the same in Case III except with negligible probability.

According to the above analyses,  $G_7$  and  $G_6$  are the same except a bad event **Bad** happens, where

**Bad:** Case II occurs to some  $(\mathsf{S}^{(j)}, \textit{iid}')$ ,  $\sigma = H_3(K_0, K_1, \textit{sid}|\psi)$ , but until then none of the Sim's (OfflineTestPW,  $C^{(i)}, S^{(j)}, pw'$ ) queries results in "correct guess".

With difference lemma, we know that  $|Pr(G_7 \Rightarrow 1] - Pr(G_6 \Rightarrow 1)| \leq Pr[\text{Bad}].$ Next we show a reduction algorithm  $\mathcal{B}_{\mathsf{KEM}}$  and prove Pr  $[\mathbf{Bad}] \leq N^2 \ell \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow-pca}}(\mathcal{B}_{\mathsf{KEM}})$ .

 $\mathcal{B}_{\mathsf{KEM}}$  obtains a public key  $pk^*$ , a key encapsulation  $c^*$ , and has access to oracle  $\mathsf{Check}(\cdot, \cdot)$  which on input  $(c, K)$  returns 1 iff  $\mathsf{Decap}(sk^*, c) = K$ .  $\mathcal{B}_{\mathsf{KEM}}$ aims to find  $K^*$  s.t.  $\mathsf{Decap}(sk^*, c^*) = K^*$ .

In the reduction,  $\mathcal{B}_{\mathsf{KEM}}$  plays the role of the simulator Sim in  $G_7$ . It first  $\text{randomly choose } (C^{(*)}, S^{(*)}, \text{iid}') \leftarrow_{\text{s}} [N] \times [N] \times [\ell].$   $\mathcal{B}_{\text{KEM}} \text{ sets } (S^{(*)}, \text{iid}') \text{ as the }$ target instance. Suppose that  $(C^{(*)},iid)$  is the partnered instance.  $\mathcal{B}_{KEM}$  will detect whether **Bad** happen on (S (*∗*) *, iid′* ).

- For any other instances,  $B_{KEM}$  behaves just like Sim does in  $G_7$ .
- $-$  For the instances ( $C^{(*)}$ , *iid*) and ( $S^{(*)}$ , *iid'*), *B*<sub>KEM</sub> does the simulations as follows.
	- Before receiving (StealPWFile,  $C^{(*)}$ ,  $S^{(*)}$ ) from *A*: *B*<sub>KEM</sub> behaves just like Sim does in G7.
	- *•* **Upon receiving** (StealPWFile,  $C^{(*)}$ ,  $S^{(*)}$ ) from *A*: *B*<sub>KEM</sub> issues a StealPWFile query to  $\mathcal{F}_{apake}$ . If  $\mathcal{F}_{apake}$  replies with *pw* and "correct guess", then **Bad** does not happen on  $(S^{(*)}, iid')$ , and  $B_{KEM}$  aborts the game. Otherwise,  $\mathcal{B}_{\mathsf{KEM}}$  randomly samples  $rw \leftarrow \{0, 1\}^{\lambda}$ , sets the trapdoor record as  $(\mathsf{C}^{(*)}, \mathsf{S}^{(*)}, rw, pk^*, sk =?, r =?),$  and returns the password file  $(rw, pk^*)$  to A.
	- **After receiving** (StealPWFile,  $C^{(*)}$ ,  $S^{(*)}$ ) from *A*: During this phase, for any offline attack from  $A$ , if the corresponding (OfflineTestPW,  $C^{(*)}, S^{(*)},$  $pw'$ ) query to  $\mathcal{F}_{apake}$  results in "correct guess", then **Bad** does not happen on  $(S^{(*)}, \text{iid}')$  and  $\mathcal{B}_{\mathsf{KEM}}$  aborts the game.

For the session between instances  $(C^{(*)},iid)$  and  $(S^{(*)}, iid')$ ,  $B_{KEM}$  first simulates the generation of PAKE key  $K_0$  without  $pw$  and trapdoor record, just like Sim. If Case II does not happen, then **Bad** does not happen on  $(S^{(*)}, \text{iid}')$  and  $\mathcal{B}_{\mathsf{KEM}}$  aborts the game.

Next, for server instance  $(S^{(*)}, \text{iid}'), \mathcal{B}_{\mathsf{KEM}}$  can simulate the generation of  $\psi$ ,  $\mathcal{B}_{\mathsf{KEM}}$  invokes  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), c^*)$ . Since  $c^*$  is an encapsulation under  $pk^*$ , we know  $\mathcal{B}_{\mathsf{KEM}}$ 's simulation is perfect, just like Sim.

For client instance  $(C^{(*)}, id)$  to deal with  $\psi'$ , recall that Sim may decrypt  $\psi'$  with  $sk^*$  to generate  $K'_1$  and then  $\sigma$ , but  $\mathcal{B}_{\mathsf{KEM}}$  has no  $sk^*$ at all. To deal with this problem,  $\mathcal{B}_{\mathsf{KEM}}$  resorts to ROs and the reprogramming techniques: upon  $(C^{(*)}, iid)$  receiving  $\psi'$ ,  $\mathcal{B}_{\mathsf{KEM}}$  invokes  $c \leftarrow$  AE.Dec( $H_2(K_0), \psi'$ ). If  $c = \bot$ ,  $\mathcal{B}_{\mathsf{KEM}}$  behaves just like Sim (no  $sk^*$ involved). If  $c \neq \bot$ , for all hash queries  $H_3(K_0, K'_1, sid|\psi')$ ,  $\mathcal{B}_{\mathsf{KEM}}$  first checks whether there exists  $K'_{1}$  such that  $\mathsf{Check}(c, K'_{1}) = 1$ . If yes, set  $\sigma := H_3(K_0, K'_1, sid|\psi')$ . Otherwise randomly choose  $\sigma \leftarrow s \{0,1\}^{\lambda}$ , and set  $H_3(K_0, K'_1 = ?$ ,  $sid|\psi') := \sigma$  with ? denoting the undermined value of *K*<sup> $\prime$ </sup><sub>1</sub>. It then returns  $\sigma$  to  $\mathcal A$  and returns **sKey** :=  $H_2(K_0)$  as the session key. If later *A* issues any new query  $H_3(K_0, K'_1, sid|\psi'), \mathcal{B}_{\text{KEM}}$  checks whether Check $(c, K'_1) = 1$ . If yes,  $\mathcal{B}_{\mathsf{KEM}}$  re-programme  $H_3(K_0, K'_1, sid|\psi') := \sigma$ . Upon  $(S^{(*)}, \text{iid}')$  receiving  $\sigma'$ . If there is no  $H_3$  query such that  $\sigma' =$  $H_3(K_0, K'_1, sid|\psi)$ , then **Bad** does not happen on  $(\mathsf{S}^{(*)}, iid')$  since  $\sigma'$  is hardly valid due to the uniformity of RO, and then  $B_{KEM}$  aborts the game. Otherwise, find *A*'s hash query such that  $\sigma' = H_3(K_0, K'_1, sid|\psi)$ ,  $\mathcal{B}_{\mathsf{KEM}}$  checks whether  $\mathsf{Check}(c, K'_1) = 1.$  If no, then  $K'_1 \neq K^*_1 := \mathsf{Decap}(sk^*,$ *c*) so  $\sigma' \neq H_3(K_0, K_1^*, sid|\psi)$ . Thus **Bad** does not happen on  $(\mathsf{S}^{(*)}, iid')$ and  $\mathcal{B}_{\mathsf{KEM}}$  aborts the game. If yes,  $\mathsf{Check}(c, K'_1) = 1$  implies  $K'_1 = K^*_1 =$ Decap( $sk^*$ , *c*). Now **Bad** happens,  $\mathcal{B}_{\mathsf{KEM}}$  just replies this  $K'_1$  as it answer to its OW-PCA challenger.

The above description and analysis shows that  $B_{KEM}$  presents a perfect simulation like Sim. As long as  $\bf{Bad}$  happens on  $({\sf S}^{(*)},iid'), {\cal B}_{\sf KEM}$  wins in its OW-PCA

experiment. Consequently, we have

$$
\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow-pca}}(\mathcal{B}_{\mathsf{KEM}}) = \Pr\left[\mathbf{Bad} \text{ on } (\mathsf{S}^{(*)}, iid')\right] = \frac{1}{N^2\ell} \cdot \Pr\left[\mathbf{Bad}\right].
$$

So,

$$
|\Pr\left[\mathsf{G}_7 \Rightarrow 1\right] - \Pr\left[\mathsf{G}_6 \Rightarrow 1\right]| \leq \Pr\left[\mathbf{Bad}\right] = N^2 \ell \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow-pca}}(\mathcal{B}_{\mathsf{KEM}}).
$$

Finally, by combining all the statements across  $G_0-G_7$ , we get

$$
\begin{aligned} \left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}\right]-\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}\right]\right| \leq & N^2\ell\cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow-pca}}(\mathcal{B}_{\mathsf{KEM}}) + \frac{q^2+q+1}{2^{\lambda}} \\ & + \ell\cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot-cut}}(\mathcal{B}_{\mathsf{AE}}) + \ell\cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot-cca}}(\mathcal{B}_{\mathsf{AE}}). \end{aligned}
$$

*⊓⊔*

*Remark 2 (Tightly secure aPAKE compiler.).* Note that in the security proof, the security loss is due to the guessing strategy in the reduction between  $G_7$ and  $G_6$ , which relies on the OW-PCA security of the KEM scheme. By imposing stronger security on KEM, it is possible to make the reduction a tight one. For example, Pan et. al. [26] proposed the OW-ChCCA security for KEM and presented instantiation of such a KEM with tight OW-ChCCA secuity based on the (Matrix) DDH assumption. The OW-ChCCA security considers a multi-user setting, where besides the check oracle, the adversary adaptively corrupt users to obtain their secret keys, g[ets](#page-30-4) multiple challenge ciphertext encapsulations  $\{c_i\}$ , adaptively reveals some ciphertexts  ${c_i'} \subseteq {c_j}$  to obtain encapsulation keys, and adaptively obtains decryption results for  ${c'_{k}}$  such that  ${c_{j}} \cap {c'_{k}} = \emptyset$ , and it requires that it is still hard for such an adversary to guess correctly an encapsulation key for any unrevealed ciphertext under public keys of uncorrupted users.

**Theorem 2.** *Under the same condition as in Theorem 1, if* KEM *is replaced with an OW-ChCCA secure one and* AE *is an information-theoretical AE scheme with one-time authenticity and one-time CCA security, then we have*

<span id="page-26-0"></span>
$$
\left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}\right]-\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}\right]\right|\leq \mathsf{Adv}_{\mathsf{KEM}}^{(N^2,\ell)\text{-}\mathsf{ChCCA}}(\mathcal{B}_{\mathsf{KEM}})+\frac{2\ell+q^2+q+1}{2^{\lambda}}.
$$

The security proof of Theorem 2 is given in Appendix B. In Appendix B, we recall the formal definition of OW-ChCCA security for KEM. Then we prove that our compiler is tightness-preserving when equipped with such a OW-ChCCA secure KEM. In Fig.10, we review the specific tightly OW-ChCCA secure KEM based on the Matrix DDH assumpti[on](#page-26-0), which was propose[d](#page-34-0) in [26].

## **4 Instan[tia](#page-40-0)tions of aPAKE from Our Compiler and PAKE**

<span id="page-26-1"></span>Our aPAKE compiler needs an OW-PCA secure KEM scheme and an AE scheme with one-time authenticity and one-time CCA security. A good candidate for AE is the canonical information-theoretic construction.

**Instantiation of one-time secure AE.** For one-time secure AE, we present a concrete information-theoretically secure AE scheme AEit = (AE*.*Enc*,* AE*.*Dec) from one-time pad and one-time MAC, using the Encrypt-then-MAC approach. Here the key space is  $\{0,1\}^{3\lambda}$ , the message space is  $\{0,1\}^{\lambda}$ , and the ciphertext space is  $\{0,1\}^{2\lambda}$ . All operations are over field  $\mathbb{F}_{2^{\lambda}}$ .

- $-$  AE.Enc( $k, m$ ) : Parse  $k := (k_1, k_2, k_3) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$  and  $m \in \mathbb{F}_{2^\lambda}$ . Compute  $c_1 := k_1 + m$  and  $c_2 = k_2 \times c_1 + k_3$ .
- **−** AE.Dec( $k, c$ ) : Parse  $k := (k_1, k_2, k_3) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$  and  $c = (c_1, c_2)$ . If  $c_2 \neq k_2 \times c_1 + k_3$ : return ⊥. Otherwise return  $c_1 - k_1$ .

**Lemma 1.** *For the above AE, we have*

$$
\mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot}\text{-}\mathsf{auth}}(\mathcal{A}) \le 1/2^{\lambda}, \quad \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot}\text{-}\mathsf{cca}}(\mathcal{A}) \le 1/2^{\lambda}.
$$

<span id="page-27-0"></span>*for any (even all-powerful) adversaries.*

The security proof of Lemma 1 is shown in Appendix C.

The OW-PCA security is a security notion weaker the CCA security, which admits flexible choices for the underling KEM in our compiler. Next we show how to instantiate our compiler according to the PAKE scheme so as the resulting aPAKE scheme enjoying good fe[at](#page-27-0)ures.

#### **4.1 Most Efficient aPAKE from Lattice**

<span id="page-27-1"></span>Recall that there exists efficient UC-secure PAKE protocols [29,2,20] from lattice, where the PAKE protocol in  $[2,20]$  is based on Kyber  $[7]$  and that in  $[29]$  is based on Saber [10]. Kyber  $[7]$  is the NIST PQC winner for KEM, and its CCA security is based on the module-LWE assumption [21], so we choose the Kyber-based UCsecure PAKE scheme [2].

Now we also take Kyber [\(](#page-29-2)[who](#page-30-9)se CCA security n[at](#page-29-9)urally implie[s O](#page-31-1)W-PCA security) [as](#page-29-10) the KE[M](#page-29-9) in our compiler. Then together with the informationtheoretically secure  $AE_{it}$ , we obtain a Ky[ber-](#page-30-10)based aPAKE scheme (see Fig. 9 in App[e](#page-29-2)ndix  $E$  for a more detailed description of the scheme).

Note that the second to last round of our compiler can be merged in the last round of PAKE, resulting in a 3-round Kyber-based aPAKE scheme with UC-security.

**Compari[so](#page-38-2)n to other lattice-based aPAKEs.** Up to now, the only approach to lattice-based UC-secure aPAKE is via the signature-aided compiler proposed in [15]. However, signature schemes from lattice, like Dilithium or Falcon, are far less efficient than their KEM counterpart like Kyber. We compare our Kyberbased compiler to the Dilithium-based Compiler in [15] and the Falcon-based Compiler in [15] in terms of the computing and communication efficiency in Ta[ble](#page-30-0) 1.

The comparison in Table 1 suggests that our Kyber-based compiler is the most efficient one, and hence the resulting Kyber-base[d aP](#page-30-0)AKE scheme (=Kyberbased compil[er +](#page-30-0) Kyber-based PAKE) is the *most efficient* aPAKE scheme with UC-se[cu](#page-8-0)rity from lattice up to now.

**Corollary 1.** *Under the same condition as in Theorem 1, suppose that* KEM *is instantiated with the CCA secure KEM scheme Kyber [7] and* AE *is an informationtheoretical AE scheme with one-time authenticity and one-time CCA security in our compiler. If the underlying PAKE is instantiated with the UC-secure Kyberbased PAKE protocol in [2], then the resulting aPAKE h[as](#page-14-0) UC-security s.t.*

 $|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}\right]-\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}\right]|\leq (5q+2q\ell+2N^2\ell)\cdot\mathsf{Adv}_{k+1,k,\chi}^{\mathsf{mlive}}(\mathcal{A})+2^{-\varOmega(\lambda)},$ 

w[he](#page-29-2)re  $\text{Adv}_{k+1,k,\chi}^{\text{mlwe}}(\mathcal{A})$  *is the advantage function for the Module-LWE problem [21].* 

#### **4.2 Tightly Secure aPAKE Scheme from Matrix DDH**

For the underlying PAKE protocol, there exist tightly UC-secure PAKE pr[oto](#page-30-10)cols from the CDH assumption [22,27]. Accordingly, our compiler also has tightly secure OW-ChCCA secure KEM [26] from the (Matrix) DDH assumption as candidate.

<span id="page-28-0"></span>Now we take the MDDH-based KEM [26] with tight OW-ChCCA security as the KEM in our compiler. [The](#page-30-5)[n t](#page-30-11)ogether with the information-theoretically secure  $AE_{it}$ , our co[mpi](#page-30-4)ler can compile the CDH-based tightly UC-secure PAKE scheme to a MDDH-based aPAKE scheme, which serves as the *first tightly UCsecure* aPAKE scheme up to now (See F[ig.](#page-30-4) 9 in Appendix E for the tightly UC-secure aPAKE scheme).

**Corollary 2.** *Under the same condition as in Theorem 2, suppose that* KEM *is instantiated with the tightly OW-ChCCA secure MDDH-based KEM in [26] and* AE *is an information-theoretical AE sche[me](#page-39-0) with one-tim[e a](#page-38-2)uthenticity and one-time CCA security in our compiler. If the underlying PAKE is instantiated with the tightly UC-secure CDH-based PAKE protocol in [[22](#page-26-0)], then the resulting aPAKE has tight UC-security such that*

 $|Pr[\textbf{Real}_{\mathcal{Z},\mathcal{A}}] - Pr[\textbf{Ideal}_{\mathcal{Z},\textbf{Sim}}]| \leq 12 \cdot \textsf{Adv}_\textbf{MDDH}(\mathcal{A}) + 2 \cdot \textsf{Adv}_\textbf{CDH}(\mathcal{A}) + 2^{-\Omega(\lambda)},$ 

*where*  $\text{Adv}_{\text{MDDH}}(\mathcal{A})$  *and*  $\text{Adv}_{\text{CDH}}(\mathcal{A})$  *are the advantage fu[nct](#page-30-5)ion for the MDDH problem [13] and the CDH problem.*

**Acknowledgements.** We would like to thank the reviewers for their valuable comments. This work was partially supported by Guangdong Major Project of Basic an[d A](#page-29-11)pplied Basic Research (2019B030302008), National Natural Science Foundation of China under Grant 61925207 and Grant 62372292, and the National Key R&D Program of China under Grant 2022YFB2701500.

## **References**

1. Barbosa, M., Gellert, K., Hesse, J., Jarecki, S.: Bare PAKE: universally composable key exchange from just passwords. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part II. LNCS, vol. 14921, pp. 183–217. Springer (2024), https://doi.org/10. 1007/978-3-031-68379-4\_6

- 2. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In: Tibouchi, M., Wang, X. (eds.) ACNS 23, Part II. LNCS, vol. 13906, pp. 516–538. Springer, Heidelberg (Jun 2023). https://doi.org/10.1007/978-3-031-33491-7\_19
- <span id="page-29-2"></span>3. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 244–250. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168618
- <span id="page-29-0"></span>4. [Benhamouda, F., Pointcheval, D.: Verifier-based pa](https://doi.org/10.1007/978-3-031-33491-7_19)ssword-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833 (2013), https://eprint.iacr.org/2013/833
- <span id="page-29-1"></span>5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS<sup>+</sup> signature framew[ork. In: Cavallaro, L., Kinder, J., Wang, X.,](https://doi.org/10.1145/168588.168618) Katz, J. (eds.) ACM CCS 2019. pp. 2129–2146. ACM Press (Nov 2019). https: //doi.org/10.1145/3319535.3363229
- <span id="page-29-5"></span>6. Boneh, D., Dag[delen, Ö., Fischlin, M., Lehmann, A](https://eprint.iacr.org/2013/833)., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASI-ACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (Dec [2011\).](https://doi.org/10.1145/3319535.3363229) https://doi.org/10.1007/978-3-642-25385-0\_3
- <span id="page-29-7"></span>7. [Bos, J., Ducas, L., Kiltz, E., Lepoin](https://doi.org/10.1145/3319535.3363229)t, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-latticebased kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
- <span id="page-29-9"></span>8. [Canetti, R.: Universally composable security: A ne](https://doi.org/10.1007/978-3-642-25385-0_3)w paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
- 9. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (May 2005). https://doi. org/10.1007/11426639\_24
- <span id="page-29-8"></span>10. [D'Anvers, J.P., Karmakar, A., Roy, S.S.,](https://doi.org/10.1109/SFCS.2001.959888) Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 18. LNCS, vol. 10831, pp. 282–305. Springer, Heidelberg (May 2018). https://doi[.org/10.1007/](https://doi.org/10.1007/11426639_24) [978-3-319-89339-6\\_16](https://doi.org/10.1007/11426639_24)
- <span id="page-29-10"></span>11. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.1315[4/tches.v2018.i1.238-268](https://doi.org/10.1007/978-3-319-89339-6_16), https://tches.iacr.org/index.php/TCHES/article/view/839
- <span id="page-29-3"></span>12. [Duman, J., Hartmann,](https://doi.org/10.1007/978-3-319-89339-6_16) D., Kiltz, E., Kunzweiler, S., Lehmann, J., Riepel, D.: Generic models for group actions. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNC[S, vol. 13940, pp. 406–435. Springer, Heidelberg \(May](https://doi.org/10.13154/tches.v2018.i1.238-268) 2023). https://doi.org/10.1007/978-3-031-31368-4\_15
- <span id="page-29-6"></span>13. [Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An alge](https://tches.iacr.org/index.php/TCHES/article/view/839)braic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (Aug 2013). https: //doi.org/10.1007/978-3-642-40084-1\_8
- <span id="page-29-11"></span><span id="page-29-4"></span>14. Fouqu[e, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky,](https://doi.org/10.1007/978-3-031-31368-4_15) V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier

lattice-based compact signatures over ntru. Submission to the NISTs post-quantum cryptography standardization process **36**(5), 1–75 (2018)

- 15. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (Aug 2006). https://doi. org/10.1007/11818175\_9
- <span id="page-30-0"></span>16. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi[.org/10.1145/](https://doi.org/10.1007/11818175_9) 1374376.1374407
- 17. [Gu, Y., Jarecki, S., Kraw](https://doi.org/10.1007/11818175_9)czyk, H.: KHAPE: Asymmetric PAKE from key-hiding key exchange. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 701–730. Springer, Heidelberg, Virtua[l Event \(Aug 2021\).](https://doi.org/10.1145/1374376.1374407) https: [//doi.org/10.10](https://doi.org/10.1145/1374376.1374407)07/978-3-030-84259-8\_24
- <span id="page-30-2"></span>18. Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 579–599. Springer, Heidelberg (Sep 2020). https://doi.org/1[0.1007/](https://doi.org/10.1007/978-3-030-84259-8_24) 978-3-030-57990-6\_29
- <span id="page-30-8"></span>19. [Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Sh](https://doi.org/10.1007/978-3-030-84259-8_24)in, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 1[1035, pp. 485–504. Springer,](https://doi.org/10.1007/978-3-030-57990-6_29) [Heidelberg \(Sep 2018\).](https://doi.org/10.1007/978-3-030-57990-6_29) https://doi.org/10.1007/978-3-319-98113-0\_26
- <span id="page-30-7"></span>20. Januzelli, J., Roy, L., Xu, J.: Under what conditions is encrypted key exchange actually secure? Cryptology ePrint Archive, Paper 2024/324 (2024), https:// eprint.iacr.org/2024/324
- 21. Langlois, A., Stehlé, D.[: Worst-case to average-case reductions for module la](https://doi.org/10.1007/978-3-319-98113-0_26)ttices. DCC **75**(3), 565–599 (2015). https://doi.org/10.1007/s10623-014-9938-4
- <span id="page-30-9"></span>22. Liu, X., Liu, S., Han, S., Gu, D.: EKE meets tight security in the U[niversally](https://eprint.iacr.org/2024/324) [Composable framework. In: B](https://eprint.iacr.org/2024/324)oldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 685–713. Springer, Heidelberg (May 2023). https://doi. org/10.1007/978-3-031-31368-4\_24
- <span id="page-30-10"></span><span id="page-30-5"></span>23. Lyubashevsky, V.: Lattice si[gnatures without trapdoors. In: Pointcheval, D.,](https://doi.org/10.1007/s10623-014-9938-4) Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4\_43
- <span id="page-30-3"></span>24. [McQuoid, I., Xu, J.: An efficient stro](https://doi.org/10.1007/978-3-031-31368-4_24)ng asymmetric pake compil[er instantiable](https://doi.org/10.1007/978-3-031-31368-4_24) from group actions. In: ASIACRYPT 2023. pp. 176–207. Springer (2023), https: //eprint.iacr.org/2023/1434
- <span id="page-30-1"></span>25. Okamoto, T., Pointch[eval, D.: REACT: Rapid Enhanced-security Asym](https://doi.org/10.1007/978-3-642-29011-4_43)metric Cryptosystem Transform. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 159–175. Springer, Heidelberg (Apr 2001). https://doi.[org/10.](https://eprint.iacr.org/2023/1434) 1007/3-540-45353-9\_13
- <span id="page-30-6"></span>26. [Pan, J., Wagner, B., Zeng, R.: La](https://eprint.iacr.org/2023/1434)ttice-based authenticated key exchange with tight security. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 616–647. Springer, Heidelberg (Aug 2023). [https://doi.org/10.](https://doi.org/10.1007/3-540-45353-9_13) [1007/978-3-031-38554-](https://doi.org/10.1007/3-540-45353-9_13)4\_20
- <span id="page-30-11"></span><span id="page-30-4"></span>27. Pan, J., Zeng, R.: A generic construction of tightly secure password-based authenticated key exchange. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VIII. LNCS, vol. 14445, pp. 143–175. Springer, H[eidelberg \(Dec 2023\).](https://doi.org/10.1007/978-3-031-38554-4_20) [https://doi.org/10.1007/97](https://doi.org/10.1007/978-3-031-38554-4_20)8-981-99-8742-9\_5
- 28. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
- 29. Santos, B.F.D., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 128–156. Springer, Heidelberg (Apr 2023). https: //doi.org/10.1007/978-3-031-30589-4\_5
- <span id="page-31-1"></span>30. Santos, B.F.[D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymm](https://doi.org/10.1145/1060590.1060603)etric PAKE with low computation and communication. In: Dunkelman, O., Dziembowski, S. (eds.) EU-ROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 127–156. Springer, Hei[delberg](https://doi.org/10.1007/978-3-031-30589-4_5) (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3\_5
- <span id="page-31-2"></span>31. [Schmidt, J.: Requirements for password-a](https://doi.org/10.1007/978-3-031-30589-4_5)uthenticated key agreement (pake) schemes. Tech. rep. (2017), https://tools.ietf.org/html/rfc8125
- <span id="page-31-3"></span><span id="page-31-0"></span>32. Shoup, V.: Security analysis of itSPAKE2+. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III[. LNCS, vol. 12552, pp. 31–60. Springer, Heidelberg \(](https://doi.org/10.1007/978-3-031-07085-3_5)Nov 2020). https://doi.org/10.1007/978-3-030-64381-2\_2

## **Appendix**

## **A Full Descriptions of Simulator Sim**

<span id="page-32-1"></span>The simulator  $\mathsf{Sim}$  in  $\mathsf{G}_7$  is presented in Fig. 6 and Fig. 7.

<span id="page-32-0"></span>

Fig. 6: Simulator Sim part I.

```
Additional aPAKE operations
Generating \psi for server instance (S^{(j)}, iid'):
    Retrieve record (S
(j)
, iid′
, sid, Key).
    If there exists a compromised record (\mathsf{S}^{(j)}, \textit{id}', \mathsf{C}^{(i)}, (\textit{rw}, \textit{r})), then compute (\textit{pk}, \textit{sk}) \leftarrow \mathsf{KeyGen}(\textit{r}),
   (c, K_1) \leftarrow \text{Encap}(pk), and \psi \leftarrow \text{AE}.\text{Enc}(H_2(\text{Key}), c). Otherwise, compute \psi \leftarrow \text{AE}.\text{Enc}(H_2(\text{Key}), 0) and set K_1 := \bot.
    \text{Return } \psi \text{ from } \mathsf{S}^{(j)} \text{ to } \mathcal{A} \text{, sent} := \textsf{sent} \cup \{ \mathsf{S}^{(j)}, iid', sid, \psi, K_1 \}.on \psi from \mathcal A as a server message from S^{(j)} to (C^{(i)}, iid):
    \mathsf{recv} := \mathsf{recv} \cup \{ \mathsf{C}^{(i)}, iid, sid, \psi\}Retrieve record (C^{(i)},iid, sid, Key).
    If there are no compromised records (C^{(i)},iid, S^{(j)}, (rw, r)):
        If there exists record (\mathsf{S}^{(j)}, \text{iid}', \text{sid}, \psi, K_1) \in \mathsf{sent}:Generate \sigma \leftarrow \{0,1\}^{\lambda}, send (FreshKey, C^{(i)}, iid, sid|\psi|\sigma) to \mathcal{F}_{apake}, and return \sigma from C^{(i)} to A.
        If there are no records (\mathsf{S}^{(j)}, \text{iid}', \text{sid}, \psi) \in \mathsf{sent}:Reject the session by sending (Abort, C^{(i)}, iid) to \mathcal{F}_{apake}.
    Else there exists compromised records (C^{(i)}, \text{iid}, S^{(j)}, (rw, r)):
       Compute c \leftarrow \mathsf{AE}.\mathsf{Dec}(H_2(\mathsf{Key}), \psi).If c = \bot: reject the session by sending (Abort, C^{(i)}, iid) to \mathcal{F}_{apake}.
       Else:
           If there exists a stolen record (C^{(i)}, S^{(j)}, \cdot, rw, r), send (Impersonate, C^{(i)}, iid) to \mathcal{F}_{apake}.
           Generate (pk, sk) \leftarrow \mathsf{KeyGen}(r), decrypt K_1 \leftarrow \mathsf{Decap}(sk, c), compute \sigma := H_3(\mathsf{Key}, K_1, sid|\psi),
           send (CorruptKey, C<sup>(i)</sup>, iid, sid|\psi|\sigma, H_4(Key)) to \mathcal{F}_{apake}, return \sigma from C<sup>(i)</sup> to A.
on \sigma from \mathcal A as a client message from C^{(i)} to (S^{(j)}, \text{iid}'):
    Retrieve record (S
(j)
, iid′
, sid, Key).
    \text{If } \exists (\mathsf{S}^{(j)}, \mathit{iid}', \mathit{sid}, \psi, K_1) \in \mathsf{sent} \land (\mathsf{C}^{(i)}, \mathit{iid}, \mathit{sid}, \psi) \in \mathsf{recv} \land (\mathsf{C}^{(i)}, \mathit{iid}, \mathit{sid}, \sigma) \in \mathsf{sent}:Send (FreshKey, S^{(j)}, iid', sid|\psi|\sigma) to \mathcal{F}_{apake}.
    Retrieve K_1 from record (\mathsf{S}^{(j)}, \text{iid}', \text{sid}, \psi, K_1)If Sim queried (Testpw, S^{(j)}, iid', pw) to \mathcal{F}_{apake} before and obtained "correct guess":
        If \sigma = H_3(\text{Key}, K_1, sid|\psi): send (CorruptKey, S^{(j)}, iid', sid|\psi|\sigma, H_4(\text{Key})) to \mathcal{F}_{apake}.
    If Sim queried (OfflineTestPW, C^{(i)}, S^{(j)}, pw) before and obtained "correct guess":
        If \sigma = H_3(\text{Key}, K_1, sid|\psi): send (CorruptKey, S^{(j)}, iid', sid|\psi|\sigma, H_4(\text{Key})) to \mathcal{F}_{apake}.
    In any other cases: send (About, S<sup>(j)</sup>, <i>iid'</i>) to \mathcal{F}_{apake}.
Random Oracles
on H_0(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}) from A:
    If ∃(pw, C^{(i)}, S^{(j)}, Y) ∈ L_{H_0}: return Y
    Else send (OfflineTestPW, C^{(i)}, S^{(j)}, pw) to \mathcal{F}_{apake} and if it returns "correct guess":
        Update the stolen record with (C^{(i)}, S^{(j)}, pw, rw, r), \mathcal{L}_{H_0} := \mathcal{L}_{H_0} \cup \{ (pw, rw) \}, \mathcal{L}_{H_1} := \mathcal{L}_{H_1} \cup \{ (pw, r) \}, return rw.
    In any other cases: Y \leftarrow \{0,1\}^{\lambda}, \mathcal{L}_{H_0} := \mathcal{L}_{H_0} \cup \{(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, Y)\}, and return Y.
on H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}) from A:
    If ∃(pw, \, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, Y) \in \mathcal{L}_{H_1}: return Y
    Else send (OfflineTestPW, C^{(i)}, S^{(j)}, pw) to \mathcal{F}_{apake} and if it returns "correct guess":
        Update the stolen record with (C^{(i)}, S^{(j)}, pw, rw, r), \mathcal{L}_{H_0} := \mathcal{L}_{H_0} \cup \{(pw, rw)\}, \mathcal{L}_{H_1} := \mathcal{L}_{H_1} \cup \{(pw, r)\}, return r.
    In any other cases: Y \leftarrow \mathbb{R}, \mathcal{L}_{H_1} := \mathcal{L}_{H_1} \cup \{(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, Y)\}, and return Y.
 on H_i(X) from \mathcal A for i \in \{2,3,4\}:
    If ∃(X, Y ) ∈ LHi
: return Y
    Else: Y \leftarrow \{0,1\}^{\lambda}, \mathcal{L}_{H_i} := \mathcal{L}_{H_i} \cup \{(X,Y)\}, and return Y.
```
Fig. 7: Simulator Sim part II.

## **B Proof of Theorem 2: Tight Reduction for Our aPAKE Compiler from OW-ChCCA Secure KEM**

<span id="page-34-0"></span>If the underlying KEM in our compiler has OW-ChCCA security, then our our compiler becomes a tightness[-pr](#page-26-0)eserving one, as shown in Theorem 2. In this section, we will present the proof for Theorem 2.

We first recall the OW-ChCCA security [26] for KEM.

**Definition 6 (OW-ChCCA security for KEM [26]).** *For a KE[M](#page-26-0) scheme* KEM = (KeyGen*,* Encap*,* Decap)*, the advantage [fu](#page-26-0)nction of an adversary A for N*  $u$ *sers and*  $\ell$  *challenges is defined by*  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)}\text{-}\mathsf{chCCA}\n(\mathcal{A}) := \Pr\big[\mathsf{OW}\text{-}\mathsf{ChCCA}_{\mathsf{KEM}}^{\mathcal{A}} \Rightarrow 1\big],$  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)}\text{-}\mathsf{chCCA}\n(\mathcal{A}) := \Pr\big[\mathsf{OW}\text{-}\mathsf{ChCCA}_{\mathsf{KEM}}^{\mathcal{A}} \Rightarrow 1\big],$  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)}\text{-}\mathsf{chCCA}\n(\mathcal{A}) := \Pr\big[\mathsf{OW}\text{-}\mathsf{ChCCA}_{\mathsf{KEM}}^{\mathcal{A}} \Rightarrow 1\big],$ *where experiment* **OW-ChCCA** $_{\text{KEM}}^{A}$  *is defined in Fig. 8. The OW-ChCCA security for KEM requires*  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)\text{-}\mathsf{ChCCA}}(\mathcal{A}) = \mathsf{negl}(\lambda) \text{ for all PPT } \mathcal{A}.$  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)\text{-}\mathsf{ChCCA}}(\mathcal{A}) = \mathsf{negl}(\lambda) \text{ for all PPT } \mathcal{A}.$  $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)\text{-}\mathsf{ChCCA}}(\mathcal{A}) = \mathsf{negl}(\lambda) \text{ for all PPT } \mathcal{A}.$ 

<span id="page-34-1"></span>

OW-ChCCA $A_{\mathsf{KEM}}^{\mathcal{A}}(\lambda)$ :	Oracle $CORR(i)$
$pp \leftarrow$ Setup $(1^{\lambda})$	$\mathcal{L}_{CORR} := \mathcal{L}_{CORR} \cup \{i\}$
For $i \in [N]$ :	return $r_i$
$r_i \leftarrow \mathbb{R} R$	
$(pk_i, sk_i = r_i) \leftarrow Gen(pp; r_i)$	Oracle $\text{Dec}(i, c')$
$O := \{Enc, Dec, RevEAL, CORR, CHECK\}$	If $\exists K'$ s.t. $(i, c', K') \in \mathcal{L}_{\text{ENC}}$ : return $\bot$
$(i^*, c^*, K^*) \leftarrow \mathcal{A}^O(\mathsf{pp}, (pk_i)_{i \in [N]})$	return $K' := \text{Decap}(sk_i, c')$
If $(i^*, c^*, K^*) \notin \mathcal{L}_{\text{ENC}}$ : return 0	
If $i^* \in \mathcal{L}_{\text{CORR}}$ : return 0	<b>Oracle</b> $\text{Enc}(i)$ :
If $(i^*, c^*) \in \mathcal{L}_{REVEAL}$ : return 0	If $ \mathcal{L}_{\text{ENC}}  \geq \ell$ : return $\perp$
If $\text{CHECK}(i^*, c^*, K^*) = 1$ : return 1	$(c, K) \leftarrow$ Encap $(pk_i)$
	$\mathcal{L}_{\text{ENC}} := \mathcal{L}_{\text{ENC}} \cup \{(i, c, K)\}\$
<b>Oracle</b> REVEAL $(i, c)$ :	return $c$
If $\exists K$ s.t. $(i, c, K) \in \mathcal{L}_{\text{ENC}}$ :	
$\mathcal{L}_{REVEAL} := \mathcal{L}_{REVEAL} \cup \{(i, c)\}\$	<b>Oracle</b> CHECK $(i, c, K)$
return $K$	If Decap $(sk_i, c) = K$ :
return $\perp$	return 1
	return 0

Fig. 8: The OW-ChCCA security experiment for KEM.

Note that we take KEM as a canonical one, where the randomness  $r_i$  used in Gen is taken as the secret key  $sk_i$ . That is reflected in the OW-ChCCA security experiment, where the CORR oracle outputs the randomness  $r_i$  which is used to derive the key pair  $(pk_i, sk_i)$ .

In [26], Pan et. al proposed a specific tightly OW-ChCCA secure KEM based on the Matrix DDH assumption in the RO model. We review their KEM in Fig.10.

Next we recall Theorem 2 and present its proof.

**Theorem 2.** *If* KEM *is a KEM with OW-ChCCA security,* AE *is the informationtheoretically secure AE (shown in Sect. 4) with one-time authenticity and onetime CCA security, H*0*, H*1*, H*2*, H*3*, H*<sup>4</sup> *work as random oracles, then the aPAKE scheme in Fig. 5 securely emulates Fapake, hence achieving UC security in the*  ${F_{\text{make}}}, F_{\text{RO}}$ }-hybrid model. More precisely, suppose there are at most N par*ties and ℓ sessions, and A issues q ran[do](#page-26-1)m oracle queries, then there exists a simulator* Sim *such that*

$$
|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}\right]| \leq \mathsf{Adv}_{\mathsf{KEM}}^{(N^2,\ell)\text{-ChCCA}}(\mathcal{B}_{\mathsf{KEM}}) + \frac{2\ell + q^2 + q + 1}{2^{\lambda}}.\tag{1}
$$

*Proof.* Recall that if we use the information-theoretically secure  $AE$  scheme  $AE_{it}$ in our compiler, then  $\mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot-auth}}(\mathcal{B}_{\mathsf{AE}}) = \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ot-cca}}(\mathcal{B}_{\mathsf{AE}}) = 1/2^{\lambda}$ . Consequently, the security loss in the proof of Theorem 1 owes to the reduction between  $G_7$  and  $G_6$ , which is based on the OW-PCA security of KEM. Now we show that if KEM has OW-ChCCA security, then the reduction between  $G_7$  and  $G_6$  can be a tight one, and our compiler becomes a tightness-preserving one captured by (1).

Below we will give a tight reductio[n](#page-14-0) between  $G_7$  and  $G_6$  based on the OW-ChCCA security of KEM.

Recall that G<sup>7</sup> and G<sup>6</sup> differ only when event **Bad** happens.

**Bad:** Case II occurs to some  $(\mathsf{S}^{(j)}, \textit{iid}')$ ,  $\sigma = H_3(K_0, K_1, \textit{sid}|\psi)$ , but until then none of the Sim's (OfflineTestPW,  $C^{(i)}, S^{(j)}, pw'$ ) queries results in "correct guess".

With difference lemma, we know that  $|Pr[G_7 \Rightarrow 1] - Pr[G_6 \Rightarrow 1]| \leq Pr[\text{Bad in } G_7]$ *.* Next we show a reduction algorithm  $\mathcal{B}_{\mathsf{KEM}}$  and prove Pr  $[\mathbf{Bad}\,$  in  $\mathsf{G}_7] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(N^2,\ell)\text{-}\mathsf{ChCCA}}$ .

 $\mathcal{B}_{\mathsf{KEM}}$  obtains at most  $N^2$  public keys  $pk_{\mathsf{C}(i)}^{\mathsf{S}(j)}$  $S_{\text{C}^{(i)}}^{(j)}$  for each pair of  $(\text{C}^{(i)}, \text{S}^{(j)})$  and has access to oracles defined in Fig. 8.

In the reduction,  $B_{KEM}$  plays the role of the simulator Sim in  $G_7$ .  $B_{KEM}$  does the simulations as follows.

- $-$  **Before receiving** (StealPWFile[,](#page-34-1)  $C^{(i)}, S^{(j)})$  from  $\mathcal{A}$ :  $\mathcal{B}_{\mathsf{KEM}}$  behaves just like Sim does in  $G_7$ .
- $\bf{Upon}$   $\bf{receiving}$  (StealPWFile,  $\mathsf{C}^{(i)}, \mathsf{S}^{(j)})$   $\bf{from}$   $\mathcal{A}$ :  $\mathcal{B}_{\mathsf{KEM}}$  issues a StealPWFile query to  $\mathcal{F}_{apake}$ . If  $\mathcal{F}_{apake}$  replies with *pw* and "correct guess", then  $\mathcal{B}_{\mathsf{KEM}}$ behaves just like Sim does in  $G_7$ . More precisely,  $\mathcal{B}_{\mathsf{KEM}}$  computes  $rw :=$  $H_0(pw)$ , sets the trapdoor record as  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, rw, pk, sk, H_1(pw))$  where  $(pk, sk) \leftarrow \text{KeyGen}(H_1(pw))$ , and returns the password file  $(rw, pk)$ . Otherwise,  $\mathcal{B}_{\mathsf{KEM}}$  randomly samples  $rw \leftarrow \{0,1\}^{\lambda}$ , sets the trapdoor record as  $(C^{(i)}, S^{(j)}, rw, pk_{\sigma(i)}^{S^{(j)}})$  $S^{(j)}_{\text{C}^{(i)}}, sk = ?, r = ?$ ), and returns the password file  $(rw, pk_{\text{C}^{(i)}}^{S^{(j)}})$  $\zeta^{(i)}$ to *A*.
- **After receiving** (StealPWFile,  $C^{(i)}$ ,  $S^{(j)}$ ) from A: During this phase, for any offline attack from  $\mathcal{A}$ , if the corresponding (OfflineTestPW,  $\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, pw'$ ) query to  $\mathcal{F}_{apake}$  results in "correct guess", then **Bad** does not happen on any

session between instances  $(C^{(i)},iid)$  and  $(S^{(j)}, iid')$ . In this case,  $B_{KEM}$  queries  $CORR(\mathsf{C}^{(i)}, \mathsf{S}^{(j)})$  from its OW-ChCCA challenger to obtain the randomness *r* and then re-programmes  $H_0(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}) := rw$  and  $H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}) :=$ *r*. Then  $\mathcal{B}_{\mathsf{KEM}}$  updates its trapdoor record to  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, rw, pk_{\mathsf{C}^{(i)}}^{S^{(j)}})$  $k_{\text{C}^{(i)}}^{\text{S}^{(j)}}, sk_{\text{C}^{(i)}}^{\text{S}^{(j)}}$  $\mathcal{S}_{\mathsf{C}^{(i)}}^{\mathsf{S}^{(j)}},r$ .

For the session between instances  $(C^{(i)}, iid)$  and  $(S^{(j)}, iid')$ ,  $B_{KEM}$  first simulates the generation of PAKE key  $K_0$  without  $pw$  and trapdoor record, just like Sim. If Case II does not happen to instance  $(C^{(i)},iid)$  or  $(S^{(j)},iid')$ , then **Bad** event cannot happen to  $(S^{(j)}, \text{iid}')$  and  $\mathcal{B}_{\mathsf{KEM}}$  can simulate them without neither password *pw* nor the trapdoor record just like Sim does in G7. We show how  $B_{\text{KEM}}$  simulates instances  $(C^{(i)},iid)$  and  $(S^{(j)}, iid')$  in Case II below.

For server instance  $(S^{(j)}, \text{iid}')$ ,  $\mathcal{B}_{\mathsf{KEM}}$  can simulate the generation of  $\psi$ . If  $\mathcal{F}_{apake}$  returned *pw* and "correct guess" before, then  $\mathcal{B}_{\mathsf{KEM}}$  generates  $\psi$  just like Sim does in  $G_7$ . Otherwise,  $B_{\text{KEM}}$  queries its challenge oracle to obtain  $c^*$  *←* Enc( $C^{(i)}$ ,  $S^{(j)}$ ) to obtain a challenge ciphertext  $c^*$ . Then  $B_{KEM}$  invokes  $\psi \leftarrow \mathsf{AE}.\mathsf{Enc}(H_2(K_0), c^*)$ . Since  $c^*$  is an encapsulation under  $pk_{c^{(i)}}^{S^{(j)}}$  $\mathcal{B}_{\mathsf{C}^{(i)}}^{\mathsf{S}^{(j)}}, \mathcal{B}_{\mathsf{KEM}}^{\mathsf{N}}$ 's simulation of  $\psi$  is perfect.

For client instance  $(C^{(i)},iid)$ , if  $B_{KEM}$  has the trapdoor record  $(C^{(i)}, S^{(j)}, rw, pk, sk \neq 0)$ ?, *r*), then  $B_{\text{KEM}}$  can use *sk* to simulate  $\sigma$  and the session key just like Sim does in  $G_7$ . Otherwise,  $\mathcal{B}_{\mathsf{KEM}}$  has no  $sk = sk_{\mathsf{C}(i)}^{S(j)}$  $\mathcal{L}^{S^{(J)}}_{\mathsf{C}^{(i)}}$  now. To simulate  $\sigma$  and the session key,  $\mathcal{B}_{\mathsf{KEM}}$  resorts to ROs and the re-programming techniques: upon  $(\mathsf{C}^{(i)},iid)$  receiving  $\psi'$ ,  $\mathcal{B}_{\mathsf{KEM}}$  invokes  $c \leftarrow \mathsf{AE}.\mathsf{Dec}(H_2(K_0), \psi')$ . If  $c = \bot$ ,  $\mathcal{B}_{\mathsf{KEM}}$  behaves just like Sim (no  $sk_{\mathcal{L}^{(i)}}^{\mathsf{S}^{(j)}}$  $C^{(i)}_{(i)}$  involved). If  $c \neq \bot$ , for all hash queries  $H_3(K_0, K_1, sid|\psi\rangle)$ ,  $\mathcal{B}_{\mathsf{KEM}}$ first checks whether there exists  $K_1$  such that  $\text{CHECK}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, c, K_1) = 1$ , if yes, set  $\sigma := H_3(K_0, K_1, sid|\psi\rangle)$ . Otherwise, randomly choose  $\sigma \leftarrow s \{0,1\}^{\lambda}$ , and set  $H_3(K_0, ?, sid|\psi') := \sigma$  with ? denoting the undermined value of  $K_1$ . It then returns  $\sigma$  to  $\mathcal A$  and returns sKey :=  $H_2(K_0)$  as the session key. If later  $\mathcal A$  issues any new query  $H_3(K_0, K'_1, sid|\psi')$ ,  $\mathcal{B}_{\mathsf{KEM}}$  checks whether  $\text{CHECK}(c, K'_1) = 1$ . If  $y$ es,  $\mathcal{B}_{\mathsf{KEM}}$  re-programme  $H_3(K_0, K'_1, sid|\psi') := \sigma$ .

Upon  $(\mathsf{S}^{(j)}, \text{iid}')$  receiving  $\sigma'$ . If there is no  $H_3$  query such that  $\sigma' = H_3(K_0, K'_1, \text{sid}|\psi)$ , then  $\sigma' = H_3(K_0,K_1,sid|\psi)$  with negligible probability.  $\mathcal{B}_{\mathsf{KEM}}$  sends  $(\mathsf{Abort},\mathsf{S}^{(j)},iid')$ to  $\mathcal{F}_{apake}$ , and accordingly  $\mathcal{F}_{apake}$  returns  $\bot$ , sKey :=  $\bot$  to  $(\mathsf{S}^{(j)},iid')$  to reject the session. Otherwise, from *A* hash query such that  $\sigma' = H_3(K_0, K'_1, sid|\psi)$ ,  $\mathcal{B}_{\mathsf{KEM}}$  checks whether CHECK( $\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, c, K'_1$ ) = 1. If no, then  $\sigma' \neq H_3(K_0, K_1 =$  $\mathsf{Decap}(sk_{\mathsf{C}^{(i)}}^{\mathsf{S}^{(j)}})$  $S_{\mathsf{C}^{(i)}}^{(j)}, c^*$ ,  $sid|\psi)$  and  $\mathcal{B}_{\mathsf{KEM}}$  sends (Abort,  $S^{(j)}, iid'$ ) to  $\mathcal{F}_{apake}$ . If yes,  $\mathrm{CHECK}(\mathsf{C}^{(i)}, S^{(j)}, c^*, K_1')$ 1 implies  $K'_1 = \mathsf{Decap}(sk_{\mathsf{C}^{(i)}}^{\mathsf{S}^{(j)}})$  $\mathcal{S}^{(j)}_{\mathsf{C}^{(i)}}, c^*$ ). Then  $\mathcal{B}_{\mathsf{KEM}}$  sends (CorruptKey,  $\mathsf{S}^{(j)},iid', sid|\psi|\sigma', H_4(K_0)$ to  $\mathcal{F}_{apake}$ . Then  $\mathcal{F}_{apake}$  returns  $H_4(K_0)$  to  $(\mathsf{S}^{(j)},iid')$ .

Note that during the whole simulation,  $B_{KEM}$  never queries DEC oracle and REVEAL oracle.  $\mathcal{B}_{\mathsf{KEM}}$  only queries CORR oracle if  $\mathcal{F}_{apake}$  returns *pw* and "correct guess" corresponding to some OfflineTestPW query. Therefore, if Bad event happens,  $B_{\text{KEM}}$  does not query CORR oracle or REVEAL oracle corresponding to the challenge ciphertext *c <sup>∗</sup>* but successfully obtains the encapsulated key  $K_1 = \mathsf{Decap}(sk_{\bm{\mathsf{C}}^{(i)}}^{\mathsf{S}^{(j)}})$  $\mathcal{L}_{\mathsf{C}^{(i)}}^{(j)}, c^*$ ). Finally,  $\mathcal{B}_{\mathsf{KEM}}$  submits  $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, K_1)$  and wins the OW-

ChCCA game. So we have

$$
\left|\Pr\left[\mathsf{G}_7 \Rightarrow 1\right]-\Pr\left[\mathsf{G}_6 \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Bad}\text{ in }\mathsf{G}_7\right] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(N^2,\ell)\text{-}\mathsf{ChCCA}}.
$$

Collecting the statistical loss  $\frac{2\ell + q^2 + q + 1}{2\lambda}$ , we obtain (1).  $\Box$ 

## **C Proof of Lemma 1: Security Proof for the Information-Theoretic AE Scheme**

Recall that our AE construction works as follows.

- <span id="page-37-0"></span> $-$  AE.Enc( $k = (k_1, k_2, k_3), m$  $k = (k_1, k_2, k_3), m$  $k = (k_1, k_2, k_3), m$ ) : it generates a ciphertext  $c = (c_1 = k_1 + m, c_2 = 1)$  $k_2 \times c_1 + k_3$ .
- $-$  AE.Dec( $k = (k_1, k_2, k_3), c = (c_1, c_2)$ ) : it first checks whether  $c_2 = k_2 \times c_1 + k_3$ . If yes, it outpus *c*<sup>1</sup> *− k*1. Otherwise, it outputs *⊥*.

Next we present the proof of the one-time authenticity and one-time CCA security of our AE.

*Proof of one-time authenticity.* The one-time authenticity considers an adversary *A* who obtains an honestly generated challenge ciphertext of message *m<sup>∗</sup>* chosen by *A*, i.e.,

$$
c^* = (c_1^* = k_1 + m^*, c_2^* = c_1^* \times k_2 + k_3),
$$

and requires that *A* can hardly generate a new and valid ciphertext  $c = (c_1, c_2)$ such that  $c \neq c^*$  but  $\mathsf{AE}.\mathsf{Dec}(k, c) \neq \bot$ , i.e.,  $c_2 = c_1 \times k_2 + k_3$ .

We consider the following two cases.

- **Case 1:**  $c_1 \neq c_1^*$ . In this case,  $c_2 = c_1 \times k_2 + k_3$  implies that  $k_2 = (c_2 c_2^*)/(c_1 c_2^*)$  $c_1^*$ ). Due to the uniformity of key  $k_3$ ,  $c^* = (c_1^*, c_2^*)$  leaks no information of  $k_2$ . With the uniformity of key  $k_2$ , the event  $k_2 = (c_2 - c_2^*)/(c_1 - c_1^*)$  happens only with  $\frac{1}{2^{\lambda}}$ .
- **Case 2:**  $c_1 = c_1^*$  but  $c_2 \neq c_2^*$ . In this case, the ciphertext *c* submitted by *A* must be invalid since  $c_2 = c_1 \times k_2 + k_3 = c_1^* \times k_2 + k_3 = c_2^* \neq c_2$ .

Overall, *A* can hardly generate a new valid ciphertext, except with probability at most  $1/2^{\lambda}$ , and the one-time authenticity follows.

*Proof of one-time CCA security.* The one-time CCA security asks *A* to determine whether a challenge ciphertext  $c^* = (c_1^* = k_1 + m_b, c_2^* = k_2 \times c_1^* + k_3)$ encrypts  $m_0$  or  $m_1$ , where  $m_0$  and  $m_1$  are chosen by A itself, even if A has one-time access to the Dec oracle, which decrypts ciphertexts different from *c ∗* .

Due to the one-time authenticity of AE (as we shown above), the adversary can hardly generate a new valid ciphertext, so the Dec oracle would always return *⊥* (except with probability at most 1*/*2 *λ* ).

As such, only the challenge ciphertext  $c^* = (c_1^* = k_1 + m_b, c_2^* = c_1^* \times k_2 + k_3)$ might be useful for  $A$  to learn the challenge bit  $b$ . However, by the uniformity of  $k_1$ ,  $c_1^* = k_1 + m_b$  perfectly hides  $m_b$ . Consequently,  $m_b$  is hidden in  $c^*$  so that *A* can hardly guess *b* correctly, except with advantage at most  $1/2^{\lambda}$ , and the one-time CCA security follows. *⊓⊔*

## **D Comparison with Other aPAKE Compilers from Post-quantum Assumptions**

<span id="page-38-1"></span><span id="page-38-0"></span>Table 3: Efficiency comparison with aPAKE compilers [15] and [24]. For computing and communication cost, we use the mode of "**Client + Server = Total**" to reflect the separation of client/server computation/communication. Efficiency comparison with post-quantum aPAKE compilers in [15] instantiated from NIST PQC winner signature schemes and in [24] ins[tan](#page-30-0)tiated [fro](#page-30-1)m CSIDH. All algorithms except Falcon are used in their versions of NIST security level 3 (i.e., security comparable to breaking AES-192). As for Falcon, we use its level 1 version, due to its lack of level 3.



In Table 3, we compare our compiler with known aPAKE compilers instantiated from other post-quantum assumptions, including the signature-based compiler  $[15]$  and the isogeny-based compiler  $[24]$ . When  $[15]$  is instantiated with the NIST PQC hash-based signature scheme Sphincs<sup>+</sup>, they have smaller storage than ours, b[ut](#page-38-0) their communication cost is dozens of times larger than ours and their registration time and computing time are dozens of times slower than ours. Whe[n th](#page-30-0)e isogeny-based compiler [24] is [ins](#page-30-1)tantiate[d w](#page-30-0)ith CSIDH, their server storage and communication cost are smaller than ours, but their registration time and computing time are ten thousands times slower than ours. This shows that our compiler overwhelms these compilers in terms of registration time and computing time, and has compara[ble](#page-30-1) server storage and communication cost.

### **E Concrete aPAKE Schemes**

### **E.1 Kyber-Based aPAKE Scheme from Lattices**

<span id="page-38-2"></span>By applying our Kyber-based compiler to the UC-secure Kyber-based PAKE scheme [2], we obtain A Kyber-based aPAKE scheme, which is shown in Fig. 9.

#### **E.2 Tightly UC-Secure aPAKE Scheme from MDDH**

By appl[yi](#page-29-2)ng our MDDH-based compiler to the UC-secure CDH-based PA[KE](#page-39-0) scheme [22], we obtain a tightly UC secure aPAKE scheme based on the MDDH assumption, which is shown in Fig. 11.

<span id="page-39-0"></span>*•* Kyber = (Kyber*.*KeyGen*,* Kyber*.*Encap*,* Kyber*.*Decap). • Ideal ciphers  $IC_1$  on domain  $\mathcal{PK}$  and  $IC_2$  on domain  $\mathcal{CT}$ , where  $\mathcal{PK}$  and  $\mathcal{CT}$  are public key space and ciphertext space of Kyber. • Hash functions  $H_0, H_2, H_3, G : \{0, 1\}^* \to \{0, 1\}^{\lambda}$ ,  $H_1$ :  $\{0,1\}^* \to \mathcal{R}$ *// R* is the randomness space of Kyber.KeyGen Client C (*i*) : Server  $S^{(j)}$ : - - - - - - - - Registration on input *pw*:  $rw := H_0(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ ;  $r := H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ (*pk, sk*) *←* Kyber*.*KeyGen(*r*) *rw, pk −−−−−−−−−−−→* store (*rw, pk*) - Login - on input *pw*: on input (*rw, pk*):  $(\hat{pk}, \hat{sk}) \leftarrow$  Kyber.KeyGen  $C_1 := \mathsf{IC}_1.\mathsf{Enc}(H_0(pw), \hat{pk})$  $\hat{c} := \mathsf{IC}_2.\mathsf{Dec}(H_0(pw), C_2)$  $\hat{K} \leftarrow$  Kyber.Decap $(\hat{sk}, \hat{c})$  $K_0 := G(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, C_1, C_2, \hat{K})$  $c \leftarrow \overline{\mathsf{AE}_{it}.\mathsf{Dec}(H_2(K_0), \psi)}$ Abort if  $c = \perp$  $r := H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ (*pk, sk*) *←* Kyber*.*KeyGen(*r*) *K*<sup>1</sup> *←* Kyber*.*Decap(*sk, c*)  $sid := C_1 | C_2$  $\sigma := H_3(K_0, K_1, sid|\psi)$  $\big|$ sKey :=  $H_4(K_0)$ *<sup>C</sup>*<sup>1</sup> *−−−−−−−−→*  $C_2, \psi$ *←−−−−−−−−−−− σ −−−−−−−−→*  $\hat{pk} := \mathsf{IC}_1.\mathsf{Dec}(rw, C_1)$  $(\hat{c}, \hat{K}) \leftarrow$  Kyber. Encap $(\hat{pk})$  $C_2 := \mathsf{IC}_2.\mathsf{Enc}(rw, \hat{c})$  $K_0 := G(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, C_1, C_2, \hat{K})$  $(c, K_1) \leftarrow$  Kyber.Encap $(pk)$  $\psi \leftarrow \mathsf{AE}_{it}$ .Enc $(H_2(K_0), c)$ *sid* :=  $C_1|C_2$  $\forall$ Abort if  $\sigma \neq H_3(K_0, K_1, sid|\psi)$  $\textsf{K}$ ey :=  $H_4(K_0)$ 

Fig. 9: The Kyber-based aPAKE scheme resulting from compiling the Kyber-based PAKE in [2] with our Kyber-based compiler. The dashed part is our Kyber-based compiler. Kyber = (Kyber*.*KeyGen*,* Kyber*.*Encap*,* Kyber*.*Decap) denotes the CCA-secure Kyber.

In the [fo](#page-29-2)llowing, we also recall the definition of the MDDH assumption and present the MDDH-based OW-ChCCA KEM scheme proposed in [26] in Fig. 10. In the MDDH-based KEM scheme, we use [*a*] to represent a group element  $g^a$  and naturally extend to vectors  $[\mathbf{v}]$  and matrices  $[\mathbf{A}]$ . The UC-security of the CDH-based PAKE scheme in [22] is recalled in Lemma 2.

**Definition 7 (Matrix DDH Assumption [13]).** *Let*  $k \in \mathbb{N}$  *a[nd](#page-30-4)* GGen *is [a g](#page-40-0)roup generation algorithm that outputs*  $(\mathbb{G}, g, p)$  *on input*  $1^{\lambda}$ *. The k-MDDH assumption holds, if for any PPT A, we have*

$$
\mathsf{Adv}_\mathsf{MDDH}(\mathcal{A}) := \left| \Pr\left[ \mathcal{A}(\mathbb{G}, g, p, [\mathbf{A}], [\mathbf{b}]) \Rightarrow 1 \right] - \Pr\left[ \mathcal{A}(\mathbb{G}, g, p, [\mathbf{A}], [\mathbf{u}]) \Rightarrow 1 \right] \right| = \mathsf{negl}(\lambda),
$$
\n
$$
\text{where } (\mathbb{G}, g, p) \leftarrow \mathsf{GGen}, \mathbf{A} \leftarrow \mathbb{Z}_p^{(k+1) \times k}, \mathbf{x} \leftarrow \mathbb{Z}_p^k, \mathbf{b} := \mathbf{A}\mathbf{x}, \mathbf{u} \leftarrow \mathbb{Z}_p^{k+1}.
$$

**Lemma 2 ([26], Theorem 4).** *The KEM scheme in Fig. 10 is tightly OW-ChCCA secure. Concretely, for any PPT adversary*  $A$ *, for any*  $N = \text{poly}(\lambda)$  *and*   $\ell = \text{poly}(\lambda)$ *, there exists a PPT algorithm B such that* 

$$
\mathsf{Adv}_{\mathsf{KEM}}^{(N,\ell)\text{-}\mathsf{ChCCA}} \le 12 \cdot \mathsf{Adv}_{\mathsf{MDDH}}(\mathcal{B}) + 2^{-\varOmega(\lambda)}.
$$

**Lemma 3 ([22], Theorem 2).** *The PAKE scheme in Fig. 11 securely emulates Fpake. Concretely, we have*

$$
|\Pr[\text{Real}_{\mathcal{Z},\mathcal{A}}] - \Pr[\text{Ideal}_{\mathcal{Z},\text{Sim}}]| \leq 2 \cdot \text{Adv}_{\text{CDH}}(\mathcal{B}) + 2^{-\Omega(\lambda)}.
$$

Finally, w[hen](#page-30-5) applying our MDDH-based compiler to t[he t](#page-41-0)ightly UC-secure PAKE scheme [22], we obtain the first tightly UC-secure aPAKE scheme, as shown in Fig. 11.

<span id="page-40-0"></span>

Setup $(1^{\lambda})$ :	$Decap(sk, c)$ :
$(\mathbb{G}, p, q) \leftarrow \mathsf{GGen}(1^{\lambda})$	Parse $c = (C_0, C_1, [\mathbf{x}], \hat{h}_0, \hat{h}_1))$
$[\mathbf{A}] \leftarrow_{\$} \mathbb{G}^{3 \times 3}$	Parse $sk = (\mathbf{z}_b, b)$
Return $pp := (\mathbb{G}, p, g, [\mathbf{A}])$	$h'_b := \hat{h}_b \oplus [\mathbf{z}_b^{\top} \mathbf{x}] \in \{0, 1\}^{\log p}$
	$\hat{K}_b := H([\mathbf{x}], \hat{h}_b, h'_b)$
KeyGen(pp)	$R := C_h \oplus K_h$
$b \leftarrow s \{0,1\}, \mathbf{z}_b \leftarrow s \mathbb{Z}_p^3$	$({\bf s}, h_0, h_1) := G(R)$
$[\mathbf{u}_b] := [\mathbf{A}\mathbf{z}_b], [\mathbf{u}_{1-b}] \leftarrow \mathbb{s} \mathbb{G}^3$	$\hat{h}'_{1-h} := [\mathbf{s}^\top \mathbf{u}_{1-b}] \oplus h_1$
$pk := ([\mathbf{u}_0], [\mathbf{u}_1]), sk := (\mathbf{z}_b, b)$	$\hat{K}_{1-b} := H([\mathbf{x}], \hat{h}'_{1-b}, h_{1-b})$
	If $\mathbf{x} \neq [\mathbf{A}^\top \mathbf{s}]$ : return $\perp$
$\mathsf{Encap}(pk)$ :	If $\hat{K}_{1-b} \oplus R \neq C_{1-b}$ : return.
$R \leftarrow \{0,1\}^{\lambda}, (\mathbf{s}, h_0, h_1) := G(R)$	If $h'_b \neq h_b$ : return $\perp$
$[\mathbf{x}] := [\mathbf{A}^\top s] \in \mathbb{G}^3$	If $h'_{1-h} \neq h_{1-b}$ : return $\perp$
$\hat{h}_0 := [\mathbf{s}^\top \mathbf{u_0}] \oplus h_0 \in \{0,1\}^{\log p}$	return $K := R$
$\hat{h}_1 := [\mathbf{s}^\top \mathbf{u}_1] \oplus h_1 \in \{0, 1\}^{\log p}$	
$K_0 := H([\mathbf{x}, h_0, h_0]), C_0 := \hat{K}_0 \oplus R$	
$\hat{K}_1 := H([\mathbf{x}, \hat{h}_1, h_1]), C_1 := \hat{K}_1 \oplus R$	
$c := (C_0, C_1, [\mathbf{x}], \hat{h}_0, \hat{h}_1)$	
return $(c, K := R)$	

Fig. 10: The MDDH based OW-ChCCA secure KEM scheme in [26].

<span id="page-41-0"></span>*•* KEMMDDH := (KEMMDDH*.*KeyGen*,* KEMMDDH*.*Encap*,* KEMMDDH*.*Decap) *•* A group (G*, g, p*).  $\bullet$  Two ideal ciphers  $\textsf{IC}_1$  on domain  $\mathbb{G}\times\mathbb{G}$  and  $\textsf{IC}_2$  on domain  $\mathbb{G}.$ • Hash functions  $H_0, H_2, H_3, G : \{0, 1\}^* \to \{0, 1\}^{\lambda}$ ,  $H_1$  :  $\{0,1\}^* \to \mathcal{R}$ */* / *R* is the randomness space of KEM<sub>MDDH</sub>.KeyGen Client C (*i*) : Server  $S^{(j)}$ : - - - - - - - - - - - - Registration on input *pw*:  $rw := H_0(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ ;  $r := H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ (*pk, sk*) *←* KEMMDDH*.*KeyGen(*r*) *rw, pk −−−−−−−−−−−→* store (*rw, pk*) - Login - on input *pw*: on input (*rw, pk*):  $\overline{x_1, x_2 \leftarrow \mathbb{R} \mathbb{Z}_p}, X_1 := g^{x_1}, X_2 := g^{x_2}$  $e_1$  ← *IC*<sub>1</sub>.Enc( $H_0(pw)$ ,  $X_1|X_2$ )  $Y \leftarrow \text{IC}_2 \text{.Dec}(H_0(pw), e_2)$  $sid := C|S|e_1|e_2$  $K_0 := G(sid, Y^{x_1}, Y^{x_2}, pw)$  $c \leftarrow \mathsf{AE}_{it}.\mathsf{Dec}(H_2(K_0), \psi)$ Abort if  $c = \perp$  $r := H_1(pw, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$  $|(pk, sk) \leftarrow \textsf{KEM}_{\textsf{MDDH}}.\textsf{KeyGen}(r)$  $|K_1$  ← KEM<sub>MDDH</sub>.Decap $(sk, c)$  $\phi = H_3(K_0, K_1, sid|\psi)$  $sKey := H_4(K_0)$ *<sup>e</sup>*<sup>1</sup> *−−−−−−−−→*  $e_2, \begin{bmatrix} \psi \end{bmatrix}$ *<sup>σ</sup> −−−−−−−−→*  $y \leftarrow \mathcal{Z}_p, Y := g^y$  $e_2 \leftarrow \text{IC}_2.\text{Enc}(rw, Y)$  $X_1|X_2 \leftarrow \textsf{IC}_1.\textsf{Dec}(rw, e_1)$  $sid := \mathsf{C}|\mathsf{S}|e_1|e_2$  $K_0 := G(sid, X_1^y, X_2^y, pw)$  $(c, K_1) \leftarrow \mathsf{KEM}_{\mathsf{MDDH}}.\mathsf{Encap}(pk)$  $\psi \leftarrow \mathsf{AE}_{\mathsf{it}}.\mathsf{Enc}(H_2(K_0),c)$ Abort if  $\sigma := H_3(K_0, K_1, sid|\psi)$  $s$ Key :=  $H_4(K_0)$ 

Fig. 11: The tightly secure MDDH-based aPAKE scheme resulting from compiling the CDH-based PAKE in [22] with our KEM-based compiler. The dashed part is our KEM-based compiler installed with  $KEM_{MDDH}$  = (KEMMDDH*.*KeyGen*,* KEMMDDH*.*Encap*,* KEMMDDH*.*Decap) shown in Fig. 10, which is a tightly secure OW-ChCCA secure KEM scheme in [26].

# **Table of Contents**

