Privacy-Preserving Authenticated Key Exchange in the Standard Model

You Lyu[1,2], Shengli Liu[1,2,4](✉), Shuai Han[2,3], and Dawu Gu[1]

[1] Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai 200240, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
[4] Westone Cryptologic Research Center, Beijing 100070, China
{vergil,slliu,dalen17,dwgu}@sjtu.edu.cn

Abstract. Privacy-Preserving Authenticated Key Exchange (PPAKE) provides protection both for the session keys and the identity information of the involved parties. In this paper, we introduce the concept of robustness into PPAKE. Robustness enables each user to confirm whether itself is the target recipient of the first round message in the protocol. With the help of robustness, a PPAKE protocol can successfully avoid the heavy redundant communications and computations caused by the ambiguity of communicants in the existing PPAKE, especially in broadcast channels.

We propose a generic construction of robust PPAKE from key encapsulation mechanism (KEM), digital signature (SIG), message authentication code (MAC), pseudo-random generator (PRG) and symmetric encryption (SE). By instantiating KEM, MAC, PRG from the DDH assumption and SIG from the CDH assumption, we obtain a specific robust PPAKE scheme in the standard model, which enjoys forward security for session keys, explicit authentication and forward privacy for user identities. Thanks to the robustness of our PPAKE, the number of broadcast messages per run and the computational complexity per user are constant, and in particular, independent of the number of users in the system.

Keywords: Authenticated key exchange · Privacy · Robustness.

1 Introduction

Authenticated Key Exchange (AKE) enables two parties to authenticate each other and compute a shared session key. It has been widely deployed over Internet, like IPsec IKE (Internet Key Exchange), TLS, Tor, Google's QUIC protocol, etc. Generally, AKE focuses on the protection of session keys between two parties against adversaries implementing both passive and active attacks. As a well-studied topic, a variety of AKE schemes have been proposed, but little attention was paid to privacy of user identities in AKE. The research on Privacy-Preserving AKE (PPAKE) was ignited by the chasing of privacy protection. For instance, SKEME [14], TLS 1.3 [3], Tor [8] and private airdrop [12] all take user privacy as one of important design principles. Recently two proposals for PPAKE arise [20,19], aiming to provide protection for user identity besides their session keys. Next we overview the recent two works, namely SSL-PPAKE [20] and RSW-PPAKE [19].

SSL-PPAKE. In [20], Schäge, Schwenk, and Lauer (SSL) isolated a generic PPAKE construction from TLS 1.3, QUIC, IPsec IKE, SSH and certain patterns of NOISE to achieve user identity protection. We name it SSL-PPAKE.

SSL-PPAKE [20] has 4 rounds. In the first two rounds, P_i and P_j run a basic Diffie-Hellman (DH) handshake to obtain a shared DH key $K = g^{xy}$. In the last two rounds, P_i and P_j use the shared DH key $K = g^{xy}$ to protect protocol messages that contain identity-related data such as identities, public keys or digital signatures. As pointed out in [20], due to the lack of authenticity in the first two rounds, the SSL-PPAKE suffers a weakness on preserving the privacy of initiator's identity. More precisely, let us consider a broadcast channel with μ users as an example. First we identify three facts about SSL-PPAKE.

Fact 1. In the 1st round, to protect the identity of its intended target recipient P_j, initiator P_i has to broadcast g^x in the system. As a result, every user is able to receive g^x.

Fact 2. In the 2nd round, every user P_{j_k} has to respond to P_i by broadcasting $g^{y_{j_k}}$, here $j_k \in [\mu] \setminus \{i\}$, since P_{j_k} is uncertain about the intended recipient.

Fact 3. In the 3rd round, P_i receives all the messages $\{g^{y_{j_k}}\}_{j_k \in [\mu]\setminus\{i\}}$, but it is not able to identify the right message sent from the intended party P_j and has to computes all DH keys $\{K_{i,j_k} = g^{xy_{j_k}}\}_{j_k \in [\mu]\setminus\{i\}}$. Consequently, P_i has to encrypt the message in the third round with each K_{i,j_k} individually to obtain $\mu - 1$ ciphertext $C_{j_k} = \mathsf{SE.Enc}(K_{i,j_k}, i|pk_i|auth_i)$ and broadcast the $\mu - 1$ ciphertexts to all users. Here $\mathsf{SE.Enc}$ denotes a symmetric encryption algorithm, and $auth_i$ denotes the authentication part of the protocol.

Now let us see how an adversary reveals the identity of the initiator. After receiving g^x from P_i, the adversary can simply select $\tilde{y}$ and send $g^{\tilde{y}}$ to P_i. According to the facts, P_i will broadcast $\tilde{C} = \mathsf{SE.Enc}(\tilde{K} = g^{x\tilde{y}}, i|pk_i|auth_i)$ in the 3rd round. Then the adversary can compute $\tilde{K} = (g^x)^{\tilde{y}}$ and easily decrypt $\tilde{C}$ with $\tilde{K}$ to obtain the identity information $i|pk_i$.

RSW-PPAKE. To deal with the active attacks on the SSL-PPAKE scheme, Ramacher, Slamanig and Weninger (RSW) [19] proposed three solutions in the Random Oracle model.[5] The first one has 3 rounds and assumes pre-shared key between every pair of users. It resorts to the pre-shared key to accomplish authentication. The third one converts an AKE to a PPAKE by encrypting every message of AKE with communication peer's public key. However, it does not achieve forward privacy for user identities. If any user's secret key is corrupted, the adversary can break forward privacy by decrypting the ciphertexts in the

[5] No security proofs are provided for the three schemes in [19] and its full-version is still not available.

previous runs to reveal the used identities. The second solution has 4 rounds and does not possess forward privacy when the responder is corrupted. Here we recall the second scheme and show the weakness on its forward privacy.

- In the first two rounds, similar to SSL-PPAKE, a Diffie-Hellman handshake is implemented to share key $K = g^{xy}$ between P_i and P_j. Meanwhile, P_i has to handshake with every P_{j_k} and share $K_{i,j_k} = g^{xy_{j_k}}$ with P_{j_k}, $j_k \in [\mu] \setminus \{i\}$.
- In the 3rd round, P_i uses P_j's public key pk_j to encrypt a random string r and obtains $C = \mathsf{PKE.Enc}(pk_j, r)$, where $\mathsf{PKE.Enc}$ denotes a public-key encryption algorithm. Then it uses K to encrypt C to obtain a $c_0 = \mathsf{SE.Enc}(K, C)$. P_i signs $i|j|c_0|g^x|g^y$ to get the signature σ_i and encrypts its certificate cert_i and σ_i with a derived key $K' = H(K, r, g^x, g^y)$, resulting in $c_1 = \mathsf{SE.Enc}(K', \mathsf{cert}_i|\sigma_i)$. In the real scenario, P_i cannot identify the right K from $\{K_{i,j_k}\}_{j_k\in[\mu]\setminus\{i\}}$, thus has to use each K_{i,j_k} to obtain (c_{0,j_k}, c_{1,j_k}). Finally, P_i broadcasts $\{(c_{0,j_k}, c_{1,j_k})\}_{j_k\in[\mu]\setminus\{i\}}$ to all users.
- In the 4th round, each user j_k decrypts every pair in $\{(c_{0,j_k}, c_{1,j_k})\}_{j_k\in[\mu]\setminus\{i\}}$ with its Diffie-Hellman key $K_{i,j_k} = g^{xy_{j_k}}$, trying to recover $\mathsf{cert}_i|\sigma_i$. Only the right responder P_j can certify the validity of $\mathsf{cert}_i|\sigma_i$ and recover r. After that, P_j knows its partner is P_i. Then P_j broadcasts the hash value $h := H(r, i|j|g^x|g^y|c_0|c_1)$ to P_i.
- Finally, P_i checks if $h = H(r, i|j|g^x|g^y|c_0|c_1)$ holds (to authenticate P_j).

The attack is similar to that on SSL-PPAKE but here on forward privacy of RSW-PPAKE. After receiving g^x from P_i, the adversary $\mathcal{A}$ can simply select $\tilde{y}$ and send $g^{\tilde{y}}$ to P_i. Then $\mathcal{A}$ also shares a key $\tilde{K} = g^{x\tilde{y}}$ with P_i. In the second phase, there must exist $(\tilde{c_0}, \tilde{c_1}) \in \{(c_{0,j_k}, c_{1,j_k})\}_{j_k\in[\mu]\setminus\{i\}}$ such that $(\tilde{c_0}, \tilde{c_1})$ is computed with $\tilde{K}$. So $\mathcal{A}$ can always recover $C = \mathsf{SE.Dec}(\tilde{K}, \tilde{c_0})$. Later $\mathcal{A}$ corrupts P_j and obtains sk_j. Then $\mathcal{A}$ decrypts C with sk_j to recover $r = \mathsf{PKE.Dec}(sk_j, C)$. Finally $\mathcal{A}$ can identify P_i, P_j by finding $i, j, c_{0,j}, c_{1,j}$ such that $h = H(r, i|j|g^x|g^y|c_{0,j}|c_{1,j})$.

Our Approach to PPAKE. From the above analysis, we know that the SSL-PPAKE provides no protection for the initiator's identity, and the RSW-PPAKE loses forward privacy for identities of both the initiator and the responder when the responder is corrupted.

The reason for the attacks lies in the facts that each user replies the initiator and the initiator cannot identify the message sent from the intended peer in the 2nd round. Thus the initiator has to reply messages to each individual user in the third round. This leaks too much information, of which the adversary can take advantage to break privacy of PPAKE, as shown before.

At the same time, these facts also lead to another drawback: the communication band of the protocol is as large as $O(\mu)$ and each user's computational complexity is as high as $O(\mu)$, since each user has to compute or deal with $\mu - 1$ messages in the 3rd round. Here μ is the number of users in the system.

In this paper, we study how to avoid the above attacking problems and improve efficiency of PPAKE. Our idea in a nutshell is to make PPAKE robust.

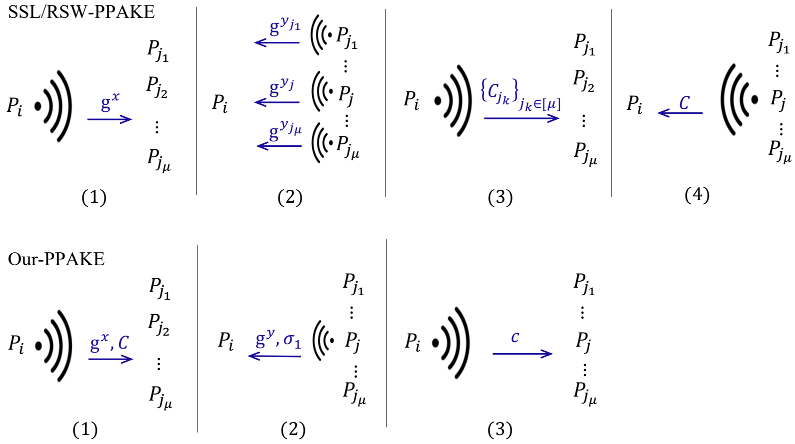

Fig. 1: The upper part is the information flows of rounds (1)(2)(3)(4) in SSL-PPAKE and RSW-PPAKE [20,19]. The lower part is the information flows of rounds (1)(2)(3) in our robust PPAKE. Here the parties communicate over a broadcast channel.

Robustness of PPAKE. We introduce the concept of robustness. It requires that only one party P_j is able to ascertain that the message in the 1st round is for him/her, hence correctly reply a message in the 2nd round.

Our robust PPAKE makes use of a key encapsulation mechanism KEM, a signature scheme SIG, a message authentication code MAC, a pseudo-random generator PRG and a symmetric encryption SE. The public/secret key pair (pk, sk) of KEM and the verification/signing key (vk, ssk) of SIG serve as the long-term key of a user. Our PPAKE has 3 rounds and is shown below.

Round 1 $(P_i \Rightarrow P_j)$: P_i broadcasts g^x and a ciphertext C to P_j, where $(C, N) \leftarrow \mathsf{KEM.Encap}(pk_j)$ with N the key encapsulated in C.

Round 2 $(P_i \Leftarrow P_j)$: P_j decrypts C with its secret key sk_j to recover N, then it uses N as the MAC key to compute a MAC tag $\sigma_1 = \mathsf{MAC}(N, g^x|C)$. P_j broadcasts (g^y, σ_1). We require that when decrypting C, only P_j succeeds and all other parties will get a special failure symbol $\perp$, which is guaranteed by the robustness of KEM (see more details later). Consequently, only P_j responds in this round, and all other parties (except P_i and P_j) will terminate the protocol in time.

Round 3 $(P_i \Rightarrow P_j)$: P_i checks the validity of σ_1 and computes the Diffie-Hellman key $K = g^{xy}$. Furthermore, it derives a session key k and a symmetric key k' from K via $(k, k') \leftarrow \mathsf{PRG}(K)$. It signs the message $g^x|C|\sigma_1|g^y$ to get the signature σ_2. Then it uses k' to encrypt its identity i and σ_2 to obtain $c \leftarrow \mathsf{SE.Enc}(k', i|\sigma_2)$. P_i broadcasts c.

Similarly, P_j can obtain (k, k') from K and decrypt c to get $i|\sigma_2$. By checking the validity of σ_2 with P_i's verification key vk_i, P_j ascertains its partner's identity i and accepts k as the session key.

We refer to Fig. 5 in Section 4 for the details of our PPAKE construction. Below is a high-level analysis of our PPAKE.

- *Robustness.* For the robustness of PPAKE, we require that the underlying KEM is robust in such a sense: if C is generated with pk_j, then decrypting C with any other secret key sk_{j_k} will result in a decryption failure.
- *Explicit mutual authentication.* The authenticity of P_j is guaranteed by KEM and MAC, and the authenticity of P_i is guaranteed by SIG. Hence our PPAKE has explicit mutual authentication.
- *Forward security for session keys.* After excluding active attacks by authenticity, $K = g^{xy}$ is pseudo-random by the DDH assumption. Hence, the session key k, as output of PRG, is pseudo-random as well. Thanks to the ephemeral randomness of x and y, session keys have forward security.
- *Privacy for user identities.* The privacy for user identities relies on KEM and SE. We require that C does not leak information about pk_j computationally, and this is formalized by $\mathsf{IK\text{-}CCA}$ security. As a function output of C, σ_1 does not leak any information either. Meanwhile, g^x and g^y are randomly chosen and independent of i and j. Moreover, ciphertext c protects i and P_i's signature σ_2. Therefore, identity information i, j is well-protected.
- *Forward privacy for user identities.* The forward privacy holds if the initiator P_i is corrupted by $\mathcal{A}$, since the knowledge of the signing key ssk_i does not help $\mathcal{A}$ to learn user's identity in previous runs of PPAKE (recall that the user privacy is guaranteed by KEM and SE). On the other hand, if the responder P_j is corrupted by $\mathcal{A}$, because of the robustness, the knowledge of sk_j can help $\mathcal{A}$ to identify j as long as decrypting C in the previous runs of PPAKE does not result in decryption failure. This suggests that the disclosure of responder's identity j is unavoidable due to the robustness of our PPAKE in the case of responder corruption. However, the initiator's identity i is still well-protected. Therefore, our PPAKE achieves semi-forward privacy when the responder P_j is corrupted and full forward privacy when the initiator P_i is corrupted.
- *Constant communication and computational complexity.* Thanks to the robustness of our PPAKE, the number of broadcast messages per run and the computational complexity per user are constant in our PPAKE, while those in the SSL-PPAKE and RSW-PPAKE schemes are linear to the number μ of users.

Our contribution. We summarize our contribution in this paper. We introduce the concept of robustness into PPAKE, and present a formalized security model for robust PPAKE. In the security model, we consider adversary's passive attacks, active attacks, corruptions of users' long-term keys, and revealing of session keys. Based on the security model, we define user authenticity, forward security for session keys, and forward privacy for user identities.

We propose a generic construction of 3-round robust PPAKE from KEM, SIG, MAC, PRG and SE. By instantiating KEM, MAC, PRG from the DDH assumption and SIG from the CDH assumption (together with a one-time pad SE), we obtain a specific PPAKE scheme in the standard model.

– Our PPAKE scheme enjoys explicit mutual authentication, forward security for session keys and forward privacy for user identities, and resists those attacks on SSL-PPAKE and RSW-PPAKE.
– Our PPAKE scheme is efficient in the sense that both the communication complexity of the protocol and the computational complexity per user is independent of the number of users, thanks to its robustness.

The comparison of our scheme with other PPAKE schemes is shown in Table 1.

Table 1: Comparison among the PPAKE schemes, where μ refers to the number of users. **Comm** denotes the communication complexity of the protocols in terms of the number of group elements. **Comp** denotes the computational complexity per user, where $O(\mu)$ means that **Comp** is linear to μ and $O(1)$ means that **Comp** is independent of μ. **"#"** denotes the number of rounds in the protocol. **Forward Security** is for session keys, where "weak" prevents adversary from modifying the messages sent by the two parties. **Privacy** denotes the privacy of user identity in case of no user corruption. **Forward Privacy** denotes the forward privacy of user identity. **CrpI** denotes forward privacy when initiator is corrupted. **CrpR** denotes forward privacy when responder is corrupted. **I** (**R**) checks whether the privacy of initiator's (responder's) identity is preserved. **Mutual Auth** denotes whether the PPAKE scheme achieves mutual authentication. **Std** denotes whether the security of PPAKE is proved in the standard model.

PPAKE schemes	**Comm**	**Comp**	#	**Forward Security**	**Privacy**		**Forward Privacy**				**Mutual Auth**	**Std**
							CrpI		**CrpR**			
					I	**R**	**I**	**R**	**I**	**R**		
IY[13]	6	$O(1)$	2	weak	✓	×	✓	×	✓	×	×	✓
SKEME[14]	16	$O(1)$	3	✓	✓	✓	×	×	×	×	✓	×
SSL[20]	5μ	$O(\mu)$	4	✓	×	✓	×	✓	×	✓	✓	✓
RSW[19]	$7\mu - 5$	$O(\mu)$	4	✓	✓	✓	✓	✓	×	×	✓	×
Ours	12	$O(1)$	3	✓	✓	✓	✓	✓	✓	×	✓	✓

On Modeling (Forward) Privacy in PPAKE. Our PPAKE works not only for broadcast channel, but also for any public channel, as long as the identifiers like IP or MAC addresses leak no identity information (as considered in [20] and [19]). In these channels, after receiving a message from an initiator, every user may give a response when not aware whether itself is the target recipient.

Some of previous works [21,15,1,2] consider the settings of pre-shared symmetric long-term keys (or passwords) among each pair of users. In this setting, it is easy to achieve authentication, but the assumption is too strong. Most recent work [13] considered a special client-server setting, where client has no long-term key. In this case, the client can be perfectly anonymous but authentication for client is lost.

Our security model, like the security models of SSL-PPAKE [20] and RSW-PPAKE [19], considers that many parties communicate over a public channel. However, We consider a more comprehensive scenario than [20] [19].

Recall that [20] [19] consider the scenario in which the sender and responder in PPAKE are agent servers, and behind each server sits many users. The adversary implements passive and active attacks over the channel between the sender (agent server) and receiver (agent server) but has no access to the channel between the agent server and the end users. The privacy for user identity in [20] [19] essentially said that the adversary cannot tell which user the agent server is delegating during the communications. In our paper, we are considering intact end-to-end user communications rather than limited communications between agent servers. For the sake of privacy protection, messages must not contain user identity explicitly, hence have to be broadcasted to all end users. Each end user may respond the message even if she/he is not the target recipient. Consequently, the initiator may have to deal with a pile of messages from different recipients. Covering end-to-end user communications must consider adversary accessing the channel connecting the end users. Hence, our security model allows adversary's eavesdropping, message insertion/modification/deletion over the broadcast channel which connects end-users. Moreover, as pointed out in [20], their security model only guarantees the privacy of user identities in accepted sessions. Our model also protects user privacy for incomplete sessions and failed sessions.

We stress that our model protects the forward privacy of user identities as much as possible while achieving robustness. To achieve robustness, the first message must be tied to the responder's long term secret key. Once the responder is corrupted, the adversary can identify whether the responder has received messages (but may still do not know the identity of the initiator). Hence, the forward privacy for responder when itself is corrupted is mutually exclusive with the robustness of PPAKE. Consequently, the best forward privacy for robust PPAKE to achieve is semi-forward privacy when the responder is corrupted and full forward privacy when the initiator is corrupted. As shown in Table 1, our PPAKE scheme achieves the best forward privacy as a robust PPAKE, and provides 3 out of 4 kinds of forward privacy, which is the most compared with other PPAKE schemes.

2 Preliminary

Let $\emptyset$ denote an empty string. 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 $x \leftarrow_{\$} \mathcal{X}$ 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 states otherwise. We use $y \leftarrow \mathcal{A}(x)$ to define the random variable y obtained by executing algorithm $\mathcal{A}$ on input x. We use $y \in \mathcal{A}(x)$ to indicate that y lies in the support of $\mathcal{A}(x)$. We also use $y \leftarrow \mathcal{A}(x; r)$ to make explicit the random coins r used in the probabilistic computation. If X and Y have identical distribution, we simply denote it by $X \equiv Y$.

In the full version [18], we review the definition of digital signature and its security notion of strongly existential unforgeability (sEUF-CMA), the definition of message authentication code (MAC) and its security notion of strongly existential unforgeability (sEUF-CMA), the definition of pseudo-random generator (PRG) and its pseudo-randomness, and the definition of ciphertext diversity and semantic security of symmetric encryption (SE).

2.1 Key Encapsulation Mechanism

Definition 1 (KEM). *A key encapsulation mechanism (KEM) scheme* $\mathsf{KEM} = (\mathsf{KEM.Setup}, \mathsf{KEM.Gen}, \mathsf{Encap}, \mathsf{Decap})$ *consists of four algorithms:*

- $\mathsf{KEM.Setup}$: *The setup algorithm outputs public parameters* $\mathsf{pp}_{\mathsf{KEM}}$*, which determines an encapsulation key space* $\mathcal{K}$*, a public key space* $\mathcal{PK}$*, a secret key space* $\mathcal{SK}$*, and a ciphertext space* $\mathcal{CT}$*.*
- $\mathsf{KEM.Gen}$: *Taking* $\mathsf{pp}_{\mathsf{KEM}}$ *as input, the key generation algorithm outputs a pair of public key and secret key* $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$*.*
- $\mathsf{Encap}(pk)$: *Taking* pk *as input, the encapsulation algorithm outputs a pair of ciphertext* $C \in \mathcal{CT}$ *and encapsulated key* $K \in \mathcal{K}$*.*
- $\mathsf{Decap}(sk, C)$: *Taking as input* sk *and* C*, the deterministic decapsulation algorithm outputs* $K \in \mathcal{K} \cup \{\perp\}$*.*

The correctness of KEM *requires that for all* $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{KEM.Setup}, (pk, sk) \in \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$*, and* $(C, K) \in \mathsf{Encap}(pk)$*, it holds that* $\mathsf{Decap}(sk, C) = K$*.*

We recall the IND-CPA and IND-CCA security of KEM.

Definition 2 (IND-CPA/IND-CCA Security for KEM). *For a key encapsulation mechanism* KEM*, the advantage functions of an adversary* $\mathcal{A}$ *are defined by* $\mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{CPA\text{-}0}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{CPA\text{-}1}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right]\right|$ *and* $\mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{CCA\text{-}0}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{CCA\text{-}1}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right]\right|$*, where the experiments* $\mathsf{Exp}^{\mathsf{CPA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$ *and* $\mathsf{Exp}^{\mathsf{CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$ *for* $b \in \{0, 1\}$ *are defined in Figure 2. The IND-CPA/IND-CCA security for* KEM *requires* $\mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{A})/\mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}(\lambda)$ *for all PPT* $\mathcal{A}$*.*

We recall the security notion indistinguishability of keys under chosen-ciphertext attack (IK-CCA Security) formalized by Bellare *et al*. in [5].

Definition 3 (IK-CCA Security for KEM). *For a key encapsulation mechanism* KEM*, the advantage function of an adversary* $\mathcal{A}$ *is defined with* $\mathsf{Adv}^{\mathsf{IK\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{IK\text{-}CCA\text{-}0}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IK\text{-}CCA\text{-}1}}_{\mathsf{KEM},\mathcal{A}} \Rightarrow 1\right]\right|$*, where the experiment* $\mathsf{Exp}^{\mathsf{IK\text{-}CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$ *for* $b \in \{0, 1\}$ *is defined in Figure 3. The IK-CCA security for* KEM *requires that* $\mathsf{Adv}^{\mathsf{IK\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}(\lambda)$ *for all PPT* $\mathcal{A}$*.*

$\mathsf{Exp}^{\mathsf{CPA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$, $\boxed{\mathsf{Exp}^{\mathsf{CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}}$:	$\boxed{\mathcal{O}_{\mathrm{DEC}}(C)}$:
$\mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{KEM.Setup}$; $(pk, sk) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$	If $C = C^*$: Return $\perp$
$(C^*, K_0^*) \leftarrow \mathsf{Encap}(pk)$; $K_1^* \leftarrow \mathcal{K}$	$K \leftarrow \mathsf{Decap}(sk, C)$
$b' \leftarrow \mathcal{A}^{\boxed{\mathcal{O}_{\mathrm{DEC}}(\cdot)}}(pk, C^*, K_b^*)$	Return K
Return b'	

Fig. 2: The IND-CPA security experiment $\mathsf{Exp}^{\mathsf{CPA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$ and the IND-CCA security experiment $\mathsf{Exp}^{\mathsf{CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$ of KEM, where in the latter the adversary can query the decapsulation oracle $\mathcal{O}_{\mathrm{DEC}}(\cdot)$.

$\mathsf{Exp}^{\mathsf{IK\text{-}CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$:	$\mathcal{O}_{sk_0}(C)$:
$\mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{KEM.Setup}$	If $C = C^*$: Return $\perp$
$(pk_0, sk_0) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$	$K \leftarrow \mathsf{Decap}(sk_0, C)$
$(pk_1, sk_1) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$	Return K
$(C^*, K^*) \leftarrow \mathsf{Encap}(pk_b)$	$\mathcal{O}_{sk_1}(C)$:
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot), \mathcal{O}_{sk_1}(\cdot)}(pk_0, pk_1, C^*, K^*)$	If $C = C^*$: Return $\perp$
Return b^*	$K \leftarrow \mathsf{Decap}(sk_1, C)$
	Return K

Fig. 3: The IK-CCA security experiment $\mathsf{Exp}^{\mathsf{IK\text{-}CCA\text{-}b}}_{\mathsf{KEM},\mathcal{A}}$.

Next we introduce the robustness and encapsulated key uniformity of KEM.

Definition 4 (Robustness of KEM). *A key encapsulation mechanism* KEM *has robustness if for all* $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{KEM.Setup}(1^\lambda)$*, it holds that*

$$\Pr\left[\begin{array}{c}(pk_1, sk_1) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); \\ (pk_2, sk_2) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); C_1 \leftarrow \mathsf{Encap}(pk_1)\end{array} : \mathsf{Decap}(sk_2, C_1) \neq \perp\right] = \mathsf{negl}(\lambda).$$

Definition 5 (Encapsulated Key Uniformity of KEM). *A key encapsulation mechanism* KEM *has encapsulated key uniformity if for all* $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{KEM.Setup}(1^\lambda)$*, it holds that*

– $\forall r \in \mathcal{R}$*, it holds that*

$$\{K | r' \leftarrow_\$ \mathcal{R}', (pk, sk) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}; r'), (C, K) \leftarrow \mathsf{Encap}(pk; r)\} \equiv \{K | K \leftarrow_\$ \mathcal{K}\},$$

– $\forall (pk, sk) \in \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$*, it holds that*

$$\{K | r \leftarrow_\$ \mathcal{R}, (C, K) \leftarrow \mathsf{Encap}(pk; r)\} \equiv \{K | K \leftarrow_\$ \mathcal{K}\},$$

where $\mathcal{R}, \mathcal{R}'$ *are the randomness spaces involved in* Encap *and* Gen *respectively.*

Definition 6 (γ-PK-Diversity of KEM). *A key encapsulation mechanism* KEM *has γ-pk-diversity if for all* $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{Setup}(1^\lambda)$*, it holds that*

$$\Pr\left[\begin{array}{c}r \leftarrow_\$ \mathcal{R}; (pk, sk) \leftarrow_\$ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}; r); \\ r' \leftarrow_\$ \mathcal{R}; (pk', sk') \leftarrow_\$ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}; r')\end{array} : pk = pk'\right] = 2^{-\gamma},$$

where $\mathcal{R}$ *is the randomness space involved in* $\mathsf{KEM.Gen}$ *algorithm.*

3 Privacy-Preserving Authenticated Key Exchange

3.1 Definition of Privacy-Preserving Authenticated Key Exchange

Definition 7 (PPAKE). *A privacy-preserving authenticated key exchange scheme* $\mathsf{PPAKE} = (\mathsf{PPAKE.Setup}, \mathsf{PPAKE.Gen}, \mathsf{PPAKE.Protocol})$ *consists of two probabilistic algorithms and an interactive protocol.*

- $\mathsf{PPAKE.Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter* 1^λ*, and outputs the public parameter* $\mathsf{pp}_{\mathsf{PPAKE}}$.
- $\mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, i)$*: The generation algorithm takes as input* $\mathsf{pp}_{\mathsf{PPAKE}}$ *and a party identity* i*, and outputs a key pair* (pk_i, sk_i).
- $\mathsf{PPAKE.Protocol}(P_i(\mathsf{res}_i) \rightleftharpoons P_j(\mathsf{res}_j))$*: The protocol involves two parties* P_i *and* P_j*, who have access to their own resources,* $\mathsf{res}_i := (sk_i, \mathsf{pp}_{\mathsf{PPAKE}}, \{pk_u\}_{u\in[\mu]})$ *and* $\mathsf{res}_j := (sk_j, \mathsf{pp}_{\mathsf{PPAKE}}, \{pk_u\}_{u\in[\mu]})$*, respectively. Here* μ *is the total number of users. After execution,* P_i *outputs a flag* $\Psi_i \in \{\emptyset, \textbf{\textit{accept}}, \textbf{\textit{reject}}\}$*, and a session key* k_i *(*k_i *might be empty string* $\emptyset$*), and* P_j *outputs* (Ψ_j, k_j) *similarly.*

Correctness of PPAKE. For all $\mathsf{pp}_{\mathsf{PPAKE}} \in \mathsf{PPAKE.Setup}(1^\lambda)$, for any distinct and honest parties P_i and P_j with $(pk_i, sk_i) \leftarrow \mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, i)$ and $(pk_j, sk_j) \leftarrow \mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, j)$, after the execution of $\mathsf{PPAKE.Protocol}(P_i(\mathsf{res}_i) \rightleftharpoons P_j(\mathsf{res}_j))$, it holds that $\Psi_i = \Psi_j = \textbf{accept}$ and $k_i = k_j \neq \emptyset$.

Definition 8 (Robustness of PPAKE). *A* PPAKE *scheme is robust if for any party* P_i *who initializes the protocol, then with overwhelming probability, only* P_i*'s intended peer* P_j *is able to determine the validity of the first message sent by* P_i *when following the protocol specifications.*

Remark 1. The correctness and robustness of PPAKE implies the following: in the scenario of honest setting (i.e., all users are honest in the system), if P_i broadcasts the first message and its intended peer is P_j, then only P_j is able to ascertain that the message is for him/her and hence responds to this message.

3.2 Security Model and Security Definitions for PPAKE

We will adapt the security model formalized by [11,4,16], which in turn followed the model proposed by Bellare and Rogaway [6]. We also include replay attacks [17]. In addition, we extend the security model so that the (forward) privacy for user identity is taken into account.

Our security notions for PPAKE include user authenticity, forward security for session key, and forward-privacy for user identity. These are characterized by three security experiments named $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$, $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. In those experiments, we will formalize oracles for adversary $\mathcal{A}$. The passive and active attacks by adversary $\mathcal{A}$ is formalize by its querying to oracles and obtaining answers from oracles. Note that the adversary can copy, delay, erase, replay, and interpolate the messages transmitted over the public channels, obtains some session keys from the PPAKE protocol instances, corrupt some users by obtaining their long-term secret keys, etc.

3.2.1 Oracles

Firstly, we define oracles and their static variables to formalize the behaviour of users and the attacks by the adversary. Suppose there are at most μ users $P_1, P_2, \ldots, P_\mu$, and each user will involve at most ℓ instances. P_i is formalized by a series of oracles, $\pi_i^1, \pi_i^2, ..., \pi_i^\ell$.

Oracle π_i^s. Oracle π_i^s will take a message as input and output a new message, simulating user P_i's execution of s-th PPAKE protocol instance. Each oracle π_i^s has access to P_i's resource $\mathsf{res}_i := (sk_i, \mathsf{pp}_{\mathsf{PPAKE}}, \mathsf{PKList} := \{pk_u\}_{u\in[\mu]})$. π_i^s also has its own variables $\mathsf{var}_i^s := (st_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$.

- st_i^s : State information that has to be stored for π_i^s's next round in the execution of the protocol.
- Pid_i^s : The intended communication peer's identity.
- $k_i^s \in \mathcal{K}$: The session key computed by π_i^s. Here $\mathcal{K}$ is the session key space. We assume that $\emptyset \in \mathcal{K}$.
- $\Psi_i^s \in \{\emptyset, \mathbf{accept}, \mathbf{reject}\}$: Ψ_i^s indicates whether π_i^s has completed the protocol execution and accepted k_i^s.

At the beginning, $(st_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$ are initialized to $(\emptyset, \emptyset, \emptyset, \emptyset)$. We declare that $k_i^s \neq \emptyset$ if and only if $\Psi_i^s = \mathbf{accept}$.

Next, we formalize the oracles that dealing with $\mathcal{A}$'s queries as follows.

Oracle $\mathsf{Send}(i, s, j, \mathsf{MsgList})$. For the query $(i, s, j, \mathsf{MsgList})$, it means that $\mathcal{A}$ invokes π_i^s with $\mathsf{MsgList}$, making π_i^s to play the role of initiator with j being the intended communication peer. Oracle π_i^s will deal with each message in $\mathsf{MsgList}$ to generate new messages $\mathsf{MsgList}'$ according to the protocol specification and update its own variables $\mathsf{var}_i^s = (st_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$. The output messages $\mathsf{MsgList}'$ is returned to $\mathcal{A}$. If $\mathsf{MsgList} = \emptyset$, $\mathcal{A}$ asks oracle π_i^s to send the first round message to j (via broadcast channel).
If $\mathsf{Send}(i, s, j, \mathsf{MsgList})$ is the τ-th query asked by $\mathcal{A}$ and π_i^s changes Ψ_i^s to **accept** after that, then we say that π_i^s is τ-*accepted*.

Oracle $\mathsf{Respond}(\mathsf{OList}, \mathsf{MsgList})$. For the query $(\mathsf{OList}, \mathsf{MsgList})$, it means that $\mathcal{A}$ chooses an oracle set $\mathsf{OList} = \{\pi_j^t\}$ to respond messages in $\mathsf{MsgList}$. For $\forall \pi_j^t \in \mathsf{OList}$, π_j^t executes the PPAKE protocol with messages in $\mathsf{MsgList}$ as a potential recipient, and its variables $\mathsf{var}_j^t = (st_j^t, \mathsf{Pid}_j^t, k_j^t, \Psi_j^t)$ are updated accordingly. Those responding messages generated by OList constitute message set $\mathsf{MsgList}'$. The output message set $\mathsf{MsgList}'$ is returned to $\mathcal{A}$.

Oracle $\mathsf{Corrupt}(i)$. Upon $\mathcal{A}$'s query i, the oracle reveals to $\mathcal{A}$ the long-term secret key sk_i of party P_i. After this corruption, $\pi_i^1, \ldots, \pi_i^\ell$ will stop answering any query from $\mathcal{A}$. If $\mathsf{Corrupt}(i)$ is the τ-th query asked by $\mathcal{A}$, we say that P_i is τ-corrupted. If $\mathcal{A}$ has never asked $\mathsf{Corrupt}(i)$, we say that P_i is ∞-*corrupted*.

Oracle $\mathsf{RegisterCorrupt}(i, pk)$. $\mathcal{A}$'s query (i, pk) suggests that $\mathcal{A}$ registers a new party $P_i (i > \mu)$. The oracle distributes $(i, pk_i := pk)$ to all users. In this case, we say that P_i is 0-*corrupted*.

Oracle $\mathsf{SessionKeyReveal}(i, s)$. The query (i, s) means that $\mathcal{A}$ asks the oracle to reveal π_i^s's session key. If $\Psi_i^s \neq \mathbf{accept}$, the oracle returns $\perp$. Otherwise, the oracle returns the session key k_i^s of π_i^s. If $\mathsf{SessionKeyReveal}(i, s)$ is the

τ-th query asked by $\mathcal{A}$, we say that π_i^s is τ-*revealed*. If $\mathcal{A}$ has never asked $\mathsf{SessionKeyReveal}(i,s)$, we say that π_i^s is ∞-*revealed*.

Oracle $\mathsf{TestKey}(i,s)$**.** The query (i,s) means that $\mathcal{A}$ chooses the session key of π_i^s for challenge (test). If $\Psi_i^s \neq$ **accept**, the oracle returns $\perp$. Otherwise, the oracle sets $k_0 = k_i^s$, samples $k_1 \leftarrow_\$ \mathcal{K}$. The oracle returns k_b to $\mathcal{A}$, where b is the random bit chosen by the challenger.

Oracle $\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$**.** $\mathcal{A}$'s query is the privacy challenge and it consists of two pairs of identities (i_0, j_0) and (i_1, j_1). The oracle builds μ new oracles $\{\pi_u^0\}_{u\in[\mu]}$. Let $\pi_{i_b}^0$ initialize the PPAKE protocol with $\pi_{j_b}^0$ being the intended peer. After the initialization by $\pi_{i_b}^0$, the adversary is allowed to interfere the protocol execution. The transcript of the protocol execution is returned to $\mathcal{A}$, where b is the random bit chosen by the challenger.

3.2.2 Security Experiments of PPAKE

Now we are ready to describe the PPAKE experiments serving for authentication, forward security for session key, and forward privacy for user identity.

Recall that μ is the number of users and ℓ is maximum number of protocol executions per user. The security experiment $\mathsf{Exp}^{\mathsf{X}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$, where $\mathsf{X} \in \{\mathsf{AUTH}, \mathsf{IND}, \mathsf{Privacy}\}$, is played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. $\mathcal{C}$ runs $\mathsf{PPAKE.Setup}$ to get PPAKE public parameter $\mathsf{pp}_{\mathsf{PPAKE}}$.
2. For each party P_i, $\mathcal{C}$ runs $\mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, i)$ to get the long-term key pair (pk_i, sk_i). Next it chooses a random bit $b \leftarrow_\$ \{0,1\}$ and provides $\mathcal{A}$ with the public parameter $\mathsf{pp}_{\mathsf{PPAKE}}$ and the list of public keys $\mathsf{PKList} := \{pk_i\}_{i\in[\mu]}$.
3. $\mathcal{A}$ has access to oracles Send, $\mathsf{Respond}$, $\mathsf{Corrupt}$, $\mathsf{RegisterCorrupt}$, $\mathsf{SessionKeyReveal}$, $\mathsf{TestKey}$, $\mathsf{TestPrivacy}$ by issuing queries in an adaptive way. Note that $\mathcal{A}$ can issue only one query either to $\mathsf{TestKey}$ or to $\mathsf{TestPrivacy}$, but not both. The oracles will reply the corresponding answers to $\mathcal{A}$ as long as the queries lead no trivial attacks.
4. At the end of the experiment, $\mathcal{A}$ terminates with an output b^*.
5. If $b^* = b$, the experiment returns 1; otherwise the experiment returns 0.

$\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$**:** If $\mathcal{A}$ ever queried oracle $\mathsf{TestKey}$ (only once), then $\mathsf{Exp}^{\mathsf{X}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ $= \mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$, which is the experiment for forward security of session key. Through $\mathsf{TestKey}$, adversary $\mathcal{A}$ obtains a real session key k_i^s of target oracle π_i^s or a random key. The forward security of session key requires that it is hard for any PPT $\mathcal{A}$ to distinguish the two cases.

$\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$**:** If $\mathcal{A}$ ever queried oracle $\mathsf{TestPrivacy}$ (only once), then $\mathsf{Exp}^{\mathsf{X}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ $= \mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$, which is the experiment for forward privacy of user identity. Through $\mathsf{TestPrivacy}$, $\mathcal{A}$ obtains a protocol transcript, which is either the interaction of $\pi_{i_0}^0$ and $\pi_{j_0}^0$ or the interaction of $\pi_{i_1}^0$ and $\pi_{j_1}^0$. The forward privacy requires that it is hard for any PPT $\mathcal{A}$ to distinguish the two cases.

$\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$**:** If $\mathcal{C}$ checks whether event $\mathsf{Win}_{\mathsf{Auth}}$ happens ($\mathsf{Win}_{\mathsf{Auth}}$ is defined in Def. 10) at the end of the experiment (either $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ or $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$),

$\underline{\mathsf{Exp}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}}$
$\mathsf{pp}_{\mathsf{PPAKE}} \leftarrow \mathsf{PPAKE.Setup}$
For $i \in [\mu]$:
  $(pk_i, sk_i) \leftarrow \mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, i)$
  $crp_i := \textbf{false}$ //Corruption variable
$\mathsf{PKList} := \{pk_i\}_{i\in[\mu]}; b \leftarrow_{\$} \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s, st_i^s) := (\emptyset, \emptyset, \emptyset, \emptyset)$
  $\mathsf{Aflag}_i^s := \textbf{false}$ //Whether Pid_i^s is corrupted when π_i^s accepts
  $T_i^s := \textbf{false}; kRev_i^s = \textbf{false}$ // Test Key Reveal variables
$T_{key} := \textbf{false}$; $T_{id} := \textbf{false}$ //TestKey, TestPrivacy Oracle variables
$\mathsf{TUsers} := \emptyset$ //Record users queried in TestID Oracle

$\mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp}_{\mathsf{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,RegisterCorrupt SessionKeyReveal,TestKey or TestPrivacy
$\mathsf{Win}_{\mathsf{Auth}} := \textbf{false}$
$\mathsf{Win}_{\mathsf{Auth}} := \textbf{true}$, If $\exists (i,s) \in [\mu] \times [\ell]$ s.t.
(1) $\Psi_i^s = \textbf{accept}$
(2) $\mathsf{Aflag}_i^s = \textbf{false}$
(3) (3.1) $\vee$ (3.2) $\vee$ (3.3). Let $j := \mathsf{Pid}_i^s$
  (3.1) $\nexists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
  (3.2) $\exists t \in [\ell], (j', t') \in [\mu] \times [\ell]$ with $(j,t) \neq (j',t')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_i^s \leftarrow \pi_{j'}^{t'})$
  (3.3) $\exists t \in [\ell], (i', s') \in [\mu] \times [\ell]$ with $(i,s) \neq (i',s')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$
Return $\mathsf{Win}_{\mathsf{Auth}}$

If $T_{key} = \textbf{true} \wedge T_{id} = \textbf{true}$: Return($\perp$)
// Query on TestKey and TestPrivacy are mutually exclusive

$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp}_{\mathsf{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,TestKey SessionKeyReveal RegisterCorrupt
$\mathsf{Win}_{\mathsf{Ind}} := \textbf{false}$
If $b^* = b \ \wedge \ T_{key} = \textbf{true}$:
  $\mathsf{Win}_{\mathsf{Ind}} := \textbf{true}$
Return $\mathsf{Win}_{\mathsf{Ind}}$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp}_{\mathsf{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,TestPrivacy SessionKeyReveal RegisterCorrupt
$\mathsf{Win}_{\mathsf{Privacy}} := \textbf{false}$
$\mathsf{Win}_{\mathsf{Privacy}} := \textbf{true}$, If
(1) $b^* = b \ \wedge \ T_{id} = \textbf{true}$:
(2) Let $\mathsf{TUsers} := (i_0, j_0, i_1, j_1)$,
$(crp_{j_0} = \textbf{false} \wedge crp_{j_1} = \textbf{false}) \vee j_0 = j_1$ //avoid **TA5**
Return $\mathsf{Win}_{\mathsf{Privacy}}$

$\underline{\pi(i, s, j, \mathsf{MsgList}):}$
If $\Psi_i^s = \textbf{reject} \vee \Psi_i^s = \textbf{accept}$: Return $\perp$
$\mathsf{MsgList}' := \emptyset$
If $\mathsf{MsgList} = \emptyset$:
  π_i^s generates the first message msg' for user P_j
  update $(st_i^s, \mathsf{Pid}_i^s, \Psi_i^s, k_i^s)$
  Return $\{\mathsf{msg}'\}$
For each $\mathsf{msg} \in \mathsf{MsgList}$:
  If π_i^s accepts msg:
    π_i^s generates the next message msg' of PPAKE
    $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \{\mathsf{msg}'\}$
    update $(st_i^s, \mathsf{Pid}_i^s, \Psi_i^s, k_i^s)$
Return $\mathsf{MsgList}'$

$\underline{\mathsf{Tran}(i,j):}$ //Return the transcript
Build μ new oracles $\pi_t^0, t \in [\mu]$
$\mathsf{MsgList} := \emptyset$; $\mathsf{Transcript} := \emptyset$; $\mathsf{TfirstMsg} := \emptyset$
While $(\Psi_i^0 = \emptyset \wedge \Psi_j^0 = \emptyset)$ do:
  If $\mathsf{MsgList} = \emptyset$: //The adversary can not insert messages in the first round
    $\mathsf{msg}' \leftarrow \pi(i, 0, j, \emptyset)$
    $\mathsf{MsgList}' := \{\mathsf{msg}'\}$; $\mathsf{TfirstMsg} := \mathsf{msg}'$
  If $\mathsf{MsgList} \neq \emptyset$: //The adversary can insert messages in the non-first round
    $\mathsf{MsgList}' := \emptyset$;
    For $\mathsf{msg} \in \mathsf{MsgList}$
      $\mathsf{msg}' \leftarrow \pi(i, 0, j, \mathsf{msg})$
      $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \{\mathsf{msg}'\}$
    $\mathsf{InsertList} \leftarrow \mathcal{A}(\mathsf{MsgList}, \mathsf{MsgList}')$
    $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \mathsf{InsertList}$
  $\mathsf{Transcript} := \mathsf{Transcript} \cup \mathsf{MsgList}'$
  $\mathsf{MsgList} := \mathsf{MsgList}'$; $\mathsf{MsgList}' := \emptyset$
  For each $j' \in [\mu]$ and each $\mathsf{msg} \in \mathsf{MsgList}$
    $\mathsf{msg}' \leftarrow \pi(j', 0, \emptyset, \mathsf{msg})$
    $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \{\mathsf{msg}'\}$
  If $\neg(\Psi_i^0 = \emptyset \wedge \Psi_j^0 = \emptyset)$:Return $\mathsf{Transcript}$
    $\mathsf{InsertList} \leftarrow \mathcal{A}(\mathsf{MsgList}, \mathsf{MsgList}')$
    $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \mathsf{InsertList}$
    $\mathsf{Transcript} := \mathsf{Transcript} \cup \mathsf{MsgList}'$
    $\mathsf{MsgList} := \mathsf{MsgList}'$
Return $\mathsf{Transcript}$

$\underline{\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t):}$ //Checking whether $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
If $\Psi_i^s \neq \textbf{accept}$: Return 0;
If $\Psi_i^s \neq j$: Return 0;
check wheter the outputs of π_i^s are the inputs of π_j^t
upon the acceptance of π_i^s, and vice verse.
If the transcirpts are consistent: Return 1;
Return 0;

$\underline{\mathcal{O}_{\mathsf{PPAKE}}(\text{query}):}$
If query $= \mathsf{RegisterCorrupt}(u, pk_u)$:
  If $u \in [\mu]$: Return $\perp$
  $\mathsf{PKList} := \mathsf{PKList} \cup \{pk_u\}$
  $crp_u := \textbf{true}$
  Return PKList

If query $= \mathsf{Send}(i, s, j, \mathsf{MsgList})$:
  If $i \notin [\mu] \vee s \notin [\ell] \vee j \notin [\mu]$: Return $\perp$
  If $\mathsf{Pid}_i^s = \emptyset$: $\mathsf{Pid}_i^s = j$
  If $\mathsf{Pid}_i^s \neq j$: Return $\perp$
  $\mathsf{MsgList}' := \emptyset$
  If $\mathsf{MsgList} = \emptyset$:
    $\mathsf{msg}' \leftarrow \pi(i, s, j, \mathsf{msg})$
    Return $\mathsf{MsgList}' = \{\mathsf{msg}'\}$
  For $\mathsf{msg} \in \mathsf{MsgList}$
    $\mathsf{msg}' \leftarrow \pi(i, s, j, \mathsf{msg})$
    $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \{\mathsf{msg}'\}$
  Return $\mathsf{MsgList}'$

If query $= \mathsf{Respond}(\mathsf{OList}, \mathsf{MsgList})$:
  If $T_{id} = \textbf{true} \wedge ((j_0, *) \in \mathsf{OList} \vee (j_1, *) \in \mathsf{OList})$
    $\wedge \mathsf{TfirstMsg} \cap \mathsf{MsgList} \neq \emptyset$:
      Return $\perp$ //avoid **TA6**
  If $\exists (j,t) \in \mathsf{OList} \wedge (j,t) \notin [\mu] \times [\ell]$: Return $\perp$
  $\mathsf{MsgList}' := \emptyset$
  If $crp_j = \textbf{false}$:
    For each $(j,t) \in \mathsf{OList}$, and each $\mathsf{msg} \in \mathsf{MsgList}$:
      $\mathsf{msg}' \leftarrow \pi(j, t', \emptyset, \mathsf{msg})$
      $\mathsf{MsgList}' := \mathsf{MsgList}' \cup \{\mathsf{msg}'\}$
  Return $\mathsf{MsgList}'$

If query $= \mathsf{Corrupt}(i)$:
  If $i \notin [\mu]$: Return $\perp$
  $crp_i := \textbf{true}$
  Return sk_i

If query $= \mathsf{SessionKeyReveal}(i, s)$:
  If $i \notin [\mu] \vee s \notin [\ell]$: Return $\perp$
  If $\Psi_i^s \neq \textbf{accept}$: Return $\perp$
  If $T_i^s = \textbf{true}$: Return $\perp$ //avoid **TA2**
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [l]$ s.t. $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$:
    If $T_j^t = \textbf{true}$: Return $\perp$ //avoid **TA3**
  $kRev_i^s := \textbf{true}$;
  Return k_i^s

If query $= \mathsf{TestKey}(i, s)$:
//This oracle can be only queried once
  $T_{key} := \textbf{true}$
  If $\Psi_i^s \neq \textbf{accept}$:
    Return $\perp$
  If $\mathsf{Aflag}_i^s = \textbf{true} \vee kRev_i^s = \textbf{true}$
    Return $\perp$ //avoid **TA1**, **TA2**
  $T_i^s = \textbf{true}; k_0 := k_i^s; k_1 \leftarrow_{\$} \mathcal{K}$;
  Return k_b

If query $= \mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$:
//This oracle can be only queried once
  $T_{id} := \textbf{true}$
  If $crp_{i_0} \vee crp_{j_0} \vee crp_{i_1} \vee crp_{j_1}$:
    Return $\perp$ //avoid **TA4**
  $\mathsf{TUsers} = (i_0, j_0, i_1, j_1)$
  If $b = 0$: Return $\mathsf{Tran}(i_0, j_0)$
  Else: Return $\mathsf{Tran}(i_1, j_1)$

Fig. 4: The security experiments $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$(with plain text and $\boxed{\text{text}}$), $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$(with plain text and $\boxed{\boxed{\text{text}}}$), $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$(with plain text and $\boxed{\text{text}}$). The list of trivial attacks is given in Table 2.

this experiment is also regarded as $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$, which is the experiment for authenticity. Roughly speaking, the authenticity of PPAKE requires that if an oracle π_i^s accepts a session key, then there must exist a unique oracle π_j^t such that the two oracles have essentially established partnership. Meanwhile, the authenticity makes sure that replay attacks are prevented in the sense that no oracle can make two distinct oracles accepts.

Details of the three experiments are given in Figure 4.

To precisely describe the security notions for PPAKE, we have to forbid some trivial attacks by $\mathcal{A}$. To clearly describe trivial attacks, we first define partner.

Definition 9 (Partner). *We say that an oracle π_i^s is partnered to π_j^t, denoted as $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$, if the following requirements hold:*

- *π_i^s accepts with $\Psi_i^s =$ **accept** and $\mathsf{Pid}_i^s = j$.*
- *Upon the time π_i^s accepts, the transcript of π_i^s is consistent with that of π_j^t, i.e., the outputs of π_i^s are the inputs of π_j^t, and vice verse.*

We write $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$ if $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ and $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$.

We will keep track of the following variables for each party P_i and oracle π_i^s:

- crp_i: whether user i is corrupted.
- Aflag_i^s: whether the intended partner is corrupted when π_i^s accepts.
- $kRev_i^s$: whether the session key k_i^s was revealed.
- T_i^s : whether π_i^s was tested.
- T_{id} : whether oracle $\mathsf{TestPrivacy}$ is queried.
- T_{key} : whether oracle $\mathsf{TestKey}$ is queried.

For forward security for session key, we identify three trivial attacks.

TA1 Suppose that when user i (formalize by π_i^s) accepts a session key k_i^s, its partner j (formalize by π_j^t) has already been corrupted by $\mathcal{A}$, then it is quite possible that $\mathcal{A}$ impersonated j to obtain the shared session key k_i^s. In this case k_i^s cannot be tested by $\mathsf{TestKey}(i, s)$, otherwise, it will be a trivial attack.

TA2 If a session key k_i^s is accepted by user i (formalized by π_i^s) and is also revealed to $\mathcal{A}$, then k_i^s cannot be tested, otherwise, it will be a trivial attack.

TA3 If two users (formalize by oracles π_i^s and π_j^t) are partnered with each other and session key k_i^s of π_i^s is revealed to $\mathcal{A}$, then session key k_j^t of π_j^t cannot be tested due to $k_i^s = k_j^t$. Otherwise, it will be a trivial attack.

For the forward privacy for user identity, we identify three trivial attacks.

TA4 If user i is corrupted, then the adversary is able to impersonate the user in a PPAKE protocol after the corruption. After the protocol execution, the adversary will know the identity of its communicant peer. Hence, this is a trivial attack on privacy of PPAKE when testing i with $\mathsf{TestPrivacy}$.

TA5 The robustness of a PPAKE makes sure that only one target recipient j is able to use its secret key sk_j to correctly respond the first round message. If the secret key sk_j of the target recipient is corrupted by $\mathcal{A}$, no privacy on j is guaranteed. This is a trivial attack on forward privacy of robust PPAKE.

TA6 If the adversary can observe the response of each user after the user receives the first message, then the identity of the responding user is clear to the adversary. Hence, this is also a trivial attack on the privacy of robust PPAKE. This trivial attack can be extended to any core part of the first message. To exclude this trivial attack, if the adversary sees the first round message, it is not allowed to feed a message containing the core part of the first round message to other users and observe their responses.

In Table 2, we list the above trivial attacks **TA1-TA3** in $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ and trivial attacks **TA4-TA6** in $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$.

Types	Trivial attacks	Explanation
TA1	$T_i^s = \textbf{true} \wedge \mathsf{Aflag}_i^s = \textbf{true}$	π_i^s is tested but π_i^s's partner is corrupted when π_i^s accepts session key k_i^s
TA2	$T_i^s = \textbf{true} \wedge kRev_i^s = \textbf{true}$	π_i^s is tested and its session key k_i^s is revealed
TA3	$T_i^s = \textbf{true} \wedge \mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t) \wedge kRev_j^t = \textbf{true}$	π_i^s is tested, π_i^s and π_j^t are partnered to each other, and π_j^t's session key k_j^t is revealed
TA4	$T_{id} = \textbf{true} \wedge (crp_{i_0} = \textbf{true} \vee crp_{j_0} = \textbf{true}$ $\vee crp_{i_1} = \textbf{true} \vee crp_{j_1} = \textbf{true})$	When $\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$ is queried, one of i_0, j_0, i_1, j_1 has been corrupted
TA5	$T_{id} = \textbf{true} \ \wedge \ b^* = b \ \wedge$ $(crp_{j_0} = \textbf{true} \vee crp_{j_1} = \textbf{true}) \wedge j_0 \neq j_1$	$\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$ has been queried, and either j_0 or j_1 has been corrupted when checking $b^* = b$
TA6	$T_{id} = \textbf{true} \ \wedge \mathcal{A}$ queried $\mathsf{Respond}(\mathsf{OList}, \mathsf{MsgList})$ s.t. $((j_0, *) \in \mathsf{OList} \vee (j_1, *) \in \mathsf{OList}) \wedge \mathsf{TfirstMsg} \cap \mathsf{MsgList} \neq \emptyset$	$\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$ is queried, $\mathsf{TfirstMsg}$ is the first message in transcript, $\mathcal{A}$ sees the output $\pi_{j_0}^t(\mathsf{MsgList})$ or $\pi_{j_1}^t(\mathsf{MsgList})$ for some $t \in [\ell]$ via querying $\mathsf{Respond}$ with messages $\mathsf{MsgList}$ containing essential information of $\mathsf{TfirstMsg}$

Table 2: Trivial attacks **TA1-TA3** for security experiment $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. **TA4-TA6** for security experiment $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. Note that $\mathsf{Aflag}_i^s = \textbf{false}$ is implicitly contained in **TA2**, **TA3** because of **TA1**.

3.2.3 Security Notions for PPAKE

Definition 10 (Authentication of PPAKE). *Let* $\mathsf{Win}_{\mathsf{Auth}}$ *denote the event that* $\mathcal{A}$ *breaks authentication in the security experiment* $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ *(see Figure 4).* $\mathsf{Win}_{\mathsf{Auth}}$ *happens iff* $\exists (i, s) \in [\mu] \times [\ell]$, *s.t.*

(1) π_i^s *is* τ*-accepted.*
(2) P_j *is* $\hat{\tau}$*-corrupted with* $j := \mathsf{Pid}_i^s$ *and* $\hat{\tau} > \tau$.
(3) Either (3.1) or (3.2) or (3.3) happens. Let $j := \mathsf{Pid}_i^s$.
  (3.1) There is no oracle π_j^t *that* π_i^s *is partnered to.*
  (3.2) There exist two distinct oracles π_j^t *and* $\pi_{j'}^{t'}$, *to which* π_i^s *is partnered.*
  (3.3) There exist two oracles $\pi_{i'}^{s'}$ *and* π_j^t *with* $(i', s') \neq (i, s)$, *such that both* π_i^s *and* $\pi_{i'}^{s'}$ *are partnered to* π_j^t.

The advantage of an adversary $\mathcal{A}$ in $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ is defined as

$$\mathsf{Adv}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} := \Pr\left[\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \Rightarrow 1\right] = \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge ((3.1) \vee (3.2) \vee (3.3))].$$

Remark 2. Given $(1) \wedge (2)$, (3.1) indicates a successful impersonation of P_i, (3.2) suggests one instance of P_i has multiple partners, and (3.3) corresponds to a successful replay attack. Def.10 captures mutual explicit authentication since π_i^s is either an initiator or a responder.

Definition 11 (Forward Security for Session Key of PPAKE). *In experiment $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ (see Figure 4), Let b^* be $\mathcal{A}$'s output. Then $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \Rightarrow 1$ iff $b^* = b$. The advantage of $\mathcal{A}$ in $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ is defined as*

$$\mathsf{Adv}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} := \left|\Pr\left[\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \Rightarrow 1\right] - 1/2\right|.$$

Forward security for session key asks $\mathsf{Adv}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \leq \mathsf{negl}(\lambda)$ for all PPT $\mathcal{A}$.

Definition 12 (Forward Privacy for User Identity of PPAKE). *Suppose that $\mathcal{A}$ queries $\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$ and b^* is $\mathcal{A}$'s output in $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ (see Figure 4). Define event $\mathsf{Win}_{\mathsf{Privacy}}$ as $b^* = b$ and neither j_0 nor j_1 are corrupted unless $j_0 = j_1$ (i.e. $(crp_{j_0} = \textbf{\textit{false}} \wedge crp_{j_1} = \textbf{\textit{false}}) \vee j_0 \neq j_1$). Then $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \Rightarrow 1$ iff $\mathsf{Win}_{\mathsf{Privacy}}$ happens. Forward privacy for user identity requires that for all PPT $\mathcal{A}$, its advantage function $\mathsf{Adv}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$ satisfies*

$$\mathsf{Adv}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} := \left|\Pr\left[\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} \Rightarrow 1\right] - 1/2\right| \leq \mathsf{negl}(\lambda).$$

Remark 3 (Difference with security models in [20,19]). In the security models in [20,19], the initiator only deals with one responding message with accept or reject and does not take into account other users' responses. This feature excludes the application of their PPAKE schemes in broadcast channels or similar scenarios. In our security model, the initiator receives and processes all messages from other users. This is especially important in the scenario where every user may give a response when not aware whether itself is the target recipient. More precisely, in our security model, the adversarial behaviors are reflected by the formalization that $\mathcal{A}$ designates *a list of messages* for π_i^s to deal with by Send or $\mathsf{Respond}$ queries. In comparison, the security models in [20,19] only consider the case that π_i^s deals with a single message and after that π_i^s will stop responding to other messages (from other users).

Remark 4 (The best forward privacy for robust PPAKE). The best forward privacy for a robust PPAKE scheme is full forward privacy for initiator and semi-forward privacy for responder. The reason is as follows. If the responder P_j is corrupted, the robustness of PPAKE enables the adversary to use the responder's secret key to test the first round messages in previous sessions so as to determine whether P_j is the intended recipient. Therefore, this is the optimal forward privacy for robust PPAKE to achieve: full forward privacy for initiator (no matter initiator or responder is corrupted) and forward privacy for responder when initiator is corrupted.

4 Generic Construction of PPAKE and Its Security Proof

We propose a generic construction of $\mathsf{PPAKE} = (\mathsf{PPAKE.Setup}, \mathsf{PPAKE.Gen}, \mathsf{PPAKE.Protocol})$ with session key space $\mathcal{K}_1$ from the following building blocks.

- A signature scheme $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Ver})$.
- A key encapsulation mechanism scheme $\mathsf{KEM} = (\mathsf{KEM.Setup}, \mathsf{Encap}, \mathsf{Decap})$ with encapsulation key space $\mathcal{K}$.
- A one-time key encapsulation mechanism scheme $\mathsf{otKEM} = (\mathsf{otKEM.Setup}, \mathsf{otEncap}, \mathsf{otDecap})$ with the encapsulation key space $\mathcal{K}'$.
- A message authentication code scheme $\mathsf{MAC} = (\mathsf{MAC.Tag}, \mathsf{MAC.Ver})$ with key space $\mathcal{K}$.
- A symmetric encryption scheme $\mathsf{SE} = (\mathsf{SEnc}, \mathsf{SDec})$ with key space $\mathcal{K}_2$.
- A pseudo-random generator $\mathsf{PRG} : \mathcal{K}' \to \mathcal{K}_1 \times \mathcal{K}_2$.

Our generic construction is given in Figure 5.

$\underline{\mathsf{PPAKE.Setup}}$:	$\underline{\mathsf{PPAKE.Gen}(\mathsf{pp}_{\mathsf{PPAKE}}, i)}$:
$\mathsf{pp}_{\mathsf{SIG}} \leftarrow \mathsf{SIG.Setup}$ $\mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{KEM.Setup}$ $\mathsf{pp}_{\mathsf{otKEM}} \leftarrow \mathsf{otKEM.Setup}$ Return $\mathsf{pp}_{\mathsf{PPAKE}} := (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}}, \mathsf{pp}_{\mathsf{otKEM}})$	$(vk_i, ssk_i) \leftarrow \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$ $(pk_i, sk_i) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$ Return $((vk_i, pk_i), (ssk_i, sk_i))$

$\underline{\mathsf{PPAKE.Protocol}(P_i \rightleftharpoons P_j)}$:

$P_i(\mathsf{res}_i)$		$P_j(\mathsf{res}_j)$
$\mathsf{res}_i := (ssk_i, sk_i, \mathsf{pp}_{\mathsf{PPAKE}}, \{\mathsf{pp}_u\}_{u\in[\mu]} := \{(vk_u, pk_u)\})$		$\mathsf{res}_j := (ssk_j, sk_j, \mathsf{pp}_{\mathsf{PPAKE}}, \{\mathsf{pp}_u\}_{u\in[\mu]} := \{(vk_u, pk_u)\})$
$\Psi_i := \emptyset$, $k_i := \emptyset$, $st_i := \emptyset$ $(C_1, N) \leftarrow \mathsf{Encap}(pk_j)$ $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp}_{\mathsf{otKEM}})$ $st_i := \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$ $\downarrow st_i$	$\xrightarrow{pk_{\mathsf{otKEM}}, C_1}$	$\Psi_j := \emptyset$, $k_j := \emptyset$, $st_j := \emptyset$ If $\mathsf{Decap}(sk_j, C_1) = \bot$: abort $N \leftarrow \mathsf{Decap}(sk_j, C_1)$ $(C_2, K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$ $\sigma_1 \leftarrow \mathsf{MAC.Tag}(N, pk_{\mathsf{otKEM}}\|C_1\|C_2)$ $st_j := \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$ $\downarrow st_j$

If $\Psi_i \neq \emptyset$: Return $\bot$
If $\mathsf{MAC.Ver}(N, pk_{\mathsf{otKEM}}\|C_1\|C_2, \sigma_1) \neq 1$:
  $\Psi_i :=$ **reject**
  Return $\bot$
Else:
  $K \leftarrow \mathsf{otDecap}(sk_{\mathsf{otKEM}}, C_2)$
  $\bar{k}\|k_i \leftarrow \mathsf{PRG}(K)$
  $m := pk_{\mathsf{otKEM}}\|C_1\|C_2\|\sigma_1$
  $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_i, m)$
  $c \leftarrow \mathsf{SEnc}(\bar{k}, i\|\sigma_2)$
  $\Psi_i :=$ **accept**
  Return (Ψ_i, k_i)
	$\xleftarrow{C_2, \sigma_1}$
	
	$\xrightarrow{c}$
		$\bar{k}\|k_j \leftarrow \mathsf{PRG}(K)$
		$(i, \sigma_2) \leftarrow \mathsf{SDec}(\bar{k}, c)$
		$m := pk_{\mathsf{otKEM}}\|C_1\|C_2\|\sigma_1$
		If $\mathsf{SIG.Ver}(vk_i, m, \sigma_2) \neq 1$:
		  $\Psi_j :=$ **reject**
		  Return $\bot$
		Else:
		  $\Psi_j :=$ **accept**
		  Return (Ψ_j, k_j)

Fig. 5: Generic construction of PPAKE

PPAKE.Setup: The setup algorithm generates the public parameter $\mathsf{pp}_{\mathsf{PPAKE}} := (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}}, \mathsf{pp}_{\mathsf{otKEM}})$ by running $\mathsf{SIG.Setup}$, $\mathsf{KEM.Setup}$ and $\mathsf{otKEM.Setup}$.

$\mathsf{PPAKE.Gen}$**:** The key generation algorithm takes as input $\mathsf{pp}_{\mathsf{PPAKE}}$ and a user identity i, and generates a key pair (vk_i, ssk_i) for SIG and a key pair (pk_i, sk_i) for KEM. The public key of user i is (pk_i, vk_i) and the secret key is (ssk_i, sk_i).

$\mathsf{PPAKE.Protocol}(P_i \rightleftharpoons P_j)$**:** The protocol between two parties P_i and P_j is as follows. Each party has access to their own resources $\mathsf{res}_i = (ssk_i, sk_i, \mathsf{pp}_{\mathsf{PPAKE}}, \{\mathsf{pp}_u\}_{u\in[\mu]})$ and $\mathsf{res}_j = (ssk_j, sk_j, \mathsf{pp}_{\mathsf{PPAKE}}, \{\mathsf{pp}_u\}_{u\in[\mu]})$ which contain the corresponding secret key, the public parameter and a list PKList consisting of the public keys of all users. Each party initializes its local variables Ψ_i, k_i and st_i with the empty string. The protocol consists of three rounds of communications.

The First Round: When party P_i initiates a session with party P_j in PPAKE, P_i computes $(C_1, N) \leftarrow \mathsf{Encap}(pk_j)$ and generates an ephemeral key pair $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp}_{\mathsf{otKEM}})$. It then sends $(pk_{\mathsf{otKEM}}, C_1)$ to P_j and stores $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1)$ as its state st_i.

The Second Round: After receiving message $(pk_{\mathsf{otKEM}}, C_1)$, P_j computes $N \leftarrow \mathsf{Decap}(sk_j, C_1)$. If $N = \perp$, then P_j aborts, indicating that it is not the intended recipient of this message. Otherwise, P_j invokes $(C_2, K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$. It uses N as the MAC key to compute a tag $\sigma_1 \leftarrow \mathsf{MAC}(N, pk_{\mathsf{otKEM}}|C_1|C_2)$. Then it sends (C_2, σ_1) to P_i and stores $(pk_{\mathsf{otKEM}}, C_1, C_2, \sigma_1, N, K)$ as its state st_j.

The Third Round: After receiving message (C_2, σ_1), P_i retrieves its state $st_i = (pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1)$. It verifies the validity of σ_1 by checking whether $\mathsf{MAC.Tag}(N, pk_{\mathsf{otKEM}}|C_1|C_2, \sigma_1) = 1$ with the help of N. If invalid, it rejects this message. Otherwise, it continues the protocol by computing $K \leftarrow \mathsf{Decap}(sk_{\mathsf{otKEM}}, C_2)$. It then generates $\bar{k}|k_i \leftarrow \mathsf{PRG}(K)$, where $\bar{k}$ is used as the secret key for SE and k_i as its session key. P_i uses its signing key ssk_i to sign $pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$ and obtain the signature $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_i, pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1)$. Then it encrypts the identity i and the signature σ_2 with $\bar{k}$ and obtains $c \leftarrow \mathsf{SEnc}(\bar{k}, i|\sigma_2)$. It broadcasts the ciphertext c, and sets $\Psi_i = \textbf{accept}$ and outputs (Ψ_i, k_i), indicating its acceptance of k_i as its session key.

After receiving c, P_j retrieves its state $st_j = (pk_{\mathsf{otKEM}}, C_1, C_2, \sigma_1, N, K)$ and generates $(\bar{k}, k_j) \leftarrow \mathsf{PRG}(K)$. It then uses $\bar{k}$ to decrypt the ciphertext c and obtains $(i, \sigma_2) \leftarrow \mathsf{SDec}(\bar{k}, c)$. Next it checks that the validity of (i, σ_2) by checking $\mathsf{SIG.Ver}(vk_i, pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1, \sigma_2) = 1$. P_j rejects in case of invalid. Otherwise, it sets $\Psi_j = \textbf{accept}$ and outputs (Ψ_j, k_j), indicating its acceptance of k_j as its session key with P_i.

Correctness. Correctness of PPAKE follows directly from the correctness of $\mathsf{SIG}, \mathsf{KEM}, \mathsf{otKEM}, \mathsf{MAC}$ and SE.

Robustness. Robustness of PPAKE follows directly from the robustness of KEM, which guarantees that only P_j has $\mathsf{Decap}(sk_j, C_1) \neq \perp$.

Theorem 1. *For the* PPAKE *construction in Figure 5, suppose that the underlying* SIG *is sEUF-CMA secure,* MAC *is sEUF-CMA secure,* KEM *is IND-CCA* *secure and IK-CCA secure,* otKEM *is IND-CPA secure and has the properties of key uniformity and public key diversity, and* PRG *is a pseudo-random generator, and* SE *is semantic secure and has the property of ciphertext diversity, then the* PPAKE *construction has explicit mutual authenticity, forward security and forward privacy.*

G_0, $\boxed{\mathsf{G}_1}$ $\boxed{\mathsf{G}_2}$ $\boxed{\mathsf{G}_3}$

$\mathsf{pp}_{\mathsf{SIG}} \leftarrow \mathsf{SIG.Setup}$
$\mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{KEM.Setup}$
$\mathsf{pp}_{\mathsf{otKEM}} \leftarrow \mathsf{otKEM.Setup}$
$\mathsf{pp}_{\mathsf{PPAKE}} = (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}}, \mathsf{pp}_{\mathsf{otKEM}})$
For $i \in [\mu]$:
  $(pk_i, sk_i) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$
  $(vk_i, ssk_i) \leftarrow \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$
$\mathsf{PKList} := \{(pk_i, vk_i)\}_{i\in[\mu]}$;
$b \leftarrow_\$ \{0,1\}$
$(i^*, s^*) \leftarrow_\$ [\mu] \times [\ell]$
$S := \emptyset$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s, st_i^s) := (\emptyset, \emptyset, \emptyset, \emptyset)$;
  $\mathsf{Aflag}_i^s :=$ **false**
  $(\mathsf{Sent}_i^s, \mathsf{Recv}_i^s) := (\emptyset, \emptyset)$
  $T_i^s :=$ **false**; $kRev_i^s =$ **false**
$\mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp}_{\mathsf{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,RegisterCorrupt
$\mathsf{Win}_{\mathsf{Auth}} :=$ **false** //SessionKeyReveal,TestKey or TestPrivacy
For each $(i,s) \in [\mu] \times [\ell]$:
  If (i,s) s.t.
    (1) $\Psi_i^s =$ **accept**
    (2) $\mathsf{Aflag}_i^s =$ **false**
    (3) (3.1) $\vee$ (3.2) $\vee$ (3.3). Let $j := \mathsf{Pid}_i^s$
      (3.1) $\nexists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
      (3.2) $\exists t \in [\ell], (j', t') \in [\mu] \times [\ell]$ with $(j,t) \neq (j',t')$ s.t.
        $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_i^s \leftarrow \pi_{j'}^{t'})$
      (3.3) $\exists t \in [\ell], (i', s') \in [\mu] \times [\ell]$ with $(i,s) \neq (i',s')$ s.t.
        $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$
  then $S := S \cup \{(i,s)\}$

If $S \neq \emptyset$: $\mathsf{Win}_{\mathsf{Auth}} :=$ **true**
$\boxed{\text{If } (i^*, s^*) \notin S\text{: Return } \perp}$
Return $\mathsf{Win}_{\mathsf{Auth}}$

$\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$: //Checking whether $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
If $\Psi_i^s \neq$ **accept** $\vee\ \mathsf{Pid}_i^s \neq j$: Return 0;
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(pk_{\mathsf{otKEM}}, C_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(C_2, \sigma_1)\}$:
    Return 1
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(C_2, \sigma_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(pk_{\mathsf{otKEM}}, C_1), c\}$:
    Return 1
Return 0

$\pi_i^s(j, \mathsf{MsgList})$:
If $\Psi_i^s =$ **reject** $\vee\ \Psi_i^s =$ **accept**: Return $\perp$
If $\mathsf{MsgList} = \emptyset$: //Generate the first message
  $(C_1, N) \leftarrow \mathsf{Encap}(pk_j)$
  $\boxed{\text{If } (i,s) = (i^*, s^*) : N \leftarrow_\$ \mathcal{K}}$
  $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp}_{\mathsf{otKEM}})$
  $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  $\mathsf{Pid}_i^s := j$
  $\mathsf{Sent}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
  Return $\{pk_{\mathsf{otKEM}}, C_1\}$
If $\mathsf{MsgList} = \{(pk_{\mathsf{otKEM}}, C_1)\}$:
//Receive the first message and generate the second message
  If $st_i^s \neq \emptyset$: Return $\perp$
  $N \leftarrow \mathsf{KEM.Decap}(sk_i, C_1)$
  If $\mathsf{Sent}_{i^*}^{s^*} \neq \emptyset \wedge \mathsf{Pid}_{i^*}^{s^*} = i$:
    Find $C_1^* \in \mathsf{Sent}_{i^*}^{s^*}$ and $N^* \in st_{i^*}^{s^*}$
    If $C_1 = C_1^*$: $N := N^*$
  If $N = \perp$: Return $\perp$; //This message is not sent to user i
  Else:
    $\mathsf{Recv}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
    $(C_2, K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$
    $\sigma_1 \leftarrow \mathsf{MAC.Tag}(N, pk_{\mathsf{otKEM}}|C_1|C_2)$
    $st_i^s := \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
    $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{C_2, \sigma_1\}$
    Return $\{C_2, \sigma_1\}$
For each $\{(C_2, \sigma_1)\} \in \mathsf{MsgList}$:
//Receive the second message and generate the third message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  If $\mathsf{MAC.Ver}(N, pk_{\mathsf{otKEM}}|C_1|C_2, \sigma_1) \neq 1$:
    $\Psi_i^s :=$ **reject**
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{C_2, \sigma_1\}$
  $K \leftarrow \mathsf{otDecap}(pk_{\mathsf{otKEM}}, C_2)$
  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_i, m)$
  $c \leftarrow \mathsf{SEnc}(\bar{k}, i|\sigma_2)$
  $\Psi_i :=$ **accept**
  If $crp_{\mathsf{Pid}_i^s} =$ **true**: $\mathsf{Aflag} :=$ **true**
  $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{c\}$
  Return $\{c\}$
For each $\{c\} \in \mathsf{MsgList}$: //Receive the last message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  $(j, \sigma_2) \leftarrow \mathsf{SDec}(\bar{k}, c)$
  $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  If $\mathsf{SIG.Ver}(vk_j, m, \sigma_2) \neq 1$:
    $\Psi_i^s :=$ **reject**
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{c\}$
  $\Psi_i^s =$ **accept**; $\mathsf{Pid}_i^s = j$
  If $crp_j =$ **true**: $\mathsf{Aflag}_i^s =$ **true**
  Return $\emptyset$

Fig. 6: Games G_0-G_3 for authenticity of PPAKE. Queries to $\mathcal{O}_{\mathsf{PPAKE}} \in \{\mathsf{Send}, \mathsf{Respond}, \mathsf{Corrupt}, \mathsf{RegisterCorrupt}, \mathsf{SessionKeyReveal}, \mathsf{TestPrivacy}, \mathsf{TestKey}\}$ are defined as in the original game in Figure 4 and omitted here.

Before the proof, we will first define two sets Sent_i^s and Recv_i^s for oracle π_i^s. Set Sent_i^s will store outgoing messages of the oracle and Recv_i^s will store valid incoming messages, respectively. We stress that valid messages in Recv_i^s are those incoming messages that pass the verification of MAC or SIG.

We know that $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ holds if the following conditions are satisfied.

– $\mathsf{Pid}_i^s = j$ and $\Psi_i^s = \mathbf{accept}$.
– If π_i^s is the initiator, i.e., π_i^s has sent the first message, then $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(pk_{\mathsf{otKEM}}, C_1)\}$ and $\mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(C_2, \sigma_1)\}$.
– If π_i^s is the responder, i.e., π_i^s has received the first message, then $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(C_2, \sigma_1)\}$, and $\mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(pk_{\mathsf{otKEM}}, C_1), c\}$.

Besides, we define a set S recording all the pairs (i, s) such that $\mathsf{Win}_{\mathsf{Auth}} = \mathbf{true}$.

Proof of explicit mutual authenticity. To prove authenticity for PPAKE, we now describe a sequence of games G_0-G_3 and show that the advantage of $\mathcal{A}$ in adjacent games. The full codes of G_0-G_3 are also given in Figure 6. Define Win_i as the event of $\mathsf{Win}_{\mathsf{Auth}} = \mathbf{true}$ in $\mathsf{G}_i \wedge (i^*, s^*) \in S$, where $(i^*, s^*) \leftarrow_{\$} [\mu] \times [\ell]$.

Game G_0: G_0 is the original experiment $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. In addition, challenger $\mathcal{C}$ uses Sent_i^s and Recv_i^s recording valid incoming valid messages and outgoing messages for π_i^s. This is only a conceptual change. So, $\Pr[(i^*, s^*) \in S \mid \mathsf{Win}_{\mathsf{Auth}} = \mathbf{true}] = \Pr[\mathsf{Win}_0]/\Pr[\mathsf{Win}_{\mathsf{Auth}} = \mathbf{true}] \geq \frac{1}{\mu\ell}$. Then

$$\Pr[\mathsf{Win}_{\mathsf{Auth}} = \mathbf{true}] \leq \mu\ell \cdot \Pr[\mathsf{Win}_0]. \tag{1}$$

Game G_1: In G_1, challenger $\mathcal{C}$ first chooses $(i^*, s^*) \leftarrow_{\$} [\mu] \times [\ell]$. At the end of G_1, if $(i^*, s^*) \notin S$, G_1 aborts by returning $\perp$. Then for the specific pair (i^*, s^*),

$$\Pr[\mathsf{Win}_1] = \Pr[\mathsf{Win}_0] = \Pr_{(i^*,s^*)}[(1) \wedge (2) \wedge (3)]. \tag{2}$$

Game G_2: In G_2, if $\pi_{i^*}^{s^*}$ is a responder, G_2 is the same as G_1. If $\pi_{i^*}^{s^*}$ is an initiator and $\mathsf{Pid}_{i^*}^{s^*} = j^*$, $\mathsf{Sent}_{i^*}^{s^*} \neq \emptyset$, $\mathcal{C}$ changes the behavior of $\pi_{j^*}^t$ for $t \in [\ell]$.

Note $\mathsf{Sent}_{i^*}^{s^*} \neq \emptyset$ implies that $\exists(pk^*_{\mathsf{otKEM}}, C_1^*) \in \mathsf{Sent}_{i^*}^{s^*}$, where $(pk^*_{\mathsf{otKEM}}, sk^*_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp}_{\mathsf{otKEM}})$ and $(C_1^*, N^*) \leftarrow \mathsf{Encap}(pk_{j^*})$. Meanwhile, $\pi_{i^*}^{s^*}$ also has state $st_{i^*}^{s^*} = \{pk^*_{\mathsf{otKEM}}, sk^*_{\mathsf{otKEM}}, N^*, C_1^*\}$. Then for $\forall t \in [\ell]$, if $(pk_{\mathsf{otKEM}}, C_1) \in \mathsf{Recv}_{j^*}^t$, oracle $\pi_{j^*}^t(pk_{\mathsf{otKEM}}, C_1)$ will compute N' by $N \leftarrow \mathsf{Decap}(sk_{j^*}, C_1)$ in G_1. But in G_2, $\pi_{j^*}^t(pk_{\mathsf{otKEM}}, C_1)$ computes N' in the following way.

– $C_1 = C_1^*$: $\pi_{j^*}^t$ borrows N^* from $st_{i^*}^{s^*}$ and sets $N := N^*$.
– $C_1 \neq C_1^*$: $\pi_{j^*}^t$ computes $N \leftarrow \mathsf{Decap}(sk_{j^*}, C_1)$ (as in G_1).

Due to the correctness of KEM, we have

$$\Pr[\mathsf{Win}_2] = \Pr[\mathsf{Win}_1]. \tag{3}$$

Game G_3: In G_3, if $\pi_{i^*}^{s^*}$ is a responder, G_2 is the same as G_1. If $\pi_{i^*}^{s^*}$ is an initiator, then the encapsulation key N^* is randomly chosen with $N^* \leftarrow_{\$} \mathcal{K}$, instead of $N^* \leftarrow \mathsf{Encap}(pk_{j^*})$ as in G_2.

Lemma 1. $|\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_3]| \leq \mu \cdot \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}})$.

The formal proof of Lemma 1 is given in the full version [18]. Here we sketch the proof. We construct adversary $\mathcal{B}_{\mathsf{KEM}}$ against IND-CCA security of KEM scheme.

$\mathcal{B}_{\mathsf{KEM}}$ will simulates $\mathsf{G}_2/\mathsf{G}_3$ for $\mathcal{A}$. $\mathcal{B}_{\mathsf{KEM}}$ gets its challenge (C^*, K^*) w.r.t. pk^*, it sets $pk_{j^*} := pk^*$ with $j^* \leftarrow_\$ [\mu]$, and embeds C^* into $\pi_{i^*}^{s^*}$'s output message $(pk^*_{\mathsf{otKEM}}, C_1^* := C^*)$ and embeds K^* into its state $st_{i^*}^{s^*} := (pk^*_{\mathsf{otKEM}}, sk^*_{\mathsf{otKEM}}, N^* = K^*, C_1^* = C^*)$. $\mathcal{B}_{\mathsf{KEM}}$ also asks its own DECAP oracle $\mathcal{O}_{\mathsf{Decap}}$ to simulate decapsulation of $C_1 \neq C^*$ for oracle $\pi_{j^*}^t(pk_{\mathsf{otKEM}}, C_1)$. Finally, $\mathcal{B}_{\mathsf{KEM}}$ outputs 1 iff Win occurs and $j^* = \mathsf{Pid}_{i^*}^{s^*}$. If K^* is an encapsulated key for C^*, $\mathcal{B}_{\mathsf{KEM}}$ simulates G_2; if K^* is random, $\mathcal{B}_{\mathsf{KEM}}$ simulates G_3. Since $j^* = \mathsf{Pid}_{i^*}^{s^*}$ with probability $1/\mu$, we have $|\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_3]| \leq \mu \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{CCA}}(\mathcal{B}_{\mathsf{KEM}})$.

Next, we analyze (1), (2), (3.1), (3.2), (3.3) in G_3 so as to determine $\Pr[\mathsf{Win}_{\mathsf{Auth}}]$.

We define the event $\mathsf{NoPartner}(i, s)$ as $(1) \wedge (2) \wedge (3.1)$ happens for (i, s). Equivalently, π_i^s accepts, the intended partner $j := \mathsf{Pid}_i^s$ is uncorrupted when π_i^s accepts, and there does not exist $t \in [\ell]$ such that $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$.

Lemma 2. *In G_3, we have* $\Pr_{(i^*,s^*)}[(1) \wedge (2) \wedge (3.1)]$

$$= \Pr[\mathsf{NoPartner}(i^*, s^*)] \leq \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{MAC}}) + \mu \cdot \mathsf{Adv}_{\mathsf{SIG}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{SIG}}).$$

This proof of Lemma 2 relies on the sEUF-CMA security of SIG and MAC.

We consider the probability of event $\mathsf{NoPartner}(i^*, s^*)$ in two cases: $\pi_{i^*}^{s^*}$ is an initiator and $\pi_{i^*}^{s^*}$ is a responder. In the first case, $\pi_{i^*}^{s^*}$ must have received a message (C_2^*, σ_1^*) such that σ_1^* is a valid MAC tag for some non-consistent message $pk^*_{\mathsf{otKEM}}|C_1^*|C_2^*$, yielding a fresh and valid forgery for MAC. In the second case, $\pi_{i^*}^{s^*}$ must have received non-consistent messages $(pk^*_{\mathsf{otKEM}}, C_1^*)$ and c^* whose decryption results in (j^*, σ_2^*), and σ_2^* must be a valid signature for message $pk^*_{\mathsf{otKEM}}|C_1^*|C_2^*|\sigma_1^*$. Due to the ciphertext diversity of SE, $c \neq c^*$ implies that $(j^*, \sigma_2^*) \neq (j', \sigma_2^*)$. If $\mathsf{NoPartner}(i^*, s^*)$ happens, then $(pk^*_{\mathsf{otKEM}}|C_1^*|C_2^*|\sigma_1^*, \sigma_2^*)$ must be a fresh and valid message-signature pair, yielding a successful forgery for SIG. The formal proof is given in the full version [18].

Furthermore, considering the random selection of (i^*, s^*), in G_3 we have

$$\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] \leq \mu\ell \cdot (\mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{MAC}}) + \mu \cdot \mathsf{Adv}_{\mathsf{SIG}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{SIG}})). \quad (4)$$

By Lemma 1 and Eq. (1)(2)(3) and (4), we have the following corollary.

Corollary 1. *In $\mathsf{Exp}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}^{\mathsf{AUTH}}$, it holds that* $\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)]$

$$\leq (\mu\ell) \cdot \big(\mu \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{CCA}}(\mathcal{B}_{\mathsf{KEM}}) + \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{MAC}}) + \mu \cdot \mathsf{Adv}_{\mathsf{SIG}}^{\mathsf{sEUF\text{-}CMA}}(\mathcal{B}_{\mathsf{SIG}})\big).$$

Lemma 3. *In G_3, we have*

$$\Pr_{(i^*,s^*)}[(1) \wedge (2) \wedge (3.2)] \leq (\mu\ell)^2 \cdot (\mathsf{Adv}_{\mathsf{PRG}}^{\mathsf{ps}}(\mathcal{B}_{\mathsf{PRG}}) + \frac{1}{|\mathcal{K}_2|}).$$

If $(1) \wedge (2) \wedge (3.2)$ happens for (i^*, s^*) in G_3, then $\pi_{i^*}^{s^*}$ will accept with session key $k_{i^*}^{s^*}$ and there exist two oracles π_j^t and $\pi_{j'}^{t'}$ subject to $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_j^t)$ and $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_{j'}^{t'})$. Then $\pi_{i^*}^{s^*}$ must share the same session key with both

π_j^t and $\pi_{j'}^{t'}$, which happens with negligible probability, due to the independent randomness in $\pi_{i^*}^{s^*}$, π_j^t and $\pi_{j'}^{t'}$, the key uniformity of otKEM, and the pseudo-randomness of PRG. The formal proof is shown in the full version [18].

By Lemma 3 and Eq. (1)(2)(3), we have the following corollary.

Corollary 2. *In* $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$*,we have*

$$\Pr_{\exists(i,s)}[(1)\wedge(2)\wedge(3.2)] \leq (\mu\ell)^3 \cdot \big(\mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}) + \frac{1}{|\mathcal{K}_2|}\big) + (\mu^2\ell)\cdot \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}}).$$

Lemma 4. *In* $\mathsf{Exp}^{\mathsf{AUTH}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$*, we have*

$$\Pr_{\exists(i,s)}[(1)\wedge(2)\wedge(3.3)] \leq \Pr_{\exists(i,s)}[(1)\wedge(2)\wedge(3.2)] + (\mu\ell)^2\cdot 2^{-\gamma}.$$

Proof. If $\exists(i^*,s^*)$ satisfies $(1)\wedge(2)\wedge(3.3)$, then $\Psi_{i^*}^{s^*} = \mathbf{accept}$, $\mathsf{Aflag}_{i^*}^{s^*} = \mathbf{false}$, $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_j^t)$ and $\mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$. We consider the following two cases.

- **Initiator** $\pi_{i^*}^{s^*}$**.** According to the definition, we know that $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_j^t)$ and $\mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$ implies $(pk^*_{\mathsf{otKEM}}, C_1^*) \in \mathsf{Sent}_{i^*}^{s^*} = \mathsf{Recv}_j^t$, $(pk'_{\mathsf{otKEM}}, C_1') \in \mathsf{Sent}_{i'}^{s'} = \mathsf{Recv}_j^t$, $(C_2^*, \sigma_1^*) \in \mathsf{Recv}_{i^*}^{s^*} = \mathsf{Sent}_j^t$, $(C_2', \sigma_1') \in \mathsf{Recv}_{i'}^{s'} = \mathsf{Sent}_j^t$. Then it holds that $(pk^*_{\mathsf{otKEM}}, C_1^*, C_2^*) = (pk'_{\mathsf{otKEM}}, C_1', C_2')$. According to the γ -pk-diversity of otKEM, we know that $\Pr[pk'_{\mathsf{otKEM}} = pk_{\mathsf{otKEM}}] = 2^{-\gamma}$. Therefore, $(1)\wedge(2)\wedge(3.3)$ happens for (i^*,s^*) and (i',s') with probability at most $2^{-\gamma}$. As there are at most $(\mu\ell)^2$ choices of (i^*,s^*) and (i',s'), we can upper bound the probability of event $(1)\wedge(2)\wedge(3.3)$ by $(\mu\ell)^2\cdot 2^{-\gamma}$ in this case.
- **Responder** $\pi_{i^*}^{s^*}$**.** In this case, $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_j^t)$ implies $\mathsf{Partner}(\pi_j^t \leftarrow \pi_{i^*}^{s^*})$ and $\mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$ implies $\mathsf{Partner}(\pi_j^t \leftarrow \pi_{i'}^{s'})$. This further implies that $(1)\wedge(2)\wedge(3.2)$ happens for (j,t). Therefore, we can upper bound the probability of event $(1)\wedge(2)\wedge(3.3)$ by $(1)\wedge(2)\wedge(3.2)$ in this case.

Combining the above two cases yields Lemma 4. □

Finally, the authenticity of PPAKE follows from Corollary 1,2 and Lemma 4 and

$$\begin{aligned}\Pr[\mathsf{Win}_{\mathsf{Auth}}] \leq& 3\mu^2\ell\cdot\mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}}) + \mu\ell\cdot\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{\mathsf{MAC}}) + (\mu\ell)^2\cdot 2^{-\gamma}\\ &+ 2(\mu\ell)^3\cdot\big(\mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}) + \frac{1}{|\mathcal{K}_2|}\big) + \mu^2\ell\cdot\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{B}_{\mathsf{SIG}}).\end{aligned} \tag{5}$$

Proof of forward security for session key. We now consider another sequence of games $\mathbf{G}_0$-$\mathbf{G}_5$ and analyze $\mathcal{A}$'s advantages in these games. Let Win_i denote the event that $\mathbf{G}_i$ outputs 1, i.e. $\mathcal{A}$'s output bit satisfies $b^* = b$ in G_i. Let $adv_i := |\Pr[\mathsf{Win}_i] - 1/2|$. Then $|adv_i - adv_{i+1}| \leq |\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}]|$. The full codes of $\mathbf{G}_0 - \mathbf{G}_4$ are presented in Figure 7.

Game $\mathbf{G}_0$: $\mathbf{G}_0$ is the original experiment $\mathsf{Exp}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. We add the sets Sent_i^s and Recv_i^s which is only a conceptual change. So,

$$\mathsf{Adv}^{\mathsf{IND}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} := |\Pr[\mathsf{Win}_0] - 1/2| = adv_0. \tag{6}$$

Game $\mathbf{G}_1$: Challenger $\mathcal{C}$ will check whether event $\mathsf{Win}_{\mathsf{Auth}}$ occurs in $\mathbf{G}_1$. If $\mathsf{Win}_{\mathsf{Auth}}$ occurs, $\mathcal{C}$ will abort the game by returning 0. Otherwise, $\mathbf{G}_1$ is the same as $\mathbf{G}_0$. Then $|\Pr[\mathsf{Win}_0] - \Pr[\mathsf{Win}_1]| \leq \Pr[\mathsf{Win}_{\mathsf{Auth}}]$. By (5), we have

$$|adv_0 - adv_1| \leq 3\mu^2\ell \cdot \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}}) + \mu\ell \cdot \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{\mathsf{MAC}}) + (\mu\ell)^2 \cdot 2^{-\gamma}$$
$$+2(\mu\ell)^3 \cdot (\mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}) + \tfrac{1}{|\mathcal{K}_2|}) + \mu^2\ell \cdot \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{B}_{\mathsf{SIG}}). \quad (7)$$

Game $\mathbf{G}_2$: In $\mathbf{G}_2$, if event Hit does not occur, $\mathcal{C}$ will return a random bit $\theta \leftarrow_{\$} \{0,1\}$. Otherwise, $\mathbf{G}_2$ is the same as $\mathbf{G}_1$. Event Hit is defined as follows. Randomly choose $(i^*, s^*, j^*, t^*) \leftarrow_{\$} ([\mu] \times [\ell])^2$. If $\mathcal{A}$ queried $\mathsf{TestKey}(i,s)$ and $\mathsf{TestKey}(i,s)$ did not reply $\perp$, then π_i^s must accept and $\mathsf{Aflag}_i^s = \mathbf{false}$. Accordingly, π_i^s must have a unique partner π_j^t such that $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$. So $\mathsf{TestKey}(i,s)$ uniquely determines a tuple (i,s,j,t). Event Hit occurs if and only if $(i^*, s^*, j^*, t^*) = (i', s', j', t')$. Obviously, $\Pr[\mathsf{Hit}] = 1/(\mu\ell)^2$. We have $\Pr[\mathsf{Win}_2] = \Pr[\mathsf{Hit}] \cdot \Pr[\mathsf{Win}_1] + \Pr[\overline{\mathsf{Hit}}] \cdot \frac{1}{2} = \Pr[\mathsf{Hit}] \cdot (\frac{1}{2} \pm adv_1) + \Pr[\overline{\mathsf{Hit}}] \cdot \frac{1}{2} = \frac{1}{2} \pm \frac{1}{(\mu\ell)^2} \cdot adv_1$. Hence,

$$adv_1 = (\mu\ell)^2 \cdot adv_2. \quad (8)$$

Game $\mathbf{G}_3$: In $\mathbf{G}_3$, the encapsulation key K shared $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is generated by $K \leftarrow_{\$} \mathcal{K}$. Recall that in $\mathbf{G}_2$, $\pi_{i^*}^{s^*}$ computes K with $(C,K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$ while $\pi_{j^*}^{t^*}$ computes K with $K \leftarrow \mathsf{otDecap}(sk_{\mathsf{otKEM}}, C)$.

Lemma 5. $|adv_2 - adv_3| \leq |\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_3]| \leq \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{otKEM}})$.

Recall that in $\mathbf{G}_2$, if $\pi_{i^*}^{s^*}$ accepts session key $k_{i^*}^{s^*}$ and $\mathsf{Aflag}_{i^*}^{s^*} = \mathbf{false}$, then there must exist $\pi_{j^*}^{t^*}$ such that $\mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_{j^*}^{t^*})$. To prove this lemma, we construct an adversary $\mathcal{B}_{\mathsf{otKEM}}$ against the CPA security of otKEM. Given the challenge (C^*, K^*) w.r.t pk^*, $\mathcal{B}_{\mathsf{otKEM}}$ embeds C^* as C_2^* and pk^* as pk^*_{otKEM} in the transcript between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ and sets K^* in the state $st_{i^*}^{s^*}$ or $st_{j^*}^{t^*}$. Finally, $\mathcal{A}$ outputs a guessing bit b^*. If $b^* = b$, $\mathcal{B}_{\mathsf{otKEM}}$ outputs 1; otherwise, $\mathcal{B}_{\mathsf{otKEM}}$ outputs 0.

If K^* is the encapsulated key for C^*, then $\mathcal{B}_{\mathsf{otKEM}}$ perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$; it K^* is random, then $\mathcal{B}_{\mathsf{otKEM}}$ perfectly simulates $\mathbf{G}_3$ for $\mathcal{A}$. Then, we have $|adv_2 - adv_3| \leq |\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_3]| \leq \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{otKEM}})$.

The detailed proof is shown in the full version [18] .

Game $\mathbf{G}_4$: In $\mathbf{G}_4$, the symmetric key and session key of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ are uniformly sampled by $(\bar{k}, k_{i^*}^{s^*} = k_{j^*}^{t^*}) \leftarrow_{\$} \mathcal{K}_1 \times \mathcal{K}_2$. Recall that in $\mathbf{G}_3$, they are generated by $\bar{k}|k_{i^*}^{s^*} \leftarrow \mathsf{PRG}(K)$. Due to the pseudo-randomness of PRG, we have

$$|adv_3 - adv_4| \leq |\Pr[\mathsf{Win}_3] - \Pr[\mathsf{Win}_4]| \leq \mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}). \quad (9)$$

Now that the session key of $\pi_{i^*}^{s^*}$ is randomly chosen with $k_{i^*}^{s^*} \leftarrow_{\$} \mathcal{K}$, we have

$$adv_4 = |\Pr[\mathsf{Win}_4] - 1/2| = 0. \quad (10)$$

Finally, the forward security of PPAKE follows from Lemma 5 and Eq. (6)-(10).

Proof of forward privacy for user identity. To this end, we now consider another sequence of games $\mathbf{G}'_0$-$\mathbf{G}'_7$. Let Win_i denote the event that $\mathsf{Win}_{\mathsf{Privacy}} =$

$\mathbf{G}_0$ $\mathbf{G}_1$, $\mathbf{G}_2$ $\mathbf{G}_3$ $\mathbf{G}_4$

$\mathsf{pp_{SIG}} \leftarrow \mathsf{SIG.Setup}$
$\mathsf{pp_{KEM}} \leftarrow \mathsf{KEM.Setup}$
$\mathsf{pp_{otKEM}} \leftarrow \mathsf{otKEM.Setup}$
$\mathsf{pp_{PPAKE}} = (\mathsf{pp_{SIG}}, \mathsf{pp_{KEM}}, \mathsf{pp_{otKEM}})$
$\boxed{(i^*, s^*, j^*, t^*) \leftarrow_\$ ([\mu] \times [\ell])^2}$
For $i \in [\mu]$:
  $(pk_i, sk_i) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp_{KEM}})$
  $(vk_i, ssk_i) \leftarrow \mathsf{SIG.Gen}(\mathsf{pp_{SIG}})$
$\mathsf{PKList} := \{(pk_i, vk_i)\}_{i \in [\mu]}$;
$b \leftarrow_\$ \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s, st_i^s) := (\emptyset, \emptyset, \emptyset, \emptyset)$;
  $\mathsf{Aflag}_i^s := \mathbf{false}$
  $(\mathsf{Sent}_i^s, \mathsf{Recv}_i^s) := (\emptyset, \emptyset)$
  $T_i^s := \mathbf{false}; kRev_i^s = \mathbf{false}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp_{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,TestKey
$\mathsf{Win_{Ind}} := \mathbf{false}$ //SessionKeyReveal RegisterCorrupt
If $b^* = b$:
  $\mathsf{Win_{Ind}} := \mathbf{true}$

$\mathsf{Win_{Auth}} = \mathbf{true}$, If $\exists (i,s) \in [\mu] \times [\ell]$ s.t.
(1) $\Psi_i^s = \mathbf{accept}$
(2) $\mathsf{Aflag}_i^s = \mathbf{false}$
(3) (3.1) $\vee$ (3.2) $\vee$ (3.3). Let $j := \mathsf{Pid}_i^s$
  (3.1) $\nexists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
  (3.2) $\exists t \in [\ell], (j', t') \in [\mu] \times [\ell]$ with $(j,t) \neq (j',t')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_i^s \leftarrow \pi_{j'}^{t'})$
  (3.3) $\exists t \in [\ell], (i', s') \in [\mu] \times [\ell]$ with $(i,s) \neq (i',s')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$

If $\mathsf{Win_{Auth}} = \mathbf{true}$: Return 0

$\mathsf{Hit} := \mathbf{false}$
If $T_{i^*}^{s^*} = \mathbf{false} \wedge \mathsf{Partner}(\pi_{i^*}^{s^*} \leftarrow \pi_{j^*}^{t^*})$:
  $\mathsf{Hit} := \mathbf{true}$
If $\mathsf{Hit} = \mathbf{false}$:
  $\theta \leftarrow \{0,1\}$
  Return θ

Return $\mathsf{Win_{Ind}}$

$\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$: //Checking whether $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
If $\Psi_i^s \neq \mathbf{accept} \vee \mathsf{Pid}_i^s \neq j$: Return 0;
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(pk_{\mathsf{otKEM}}, C_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(C_2, \sigma_1)\}$:
    Return 1
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(C_2, \sigma_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(pk_{\mathsf{otKEM}}, C_1), c\}$:
    Return 1
Return 0

$\pi_i^s(j, \mathsf{MsgList})$:
If $\Psi_i^s = \mathbf{reject} \vee \Psi_i^s = \mathbf{accept}$: Return $\perp$
If $\mathsf{MsgList} = \emptyset$: //Generate the first message
  $(C_1, N) \leftarrow \mathsf{Encap}(pk_j)$
  $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp_{otKEM}})$
  $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  $\mathsf{Pid}_i^s := j$
  $\mathsf{Sent}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
  Return $\{pk_{\mathsf{otKEM}}, C_1\}$
If $\mathsf{MsgList} = \{(pk_{\mathsf{otKEM}}, C_1)\}$: //Receive the first message and generate the second message
  If $st_i^s \neq \emptyset$: Return $\perp$
  $N \leftarrow \mathsf{KEM.Decap}(sk_i, C_1)$
  If $N = \perp$: Return $\perp$; //This message is not sent to user i
  Else:
    $\mathsf{Recv}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
    $(C_2, K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$
    If $(i,s) = (i^*, s^*)$: $K \leftarrow_\$ \mathcal{K}$
    If $(i,s) = (j^*, t^*)$: $K \leftarrow_\$ \mathcal{K}$
    $\sigma_1 \leftarrow \mathsf{MAC.Tag}(N, pk_{\mathsf{otKEM}}|C_1|C_2)$
    $st_i^s := \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
    $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{C_2, \sigma_1\}$
    Return $\{C_2, \sigma_1\}$
For each $\{(C_2, \sigma_1)\} \in \mathsf{MsgList}$:
//Receive the second message and generate the third message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  If $\mathsf{MAC.Ver}(N, pk_{\mathsf{otKEM}}|C_1|C_2, \sigma_1) \neq 1$:
    $\Psi_i^s := \mathbf{reject}$
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{C_2, \sigma_1\}$
  $K \leftarrow \mathsf{otDecap}(pk_{\mathsf{otKEM}}, C_2)$
  If $(i,s) = (i^*, s^*)$:
    Find K^* in $st_{j^*}^{t^*} = \{pk^*_{\mathsf{otKEM}}, C_1^*, C_2^*, N^*, K^*, \sigma_1^*\}$
    $K := K^*$
  Else if $(i,s) = (j^*, t^*)$:
    Find K^* in $st_{i^*}^{s^*} = \{pk^*_{\mathsf{otKEM}}, C_1^*, C_2^*, N^*, K^*, \sigma_1^*\}$
    $K := K^*$
  Else: $K \leftarrow \mathsf{otDecap}(pk_{\mathsf{otKEM}}, C_2)$
  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  If $(i,s) = (i^*, s^*) \vee (i,s) = (j^*, t^*)$:
    $(\bar{k}, k_i^s) \leftarrow_\$ \mathcal{K} \times \mathcal{K}$
  Else:
    $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_i, m)$
  $c \leftarrow \mathsf{SEnc}(\bar{k}, i|\sigma_2)$
  $\Psi_i := \mathbf{accept}$
  If $crp_{\mathsf{Pid}_i^s} = \mathbf{true}$: $\mathsf{Aflag} := \mathbf{true}$
  $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{c\}$
  Return $\{c\}$
For each $\{c\} \in \mathsf{MsgList}$: //Receive the last message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  If $(i,s) = (i^*, s^*)$:
    $(\bar{k}, k_i^s)$ is the same with $(\bar{k}, k_{j^*}^{t^*})$ used in $\pi_{j^*}^{t^*}$
  Else if $(i,s) = (j^*, t^*)$:
    $(\bar{k}, k_i^s)$ is the same with $(\bar{k}, k_{i^*}^{s^*})$ used in $\pi_{i^*}^{s^*}$
  Else:
    $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  $(j, \sigma_2) \leftarrow \mathsf{SDec}(\bar{k}, c)$
  $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  If $\mathsf{SIG.Ver}(vk_j, m, \sigma_2) \neq 1$:
    $\Psi_i^s := \mathbf{reject}$
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{c\}$
  $\Psi_i^s = \mathbf{accept}$; $\mathsf{Pid}_i^s = j$
  If $crp_j = \mathbf{true}$: $\mathsf{Aflag}_i^s = \mathbf{true}$
  Return $\emptyset$

$\mathcal{O}_{\mathsf{PPAKE}}$(query) :
If query = $\mathsf{TestKey}(i,s)$: //This oracle can be only queried once
  If $\Psi_i^s \neq \mathbf{accept}$:
    Return $\perp$
  If $\mathsf{Aflag}_i^s = \mathbf{true} \vee kRev_i^s = \mathbf{true}$
    Return $\perp$ //avoid **TA1**, **TA2**
  $T_i^s = \mathbf{true}$;
  $k_0 := k_i^s; k_1 \leftarrow_\$ \mathcal{K}$;
  Return k_b

Fig. 7: Games $\mathbf{G}_0$-$\mathbf{G}_4$ for forward security of PPAKE. Queries to $\mathcal{O}_{\mathsf{PPAKE}}$ where query $\in \{\mathsf{Send}, \mathsf{Respond}, \mathsf{Corrupt}, \mathsf{RegisterCorrupt}, \mathsf{SessionKeyReveal}\}$ are defined as in the original game in Figure 4.

true in $\mathbf{G}'_i$. Let $adv_i := |\Pr[\mathsf{Win}_i] - 1/2|$. Then $|adv_i - adv_{i+1}| := |\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}]|$. The full codes of $\mathbf{G}'_0$-$\mathbf{G}'_7$ are presented in Figure 8.

Game $\mathbf{G}_0'$: $\mathbf{G}_0'$ is the original experiment $\mathsf{Exp}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}}$. We also add the sets Sent_i^s and Recv_i^s which is only a conceptual change. So,

$$\mathsf{Adv}^{\mathsf{Privacy}}_{\mathsf{PPAKE},\mu,\ell,\mathcal{A}} := |\Pr[\mathsf{Win}_{\mathsf{Privacy}}] - 1/2| = adv_0 \tag{11}$$

Game $\mathbf{G}_1'$: At the end of $\mathbf{G}_1'$, challenger $\mathcal{C}$ will check whether event $\mathsf{Win}_{\mathsf{Auth}}$ occurs. If $\mathsf{Win}_{\mathsf{Auth}}$ occurs, $\mathcal{C}$ will abort the game by returning 0. Otherwise, $\mathbf{G}_1'$ is the same as $\mathbf{G}_0'$. Due to the difference lemma and (5), we have

$$\begin{aligned}|adv_0 - adv_1| \le\ & 3\mu^2\ell \cdot \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}}) + \mu\ell \cdot \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{\mathsf{MAC}}) + (\mu\ell)^2 2^{-\gamma} \\ & + \mu^2\ell \cdot \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{B}_{\mathsf{SIG}}) + 2(\mu\ell)^3 \cdot \left(\mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}) + \tfrac{1}{|\mathcal{K}_2|}\right).\end{aligned} \tag{12}$$

Game $\mathbf{G}_2'$: In $\mathbf{G}_2'$, upon $\mathcal{A}$'s query to oracle $\mathsf{Tran}(i,j)$, π_i^0 and π_j^0 will not respond to any message in $\mathsf{InsertList}$ sent by $\mathcal{A}$. Note that each oracle responds to only one valid message. If this valid message is not sent by $\mathcal{A}$, then $\mathbf{G}_2'$ is the same as $\mathbf{G}_1'$. If this valid message is sent by $\mathcal{A}$ (the message can only be inserted in the second round or third round of our protocol), then this will lead to occurrence of event $\mathsf{NoPartner}(i,0)$, which is impossible. Hence, $\mathbf{G}_2'$ is identical to $\mathbf{G}_1'$, and

$$adv_1 = adv_2. \tag{13}$$

Now we define an event named Hit. When $\mathcal{A}$ queries $\mathsf{TestPrivacy}(i,j,i',j')$, a unique tuple (i,j,i',j') is determined. Even Hit happens iff $(i_0^*, j_0^*, i_1^*, j_1^*) = (i,j,i',j')$, where $(i_0^*, j_0^*, i_1^*, j_1^*) \leftarrow_{\$} [\mu]^4$ is sample at the beginning the game. Note that $(i_0^*, j_0^*, i_1^*, j_1^*)$ follows a uniform distribution, so we have $\Pr[\mathsf{Hit}] = \frac{1}{\mu^4}$.
Game $\mathbf{G}_3'$: At the end of $\mathbf{G}_3'$, if event Hit does not occur, $\mathcal{C}$ will return a random bit $\theta \leftarrow_{\$} \{0,1\}$ instead of detecting event Win. Otherwise, $\mathbf{G}_3'$ is the same as $\mathbf{G}_2'$. We have $\Pr[\mathsf{Win}_3] = \Pr[\mathsf{Hit}] \cdot \Pr[\mathsf{Win}_2] + \Pr\left[\boxed{\mathsf{Hit}}\right] \cdot \frac{1}{2} = \Pr[\mathsf{Hit}] \cdot (\frac{1}{2} \pm adv_2) + \Pr\left[\boxed{\mathsf{Hit}}\right] \cdot \frac{1}{2} = \frac{1}{2} \pm \frac{1}{\mu^4} \cdot adv_2$. As a result,

$$adv_2 = \mu^4 \cdot adv_3. \tag{14}$$

Game $\mathbf{G}_4'$: In $\mathbf{G}_4'$, the encapsulation key K shared by $\pi^0_{i_b^*}$ and $\pi^0_{j_b^*}$ is generated by $K \leftarrow_{\$} \mathcal{K}$, instead of $(C,K) \leftarrow \mathsf{otEncap}(pk)$ and $K \leftarrow \mathsf{otDecap}(C)$ as in $\mathbf{G}_3'$. Similar to the proof of Lemma 5, we have

$$|adv_3 - adv_4| \le \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{otKEM}}). \tag{15}$$

Game $\mathbf{G}_5'$: In $\mathbf{G}_5'$, the symmetric key and session key of $\pi^0_{i_b^*}$ and $\pi^0_{j_b^*}$ are generated by $(\bar{k}, k^0_{i_b^*}) = (\bar{k}, k^0_{j_b^*}) \leftarrow_{\$} \mathcal{K}_1 \times \mathcal{K}_2$ instead of $\mathsf{PRG}(K)$ as in $\mathbf{G}_4'$. Hence,

$$|adv_4 - adv_5| \le \mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRG}}(\mathcal{B}_{\mathsf{PRG}}). \tag{16}$$

Game $\mathbf{G}_6'$: In $\mathbf{G}_6'$, If $j_0 = j_1$, then $\mathbf{G}_6'$ is the same as $\mathbf{G}_5'$. Otherwise, $\pi^0_{i_b^*}$ generates C_1^* by $(C_1^*, N) \leftarrow \mathsf{Encap}(pk_{j_1^*})$, instead of $(C_1^*, N) \leftarrow \mathsf{Encap}(pk_{j_b^*})$ as in $\mathbf{G}_5'$. By IK-CCA security of KEM, we know that (C_1^*, N) w.r.t $pk_{j_0^*}$ is indistinguishable to that w.r.t $pk_{j_1^*}$. So we have Lemma 6 with proof shown in the full version [18].

$\mathbf{G}_0'$, $\mathbf{G}_1'$, $\mathbf{G}_2'$, $\mathbf{G}_3'$, $\mathbf{G}_4'$, $\mathbf{G}_5'$ $\mathbf{G}_6'$ $\boxed{\mathbf{G}_7'}$

$\mathsf{pp_{SIG}} \leftarrow \mathsf{SIG.Setup}$
$\mathsf{pp_{KEM}} \leftarrow \mathsf{KEM.Setup}$
$\mathsf{pp_{otKEM}} \leftarrow \mathsf{otKEM.Setup}$
$\mathsf{pp_{PPAKE}} = (\mathsf{pp_{SIG}}, \mathsf{pp_{KEM}}, \mathsf{pp_{otKEM}})$
$\boxed{(i_0^*, j_0^*, i_1^*, j_1^*) \leftarrow_\$ [\mu]^4}$
For $i \in [\mu]$:
  $(pk_i, sk_i) \leftarrow \mathsf{KEM.Gen}(\mathsf{pp_{KEM}})$
  $(vk_i, ssk_i) \leftarrow \mathsf{SIG.Gen}(\mathsf{pp_{SIG}})$
$\mathsf{PKList} := \{(pk_i, vk_i)\}_{i\in[\mu]}$;
$b \leftarrow_\$ \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s, st_i^s) := (\emptyset, \emptyset, \emptyset, \emptyset)$;
  $\mathsf{Aflag}_i^s := \mathbf{false}$
  $(\mathsf{Sent}_i^s, \mathsf{Recv}_i^s) := (\emptyset, \emptyset)$
  $T_i^s := \mathbf{false}; kRev_i^s = \mathbf{false}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PPAKE}}(\cdot)}(\mathsf{pp_{PPAKE}}, \mathsf{PKList})$ //$\mathcal{O}_{\mathsf{PPAKE}}$ = Send,Respond,Corrupt,TestPrivacy
$\mathsf{Win_{Privacy}} := \mathbf{false}$ //SessionKeyReveal RegisterCorrupt
$\mathsf{Win_{Privacy}} := \mathbf{true}$, If
(1) $b^* = b$
(2) Let $\mathsf{TUsers} = (i_0, j_0, i_1, j_1)$,
$crp_{j_0} = \mathbf{false} \wedge crp_{j_1} = \mathbf{false}$

$\mathsf{Win_{Auth}} = \mathbf{true}$, If $\exists (i,s) \in [\mu] \times [\ell]$ s.t.
(1) $\Psi_i^s = \mathbf{accept}$
(2) $\mathsf{Aflag}_i^s = \mathbf{false}$
(3) (3.1) $\vee$ (3.2) $\vee$ (3.3). Let $j := \mathsf{Pid}_i^s$
  (3.1) $\nexists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
  (3.2) $\exists t \in [\ell], (j', t') \in [\mu] \times [\ell]$ with $(j,t) \neq (j',t')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_i^s \leftarrow \pi_{j'}^{t'})$
  (3.3) $\exists t \in [\ell], (i', s') \in [\mu] \times [\ell]$ with $(i,s) \neq (i',s')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \cap \mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$

If $\mathsf{Win_{Auth}} = \mathbf{true}$: Return 0

If $(i_0^*, j_0^*, i_1^*, j_1^*) = (i_0, j_0, i_1, j_1)$: $\mathsf{Hit} := \mathbf{true}$
Else: $\mathsf{Hit} := \mathbf{false}$
If $\mathsf{Hit} = \mathbf{false}$:
  $\theta \leftarrow \{0,1\}$; Return θ

Return $\mathsf{Win_{Privacy}}$

$\underline{\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t):}$ //Checking whether $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
If $\Psi_i^s \neq \mathbf{accept} \vee \mathsf{Pid}_i^s \neq j$: Return 0;
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(pk_{\mathsf{otKEM}}, C_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(C_2, \sigma_1)\}$:
    Return 1
If π_i^s sent the first message:
  If $\mathsf{Sent}_i^s = \mathsf{Recv}_j^t = \{(C_2, \sigma_1)\} \wedge \mathsf{Recv}_i^s = \mathsf{Sent}_j^t = \{(pk_{\mathsf{otKEM}}, C_1), c\}$:
    Return 1
Return 0

$\underline{\mathsf{Tran}(i,j):}$ //Return the transcript
Build μ new oracles $\pi_t^0, t \in [\mu]$
$\mathsf{MsgList} := \emptyset$
$\mathsf{Transcript} := \emptyset$
While $(\Psi_i^0 \neq \mathbf{accept} \vee \Psi_j^0 \neq \mathbf{accept})$ do:
  $\mathsf{RecvList} \leftarrow \mathsf{Send}(i, 0, j, \mathsf{MsgList} \setminus \mathsf{InsertList})$
  If $\mathsf{MsgList} \neq \emptyset$: //The adversary can not insert messages in the first round
    $\mathsf{InsertList} \leftarrow \mathcal{A}(\mathsf{RecvList})$
  $\boxed{\mathsf{InsertList} := \emptyset}$
  $\mathsf{RecvList} := \mathsf{RecvList} \cup \mathsf{InsertList}$
  $\mathsf{Transcript} := \mathsf{Transcript} \cup \mathsf{RecvList}$
  If $(\Psi_i^0 = \mathbf{accept} \wedge \Psi_j^0 = \mathbf{accept})$:
    Return $\mathsf{Transcript}$
  $\mathsf{MsgList} \leftarrow \mathsf{Respond}(\{\pi_t^0\}_{t\in[\mu]}, \mathsf{RecvList})$
  $\mathsf{InsertList} \leftarrow \mathcal{A}(\mathsf{RecvList})$
  $\boxed{\mathsf{InsertList} := \emptyset}$
  $\mathsf{MsgList} := \mathsf{MsgList} \cup \mathsf{InsertList}$
  $\mathsf{Transcript} := \mathsf{Transcript} \cup \mathsf{MsgList}$
Return $\mathsf{Transcript}$

$\underline{\pi_i^s(j, \mathsf{MsgList}):}$
If $\Psi_i^s = \mathbf{reject} \vee \Psi_i^s = \mathbf{accept}$: Return $\perp$
If $\mathsf{MsgList} = \emptyset$: //Generate the first message
  $(C_1, N) \leftarrow \mathsf{Encap}(pk_j)$
  If $(i,s) = (i_b^*, 0)$:
    $(C_1, N) \leftarrow \mathsf{Encap}(pk_{j_1^*})$
  $(pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}) \leftarrow \mathsf{otKEM.Gen}(\mathsf{pp_{otKEM}})$
  $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  $\mathsf{Pid}_i^s := j$
  $\mathsf{Sent}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
  Return $\{pk_{\mathsf{otKEM}}, C_1\}$
If $\mathsf{MsgList} = \{(pk_{\mathsf{otKEM}}, C_1)\}$: //Receive the first message and generate the second message
  If $st_i^s \neq \emptyset$: Return $\perp$
  $N \leftarrow \mathsf{KEM.Decap}(sk_i, C_1)$
  If $(i,s) = (j_b^*, 0)$:
    Find $N^* \in st_{i_b^*}^0 = \{pk_{\mathsf{otKEM}}^*, sk_{\mathsf{otKEM}}^*, N^*, C_1^*\}$
    $N := N^*$
  If $N = \perp$: Return $\perp$; //This message is not sent to user i
  Else:
    $\mathsf{Recv}_i^s := \{pk_{\mathsf{otKEM}}, C_1\}$
    $(C_2, K) \leftarrow \mathsf{otEncap}(pk_{\mathsf{otKEM}})$
    If $(i,s) = (j_b^*, 0) : K \leftarrow_\$ \mathcal{K}$

    $\sigma_1 \leftarrow \mathsf{MAC.Tag}(N, pk_{\mathsf{otKEM}}|C_1|C_2)$
    $st_i^s := \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
    $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{C_2, \sigma_1\}$
    Return $\{C_2, \sigma_1\}$
For each $\{(C_2, \sigma_1)\} \in \mathsf{MsgList}$:
//Receive the second message and generate the third message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, sk_{\mathsf{otKEM}}, N, C_1\}$
  If $\mathsf{MAC.Ver}(N, pk_{\mathsf{otKEM}}|C_1|C_2, \sigma_1) \neq 1$:
    $\Psi_i^s := \mathbf{reject}$
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{C_2, \sigma_1\}$
  $K \leftarrow \mathsf{otDecap}(pk_{\mathsf{otKEM}}, C_2)$
  If $(i,s) = (i_b^*, 0)$:
    Find K^* in $st_{j_b^*}^0 = \{pk_{\mathsf{otKEM}}^*, C_1^*, C_2^*, N^*, K^*\}$
    $K := K^*$

  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  If $(i,s) = (i_b^*, 0)$:
    $(\bar{k}, k_i^s) \leftarrow_\$ \mathcal{K} \times \mathcal{K}$
  $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_i, m)$
  $c \leftarrow \mathsf{SEnc}(\bar{k}, i|\sigma_2)$
  If $(i,s) = (i_b^*, 0)$:
    $\sigma_2 \leftarrow \mathsf{SIG.Sign}(ssk_{i_1^*}, m)$
    $c \leftarrow \mathsf{SEnc}(\bar{k}, i_1^*|\sigma_2)$
  $\Psi_i := \mathbf{accept}$
  If $crp_{\mathsf{Pid}_i^s} = \mathbf{true}$: $\mathsf{Aflag} := \mathbf{true}$
  $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{c\}$
  Return $\{c\}$
For each $\{c\} \in \mathsf{MsgList}$: //Receive the last message
  If st_i^s is not the form of $\{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$:
    Return $\perp$
  parse $st_i^s = \{pk_{\mathsf{otKEM}}, C_1, C_2, N, K, \sigma_1\}$
  $\bar{k}|k_i^s \leftarrow \mathsf{PRG}(K)$
  If $(i,s) = (j_b^*, 0)$:
    $(\bar{k}, k_i^s)$ is the same with $(\bar{k}, k_{i_b^*}^0)$ used in $\pi_{i_b^*}^0$
  $(j, \sigma_2) \leftarrow \mathsf{SDec}(\bar{k}, c)$; $m := pk_{\mathsf{otKEM}}|C_1|C_2|\sigma_1$
  If $\mathsf{SIG.Ver}(vk_j, m, \sigma_2) \neq 1$:
    $\Psi_i^s := \mathbf{reject}$
    Return $\perp$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{c\}$
  $\Psi_i^s = \mathbf{accept}$; $\mathsf{Pid}_i^s = j$
  If $crp_j = \mathbf{true}$: $\mathsf{Aflag}_i^s = \mathbf{true}$
  Return $\emptyset$

$\underline{\mathcal{O}_{\mathsf{PPAKE}}(\text{query}):}$
If query = $\mathsf{TestPrivacy}(i_0, j_0, i_1, j_1)$: //This oracle can be only queried once
  If $crp_{i_0} \vee crp_{j_0} \vee crp_{i_1} \vee crp_{j_1}$:
    Return $\perp$
  $\mathsf{TUsers} = (i_0, j_0, i_1, j_1)$
  If $b = 0$:
    Return $\mathsf{Tran}(i_0, j_0)$
  Else:
    Return $\mathsf{Tran}(i_1, j_1)$

Fig. 8: Games $\mathbf{G}_0'$-$\mathbf{G}_7'$ for forward privacy of PPAKE. Queries to $\mathcal{O}_{\mathsf{PPAKE}}$ where query $\in \{\mathsf{Send}, \mathsf{Respond}, \mathsf{Corrupt}, \mathsf{RegisterCorrupt}, \mathsf{SessionKeyReveal}\}$ are defined as in the original game in Figure 4.

Lemma 6. $|adv_5 - adv_6| \leq |\Pr[\mathsf{Win}_5] - \Pr[\mathsf{Win}_6]| \leq \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IK\text{-}CCA}}(\mathcal{B}_{\mathsf{KEM}})$.

Game $\mathbf{G}_7'$: $\mathbf{G}_7'$ is almost the same as $\mathbf{G}_6'$, except for the answer generation of oracle $\mathsf{TestPrivacy}(i, j, i', j')$ (which is $\mathsf{TestPrivacy}(i_0^*, j_0^*, i_1^*, j_1^*)$). In $\mathbf{G}_7'$, c^* is an encryption of (i_1^*, σ_2^*) where σ_2^* is computed using the signing key $ssk_{i_1^*}$. However, in $\mathbf{G}_6'$, c^* is an encryption of (i_b^*, σ_2^*) with σ_2^* a signature generated by the signing key $ssk_{i_b^*}$. The semantic security of SE makes sure that this change is indistinguishable, as shown in Lemma 7.

Lemma 7. $|adv_6 - adv_7| \leq |\Pr[\mathsf{Win}_6] - \Pr[\mathsf{Win}_7]| \leq \mathsf{Adv}_{\mathsf{SE}}^{\mathsf{Sem}}(\mathcal{B}_{\mathsf{SE}})$.

The formal proof is given in the full version [18].

Finally, in $\mathbf{G}_7'$, all the messages in $\mathsf{Transcript} = \{(pk_{\mathsf{otKEM}}^*, C_1^*), (C_2^*, \sigma_1^*), c^*\}$ are independent of b, so we have

$$adv_7 = |\Pr[\mathsf{Win}_7] - 1/2| = 0. \tag{17}$$

Finally, the forward privacy of PPAKE follows from Lemma 6,7 and (11)-(17).

5 Instantiations of PPAKE

In this section, we present concrete instantiations for the building blocks of our PPAKE including KEM, otKEM, SIG, MAC, PRG and SE. This yields a specific PPAKE scheme based on the DDH assumption over a cyclic group $\mathbb{G}$ and the CDH assumption over a bilinear group in the standard model. The details of the instantiations are shown in the full version [18].

KEM**.** We employ the Cramer-Shoup KEM (CS-KEM) scheme over a cyclic group $\mathbb{G}$ of order q. It is well known that CS-KEM is IND-CCA secure. Its public parameter is $(\mathbb{G}, q, g_1, g_2)$. Now we show its robustness. Given a ciphertext $C = (u_1, u_2, v) \in \mathbb{G}^3$ under public key $pk = (c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^{z_1} g_2^{z_2}) \in \mathbb{G}^3$, we know that $u_1 = g^r, u_2 = g^r$ and $v = c^r d^{\alpha r} = u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$, where α is the hash value of (u_1, u_2). When decrypting C with another independent and random secret key $(x_1', x_2', y_1', y_2', z_1', z_2')$, we have that $\Pr\left[v = u_1^{x_1'+\alpha y_1'} u_2^{x_2'+\alpha y_2'}\right]$ with probability $2/q$. Therefore, C will be rejected except with probability $2/q$.

otKEM**.** We employ the Elgamal-KEM scheme over a cyclic group $\mathbb{G}$ of order q. It is well known that Elgamal-KEM is IND-CPA secure. The public key is given by $pk = g^x \in \mathbb{G}$ and the ciphertext is $C = g^y \in \mathbb{G}$ and the encapsulated key is $K = g^{xy}$. The encapsulated key $K = g^{xy}$ is uniformly distributed, when either the secret key $sk = x$ or the randomness y used in $\mathsf{otKEM.Encap}$ is independently and randomly chosen over $\mathbb{Z}_q$. Hence, ElGamal-KEM has encapsulated key uniformity. Meanwhile, when $x, x' \leftarrow_{\$} \mathbb{Z}_q$, two public keys $pk = g^x = g^{x'} = pk'$ collide, i.e., $pk = g^x = g^{x'} = pk'$ with probability $1/q$. Hence it has $\log q$-pk-diversity.

SIG. We employ the BSW signature scheme [7] over a bilinear group with bilinear map $e : \mathbb{G}' \times \mathbb{G}' \to \mathbb{G}_1$. Its sEUF-CMA security is based on the CDH assumption over $\mathbb{G}'$. Its signature space is $\Sigma = {\mathbb{G}'}^2 \times \mathbb{Z}_q$.

MAC. We use the MAC scheme [9] over a cyclic group $\mathbb{G}$ of order q. Its sEUF-CMA security is based on the DDH assumption over $\mathbb{G}$. The MAC key is $(\omega, x, x') \in \mathbb{Z}_q^3$ and the tag for message m is given by $\sigma = (u, v_1, v_2) \in \mathbb{G}^3$, where u is uniformly chosen, $v_1 = u^\omega$ and $v_2 = u^{x\ell + x'}$ with ℓ the hash value of (u, v_1, m). Its tag space is $\mathbb{G}^3$.

PRG. We use the PRG scheme [10], where $\mathsf{PRG} : \mathbb{Z}_q \to \mathbb{Z}_q^5$. The PRG scheme is based on the DDH assumption over a cyclic group of order q.

SE. We can use one time pad over $\mathbb{Z}_q$ as our SE scheme, which has information-theoretical semantic security. The secret key space, the plain text space and the cipher text space is $\mathcal{K} = \mathcal{M} = \mathcal{C} = \mathbb{Z}_q$ with q a prime.

Assembling the above schemes according to our generic construction, we have a specific PPAKE scheme, with communication complexity $(\mathbb{G}+3\mathbb{G})+(\mathbb{G}+3\mathbb{G})+(2\mathbb{G}'+2\mathbb{Z}_q) = 8\mathbb{G} + 2\mathbb{G}' + 2\mathbb{Z}_q$. The security of the PPAKE scheme is based on the DDH assumption over $\mathbb{G}$ and the CDH assumption over the bilinear group $\mathbb{G}'$. The detail of the scheme is shown in the full version [18].

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments. Shengli Liu and You Lyu were partially supported by National Natural Science Foundation of China (NSFC No. 61925207) and Guangdong Major Project of Basic and Applied Basic Research (2019B030302008). Shuai Han was partially supported by National Natural Science Foundation of China (Grant No. 62002223), Shanghai Sailing Program (20YF1421100), and Young Elite Scientists Sponsorship Program by China Association for Science and Technology (YESS20200185).

References

1. Abdalla, M., Izabachène, M., Pointcheval, D.: Anonymous and transparent gateway-based password-authenticated key exchange. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) Cryptology and Network Security, 7th International Conference, CANS 2008, Hong-Kong, China, December 2-4, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5339, pp. 133–148. Springer (2008). https://doi.org/10.1007/978-3-540-89641-8_10
2. Alwen, J., Hirt, M., Maurer, U., Patra, A., Raykov, P.: Anonymous authentication with shared secrets. In: Aranha, D.F., Menezes, A. (eds.) Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8895, pp. 219–236. Springer (2014). https://doi.org/10.1007/978-3-319-16295-9_12
3. Arfaoui, G., Bultel, X., Fouque, P., Nedelcu, A., Onete, C.: The privacy of the TLS 1.3 protocol. Proc. Priv. Enhancing Technol. **2019**(4), 190–210 (2019). https://doi.org/10.2478/popets-2019-0065

4. Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9014, pp. 629–658. Springer (2015). https://doi.org/10.1007/978-3-662-46494-6_26
5. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2248, pp. 566–582. Springer (2001). https://doi.org/10.1007/3-540-45682-1_33
6. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer (1993). https://doi.org/10.1007/3-540-48329-2_21
7. Boneh, D., Shen, E., Waters, B.: Strongly unforgeable signatures based on computational diffie-hellman. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3958, pp. 229–240. Springer (2006). https://doi.org/10.1007/11745853_15
8. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: Blaze, M. (ed.) Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA. pp. 303–320. USENIX (2004), http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html
9. Dodis, Y., Kiltz, E., Pietrzak, K., Wichs, D.: Message authentication, revisited. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 355–374. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_22
10. Farashahi, R.R., Schoenmakers, B., Sidorenko, A.: Efficient pseudorandom generators based on the DDH assumption. In: Okamoto, T., Wang, X. (eds.) Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4450, pp. 426–441. Springer (2007). https://doi.org/10.1007/978-3-540-71677-8_28
11. Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 95–125. Springer (2018). https://doi.org/10.1007/978-3-319-96881-0_4
12. Heinrich, A., Hollick, M., Schneider, T., Stute, M., Weinert, C.: Privatedrop: Practical privacy-preserving authentication for apple airdrop. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 3577–3594. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich

13. Ishibashi, R., Yoneyama, K.: Post-quantum anonymous one-sided authenticated key exchange without random oracles. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13178, pp. 35–65. Springer (2022). https://doi.org/10.1007/978-3-030-97131-1_2
14. Krawczyk, H.: SKEME: a versatile secure key exchange mechanism for internet. In: Ellis, J.T., Neuman, B.C., Balenson, D.M. (eds.) 1996 Symposium on Network and Distributed System Security, (S)NDSS '96, San Diego, CA, USA, February 22-23, 1996. pp. 114–127. IEEE Computer Society (1996). https://doi.org/10.1109/NDSS.1996.492418
15. Lee, M., Smart, N.P., Warinschi, B., Watson, G.J.: Anonymity guarantees of the UMTS/LTE authentication and connection protocol. Int. J. Inf. Sec. **13**(6), 513–527 (2014). https://doi.org/10.1007/s10207-014-0231-3
16. Li, Y., Schäge, S.: No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1343–1360. ACM (2017). https://doi.org/10.1145/3133956.3134006
17. Liu, X., Liu, S., Gu, D., Weng, J.: Two-pass authenticated key exchange with explicit authentication and tight security. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12492, pp. 785–814. Springer (2020). https://doi.org/10.1007/978-3-030-64834-3_27
18. Lyu, Y., Liu, S., Han, S., Gu, D.: Privacy-preserving authenticated key exchange in the standard model. Cryptology ePrint Archive, Paper 2022/1217 (2022), https://eprint.iacr.org/2022/1217
19. Ramacher, S., Slamanig, D., Weninger, A.: Privacy-preserving authenticated key exchange: Stronger privacy and generic constructions. In: Bertino, E., Shulman, H., Waidner, M. (eds.) Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12973, pp. 676–696. Springer (2021). https://doi.org/10.1007/978-3-030-88428-4_33
20. Schäge, S., Schwenk, J., Lauer, S.: Privacy-preserving authenticated key exchange and the case of ikev2. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 567–596. Springer (2020). https://doi.org/10.1007/978-3-030-45388-6_20
21. Yang, X., Jiang, H., Hou, M., Zheng, Z., Xu, Q., Choo, K.R.: A provably-secure two-factor authenticated key exchange protocol with stronger anonymity. In: Au, M.H., Yiu, S., Li, J., Luo, X., Wang, C., Castiglione, A., Kluczniak, K. (eds.) Network and System Security - 12th International Conference, NSS 2018, Hong Kong, China, August 27-29, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11058, pp. 111–124. Springer (2018). https://doi.org/10.1007/978-3-030-02744-5_8