Mild Asymmetric Message Franking: Illegal-Messages-Only and Retrospective Content Moderation

Zhengan Huang[1], Junzuo Lai[2], Gongxian Zeng[1], and Jian Weng[2]

[1] Pengcheng Laboratory, Shenzhen, China
zhahuang.sjtu@gmail.com, gxzeng@cs.hku.hk
[2] College of Information Science and Technology,
Jinan University, Guangzhou, China
{laijunzuo, cryptjweng}@gmail.com

Abstract. Many messaging platforms have integrated end-to-end (E2E) encryption into their services. This widespread adoption of E2E encryption has triggered a technical tension between user privacy and illegal content moderation. The existing solutions either support only unframeability or deniability, or they are prone to abuse (the moderator can perform content moderation for all messages, whether illegal or not), or they lack mechanisms for retrospective content moderation.
To address the above issues, we introduce a new primitive called *mild asymmetric message franking* (MAMF) to establish illegal-messages-only and retrospective content moderation for messaging systems, supporting unframeability and deniability simultaneously. We provide a framework to construct MAMF, leveraging two new building blocks, which might be of independent interest.

Keywords: Asymmetric message franking; Set pre-constrained encryption; Hash proof system; Sigma protocol; Deniability and unframeability

1 Introduction

In recent years, there has been a substantial surge in the adoption of messaging applications deploying end-to-end (E2E) encryption, e.g., Facebook Messenger, WhatsApp and Signal, ensuring that the transmitted raw information cannot be obtained by the platforms.

Despite the security advantages, the widespread adoption of E2E encryption has not been universally welcomed. These encryption services might be misused for disseminating harmful content such as harassment messages, phishing links, fake news, and other potentially illegal information. Moreover, these services conflict with content moderation directly. Law enforcement and national security communities contend that such encryption hampers their ability to investigate, prosecute criminals and ensure public safety. In fact, the conflict between privacy

and content moderation has spurred legislative proposals and policy campaigns about discouraging the deployment of E2E encryption [FBI,Tar18].

On the other hand, technical experts have voiced concerns that these proposals, if implemented, might compromise the security provided by encryption systems [AAB+15], either by requiring unsafe alterations or prohibiting the use of E2E encryption altogether.

In recent years, many works have focused on employing cryptographic techniques to strike a balance between ensuring user privacy and effectively moderating illegal content within messaging systems.

One approach to uphold content moderation is through the use of message franking (MF) [Fac16a,Fac16b,GLR17,TGL+19], seamlessly integrating with end-to-end (E2E) encryption, as discussed in [TGL+19]. This method empowers the receiver of a message to *report* it to a moderator (also referred to as the judge). In the MF framework, a valid report by the receiver includes the sender's identity, along with the message and specially constructed strings (e.g., signatures or hash values). These elements allow the moderator to verify whether the reported message originated from the identified sender. In order to strengthen user privacy, asymmetric message franking (AMF) [TGL+19] captures deniability. Informally, this property allows the sender to technically deny sending the message after a compromise, aiming to avoid potential backlash or embarrassment. AMF considers different scenarios regarding whose secret key is compromised. One scenario is judge compromise deniability, where a forger with the moderator's secret key can produce a signature indistinguishable from a real one. Unfortunately, existing works [TGL+19,IAV22,LZH+23] *do not support unframeability*. Specifically, it becomes challenging for the moderator to convince law enforcement of the identity of the content originator, because judge compromise deniability allows the moderator to forge a message in the name of the sender. More importantly, current MF solutions do not adequately address a significant concern: the moderator can identify the sender of *all* reported messages, *regardless of the message's intent*. This extensive power of the moderator has the potential for abuse within the existing system.

Recently, a different proposal has emerged, suggesting end-to-end secure messaging with content moderation *exclusively for some pre-defined illegal contents*, based on set pre-constrained (SPC) group signatures [BGJP23]. However, as demonstrated in [BGJP23], their techniques are "tailored to obtain the strongest notion of unframeability and *no deniability*". Moreover, it remains unclear how to execute content moderation for newly identified illegal content that hasn't been predefined. This feature is reminiscent of "retrospective" access to encrypted data as considered in [GKVL21], which is in a somewhat different context and relies on extractable witness encryption [GKP+13].

In this paper, we aim to establish *mild* content moderation for a messaging system.

Firstly, we restrict the moderation capabilities to *illegal messages only*, while concurrently providing the capacity for *retrospective* content moderation. Notably, the set of illegal messages may increase over time. To carry out content

moderation for the newly added illegal messages, a straightforward approach might involve system re-initialization; however, such a method impedes the moderator's ability to retrospectively moderate content. One objective of this paper is to enable the moderator to retrospectively examine past reports and identify those that qualify as illegal when new illegal messages are augmented.

Secondly, our content moderation system achieves trade-offs between deniability and unframeability. In [BGJP23], Bartusek et al. discuss the technical tension between deniability and unframeability, and claim that, while deniability [TGL+19] is a desirable property, it conflicts with unframeability. Upon meticulous examination, we discern that the primary reason why [TGL+19] cannot achieve unframeability is due to the fact that the scheme [TGL+19] supports judge-receiver compromise deniability. To elaborate, when the receiver's and the moderator's keys are compromised, a party possessing these keys can forge a signature that can be successfully accepted by both the receiver and the moderator. Our approach to achieve deniability and unframeability simultaneously imposes an additional constraint on deniability. Roughly, given any forged signature (for deniability), the receiver and the moderator can not accept it simultaneously. It then carves out space for unframeability, indicating that when both the receiver and the moderator identify the sender concurrently, the sender cannot deny sending the message.

1.1 Main contributions

Our main contributions can be summarized as follows:

1. We introduce a new primitive called *mild asymmetric message franking* (MAMF) to establish mild content moderation for a messaging system, and formalize its security notions.
2. To construct MAMF, we introduce two new building blocks, *universal set pre-constrained encryption* (USPCE) and *dual hash proof system-based key encapsulation mechanism supporting Sigma protocols* (dual HPS-KEM$^{\Sigma}$), and present their concrete constructions.
3. We offer a generic framework of constructing MAMF from USPCE and dual HPS-KEM$^{\Sigma}$, and demonstrate that it fulfills the required security properties. By integrating a concrete USPCE scheme and a dual HPS-KEM$^{\Sigma}$ into the generic framework, we can obtain a concrete MAMF. We also have some improvements discussed in Appendix H, enhancing the efficiency of the concrete MAMF.

MAMF primitive. In the context of MAMF, four types of participants are involved: the sender, the receiver, the legislative agency, and the moderator (also referred to as the judge). MAMF comprises eleven algorithms: a setup algorithm Setup for generating global public parameters, three algorithms ($\mathsf{KG}_{\mathsf{Ag}}$, KG_{J}, KG_{u}) for generating key pairs, three algorithms (Frank, Verify, Judge) for creating and verifying genuine signatures, a token generation algorithm TKGen for retrospective content moderation, and three forging algorithms (Forge, RForge, JForge) for deniability.

We offer further explanations here.

Upon receiving the public parameter generated by Setup, the legislative agency selects a set S (representing illegal messages) and uses $\mathsf{KG}_{\mathsf{Ag}}$ to generate a key pair for itself and an auxiliary parameter for the moderator. The moderator, leveraging the auxiliary parameter, invokes KG_{J} to create a key pair. The sender and receiver both utilize KG_{u} to generate their private/public keys.

The sender employs the franking algorithm Frank to generate a designated-verifier signature for a message m. The receiver utilizes Verify (with its secret key as input) to validate the received signature. If the received message is deemed illegal, and the receiver reports it to the moderator, the moderator can confirm the report using algorithm Judge, determining that the sender indeed sent the message. When the legislative agency intends to augment the set S with an additional illegal message for retrospective content moderation, it invokes the TKGen algorithm to produce a token for the new illegal message. With the aid of this token, the moderator can retrospectively examine past reports (as well as new reports). It's important to note that algorithms Forge, RForge, and JForge are not intended for execution by legitimate users. Their presence ensures deniability under specific compromise scenarios.

We address six distinct security requirements for MAMF: unforgeability, accountability, unframeability, deniability, untraceability, and confidentiality of sets.

1. *Unforgeability.* As the fundamental security prerequisite for general signatures, unforgeability in MAMF ensures prevention of successful impersonation, i.e., the receiver cannot be deceived into accepting a message not genuinely sent by the sender.
2. *Accountability.* Accountability ensures that the functionality of reporting illegal messages. In line with the definition in [TGL+19,LZH+23], accountability is formalized with two special properties: sender binding and receiver binding. Sender binding ensures that the sender cannot trick the receiver into accepting unreportable messages, and receiver binding ensures that the receiver cannot deceive the judge to frame an innocent sender.
3. *Deniability.* Deniability is formalized with three special properties: universal deniability, receiver compromise deniability, and judge compromise deniability. Universal deniability guarantees deniability when neither the receiver's secret key nor the judge's secret key is compromised. Receiver compromise deniability guarantees deniability when the receiver's secret key is compromised. Judge compromise deniability is formalized to guarantee deniability when the judge's secret key is compromised.
4. *Unframeability.* Unframeability of MAMF requires that no party, even given a receiver's secret key and the judge's secret key, is able to produce a signature acceptable to both the receiver and the judge. This property implies that once both the receiver and the judge identify the originator of some illegal message, they can generate an evidence (e.g., a NIZK proof) to convince the other party of the originator of the message.

5. *Untraceability.* Ensuring untraceability restricts the capabilities of both the legislative agency and the judge, thereby enhancing sender privacy. This concept formalizes into two distinct notions: untraceability against legislative agency and untraceability against judge. Untraceability against legislative agency guarantees that the agency cannot determine if someone has actually sent a message, no matter whether it is in the set of illegal message or not. Untraceability against judge ensures that, without the assistance of the legislative agency, the moderator cannot ascertain the sender's identity when the message is not in the set of illegal messages.
6. *Confidentiality of Sets.* Confidentiality of sets requires that the legislative agency's public key and the judge's public key will not disclose any information about the set of illegal messages (which should not be disclosed to the public, e.g., child sexual abuse material).

Analogous to AMF [TGL+19], MAMF can be integrated with E2E encryption, which guarantees the confidentiality of messages. So we do not consider confidentiality of messages for MAMF. Furthermore, our MAMF could be extended to accommodate group communications like [LZH+23]. We leave it as a future work.

Technical overview. For MAMF construction, we introduce two new primitives, USPCE and dual HPS-KEM$^{\Sigma}$, and utilize them to show a framework of constructing MAMF. We provide a technical overview here.

USPCE. In [BGJP23], Bartusek et al. formulate set pre-constrained encryption (SPCE). Generally, SPCE requires the generation of a public/secret key pair for a predefined (illegal message) set S. In SPCE, decryption of a ciphertext, produced by encrypting a message with the public key and an item x, is only possible when $x \in \mathsf{S}$. If $x \notin \mathsf{S}$, the secret key holder gains no information about the message.

Note that SPCE is insufficient for constructing MAMF, primarily due to its inability to handle ciphertexts produced by encrypting messages with respect to items where $x \notin \mathsf{S}$, while in the MAMF framework, in order to carry retrospective content moderation, the moderator should be able to handle the messages not in the set S, as long as the legislative agency has provided the corresponding tokens. Henceforth, we introduce a primitive, called universal set pre-constrained encryption (USPCE), to address these challenges.

A USPCE comprises five key algorithms: $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{TKGen}, \mathsf{Dec})$, where two kinds of entities, the authority and users, are involved. The setup algorithm Setup, executed by the authority, takes the security parameter and a pre-defined set S as input, and outputs public parameters, an auxiliary parameter, and a master secret key. The users invoke KG with the public and auxiliary parameters to generate their key pairs. The encryption algorithm Enc takes a public key, an item x, and a message m as input, producing a ciphertext.

– If the item x belongs to the set S, the user can directly employ the decryption algorithm Dec (with their secret key as input) to output the message m.

– If $x \notin \mathsf{S}$, the authority can execute the token generation algorithm TKGen (with the master secret key as input) to create a token tk for the item x. Subsequently, the user can utilize the decryption algorithm Dec, taking their secret key, the ciphertext and the token tk as input, to recover the message m.

It is required that there is a Sigma protocol to prove that the ciphertext is well-formed.

For USPCE, we require the following security properties.

– *Confidentiality against authority:* It is required that the authority cannot obtain meaningful information about the message from a ciphertext, no matter whether the item x belongs to the set S or not.
– *Confidentiality against users:* It is required that, without the token for an item $x \notin \mathsf{S}$ given by the authority, any user cannot obtain meaningful information about the message from a ciphertext associated with x.
– *Confidentiality of sets:* It is required that the public parameters and a user's public key will not disclose any information about the pre-defined set S.

A concrete construction of USPCE based on the DBDH assumption is provided in Sec. 4.

Dual HPS-KEM$^{\Sigma}$. We introduce another building block, called dual hash proof system-based key encapsulation mechanism supporting Sigma protocols (dual HPS-KEM$^{\Sigma}$), which roughly can be seen as a dual version of the HPS-KEM$^{\Sigma}$ proposed in [LZH+23].

In essence, in a dual HPS-KEM$^{\Sigma}$, ciphertexts are generated in accordance with the original HPS-KEM$^{\Sigma}$ approach, while encapsulated keys are created in two modes: one follows the original HPS-KEM$^{\Sigma}$ method, and the other adopts an extended version of HPS-KEM$^{\Sigma}$ where an additional tag t is included as input during the computation of the encapsulated key. Moreover, in dual HPS-KEM$^{\Sigma}$, two additional algorithms are required for the uniform sampling of encapsulated keys: one with a tag as input and the other without using a tag.

Expanding on this, a dual HPS-KEM$^{\Sigma}$ scheme consists of ten algorithms: $\mathsf{Setup}, \mathsf{KG}, \mathsf{Encap}_{\mathrm{c}}, \mathsf{Encap}_{\mathrm{k}}, \mathsf{Decap}, \mathsf{Encap}_{\mathrm{c}}^{*}, \mathsf{dEncap}_{\mathrm{k}}, \mathsf{dDecap}, \mathsf{SamEncK}$ and $\mathsf{dSamEncK}$.

We start by concentrating on the first six algorithms, which comprise an ordinary HPS-KEM$^{\Sigma}$ scheme. Specifically, Setup generates the public parameter, and KG produces a pair of public/secret user keys. Given the public parameter, but without user's public key, $\mathsf{Encap}_{\mathrm{c}}$ outputs a well-formed ciphertext, and $\mathsf{Encap}_{\mathrm{c}}^{*}$ outputs a ciphertext that could be either well-formed or ill-formed. The algorithm $\mathsf{Encap}_{\mathrm{k}}$, sharing the same randomness space with $\mathsf{Encap}_{\mathrm{c}}$, takes the public parameter and a public key as input, and outputs an encapsulated key. Utilizing the secret key, the algorithm Decap decapsulates the ciphertexts to obtain the encapsulated keys. Correctness requires that given a ciphertext output by $\mathsf{Encap}_{\mathrm{c}}$ with randomness r, Decap will return an encapsulated key equal to that created by $\mathsf{Encap}_{\mathrm{k}}$ with the same randomness r.

The following properties inherited from HPS-KEM$^{\Sigma}$ are required:

1. *Universality:* Given a public key, it is difficult for any unbounded adversary without the corresponding secret key to generate an ill-formed ciphertext c, an encapsulated key k, and randomness r_{c}^* (indicating that c is generated via $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^*), such that with the ciphertext c as input, Decap outputs a key equal to k.
2. *Ciphertext unexplainability:* It is difficult to generate a ciphertext c and randomness r_{c}^* (indicating that c is generated via $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^*), such that c is well-formed.
3. *Indistinguishability:* The ciphertext output by $\mathsf{Encap}_{\mathrm{c}}^*$ should be indistinguishable from the well-formed ciphertext output by $\mathsf{Encap}_{\mathrm{c}}$.
4. *SK-second-preimage resistance:* Given a pair of public/secret keys, it is difficult to generate another valid secret key for this public key.
5. *Smoothness:* For any fixed public key, the algorithm Decap, fed with a ciphertext generated via $\mathsf{Encap}_{\mathrm{c}}^*$ and a secret key randomly sampled from the set of secret keys corresponding to the public key, will output a key uniformly distributed over the encapsulated key space.

Now, let's shift our focus to the last four algorithms of dual HPS-KEM$^{\Sigma}$, i.e., $\mathsf{dEncap}_{\mathrm{k}}, \mathsf{dDecap}, \mathsf{SamEncK}$ and $\mathsf{dSamEncK}$.

The algorithm $\mathsf{dEncap}_{\mathrm{k}}$, sharing the same randomness space and the same encapsulated key space with $\mathsf{Encap}_{\mathrm{k}}$, takes the public parameter, a public key and a tag as input, and outputs an encapsulated key. Utilizing the secret key and the tag, the algorithm dDecap decapsulates the ciphertexts to obtain the encapsulated keys. Correctness requires that given a tag t and a ciphertext output by $\mathsf{Encap}_{\mathrm{c}}$ with randomness r, dDecap will return an encapsulated key equal to that generated by $\mathsf{dEncap}_{\mathrm{k}}$ using the same tag t and randomness r.

The algorithms $\mathsf{SamEncK}$ and $\mathsf{dSamEncK}$ are both used to uniformly sample encapsulated keys. In particular, $\mathsf{SamEncK}$ takes the public parameter as input, and outputs an encapsulated key, while $\mathsf{dSamEncK}$ takes both the public parameter and a tag as input, and outputs an encapsulated key.

The following properties are also required for dual HPS-KEM$^{\Sigma}$:

6. *Extended universality:* Given a public key, it is difficult for any unbounded adversary without the corresponding secret key to generate an ill-formed ciphertext c, an encapsulated key k, a tag t, and randomness r_{c}^* (indicating that c is generated via $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^*), such that with the ciphertext c and the tag t as input, dDecap outputs a key equal to k.
7. *Key unexplainability:* Given a pair of public/secret keys, it is difficult to generate $(c, r_{\mathsf{c}}^*, k, r_{\mathsf{k}}^*)$ (where c is a ciphertext generated via $\mathsf{Encap}_{\mathrm{c}}^*$ using randomness r_{c}^*, and k is an encapsulated key generated via $\mathsf{SamEncK}$ using randomness r_{k}^*), such that k is the result of decapsulating c by Decap.
8. *Extended key unexplainability:* Given a pair of public/secret keys, it is difficult to generate $(c, r_{\mathsf{c}}^*, k, t, r_{\mathsf{k}}^*)$ (where c is a ciphertext generated via $\mathsf{Encap}_{\mathrm{c}}^*$ using randomness r_{c}^*, and k is an encapsulated key generated via $\mathsf{dSamEncK}$ using tag t and randomness r_{k}^*), such that k is the result of decapsulating c by dDecap using tag t.

9. *Extended smoothness:* For any fixed public key, the algorithm dDecap, fed with a ciphertext generated via $\mathsf{Encap}_{\mathsf{c}}^{*}$, a random tag, and a secret key randomly sampled from the set of secret keys corresponding to the public key, will output a key uniformly distributed over the encapsulated key space.
10. *Special extended smoothness:* For any fixed public/secret key pair, the algorithm dDecap, fed with a ciphertext generated via $\mathsf{Encap}_{\mathsf{c}}^{*}$, a secret random tag, and the fixed secret key, will output a key uniformly distributed over the encapsulated key space.

Consistent with [LZH+23], we require that there exist Sigma protocols to prove that some results are precisely output by KG, $\mathsf{Encap}_{\mathsf{c}}$, $\mathsf{Encap}_{\mathsf{k}}$, $\mathsf{Encap}_{\mathsf{c}}^{*}$, $\mathsf{dEncap}_{\mathsf{k}}$, $\mathsf{SamEncK}$ and $\mathsf{dSamEncK}$.

A concrete construction of dual HPS-KEM$^{\Sigma}$ based on the DDH assumption is provided in Sec. 5. Similar to [LZH+23], our dual HPS-KEM$^{\Sigma}$ construction can also be extended to be based on the k-linear assumption [HK07,Sha07].

An MAMF Framework. Now, we briefly outline the generic construction of an MAMF from USPCE and dual HPS-KEM$^{\Sigma}$. The main idea is as follows.

Here, Setup algorithm directly invokes the setup algorithm of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, $\mathsf{KG}_{\mathsf{Ag}}$ calls the setup algorithm of USPCE, KG_{J} invokes the key generation algorithms of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ and USPCE (e.g., $pk_{\mathsf{J}} = (pk'_{\mathsf{J}}, pk_{\mathsf{USPCE}})$ where pk'_{J} is output by the key generation algorithm of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ and pk_{USPCE} is output by the key generation algorithm of USPCE), while KG_{u} solely calls the key generation algorithm of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$.

The algorithm Frank, executed by the sender to generate an MAMF signature for a message m, proceeds as follows. It utilizes $\mathsf{Encap}_{\mathsf{c}}$ to generate a well-formed encapsulated ciphertext c, and then employs $\mathsf{Encap}_{\mathsf{k}}$ to generate an encapsulated key k_{r} for the receiver and $\mathsf{dEncap}_{\mathsf{k}}$ to generate k_{J} (associated with a randomly chosen tag t) for the judge, where $\mathsf{Encap}_{\mathsf{c}}$, $\mathsf{Encap}_{\mathsf{k}}$ and $\mathsf{dEncap}_{\mathsf{k}}$ use the same randomness r. Following this, it calls the encryption algorithm of USPCE to encrypt the tag t with randomness r_{USPCE}, using the message m as the item, to obtain a ciphertext c_{t}. Finally, it outputs a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$, where π is a NIZK proof (generated with witness $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$) for the relation $\mathcal{R}$ in Fig. 1.

In the verification process (i.e., the algorithm Verify), the receiver confirms the signature's validity by checking (i) if the NIZK proof is valid, and (ii) if the decapsulated key, produced by decapsulating c via Decap, matches the key k_{r} provided in the signature.

In the moderation process (i.e., the algorithm Judge), if m is in the illegal message set, or the legislative agency have provided a token (by TKGen) for m to implement retrospective content moderation, the judge first decrypts c_{t} with the decryption algorithm of USPCE (with item m) to obtain a tag t. Then, he/she checks (i) if the NIZK proof is valid, and (ii) if the decapsulated key, produced by decapsulating c with tag t via dDecap, matches the key k_{J} provided in the signature.

In the token generation process (i.e., the algorithm TKGen), the legislative agency directly invokes the token generation algorithm of USPCE.

$$
\begin{aligned}
\mathcal{R} = \{((&\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}}, m), (sk_{\mathsf{s}}, t, r, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})) : \\
&(\ \ (pk_{\mathsf{s}}, sk_{\mathsf{s}}) \in \mathcal{R}_{\mathrm{s}} \\
&\quad \wedge (\ ((c, k_{\mathsf{J}}, pk'_{\mathsf{J}}), (\boxed{t}, r)) \in \mathcal{R}_{\mathrm{c,k}}^{\mathsf{d}} \ \wedge_{\mathrm{eq}}\ ((pk_{\mathsf{USPCE}}, m, c_{\mathrm{t}}), (\boxed{t}, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathrm{ct}}\ \ ) \\
&\vee (\ \ (c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathrm{c}}^*\ \wedge\ (k_{\mathsf{r}}, r_{\mathsf{k}}^*) \in \mathcal{R}_{\mathrm{k}}^*\ \wedge\ ((pk_{\mathsf{USPCE}}, m, c_{\mathrm{t}}), (t, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathrm{ct}}\ \ ) \\
&\vee (\ (c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathrm{c}}^* \wedge ((k_{\mathsf{J}}, (\boxed{t}, r_{\mathsf{k}}^*)) \in \mathcal{R}_{\mathrm{k}}^{\mathsf{d}*} \wedge_{\mathrm{eq}} ((pk_{\mathsf{USPCE}}, m, c_{\mathrm{t}}), (\boxed{t}, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathrm{ct}}\ )\ )\}
\end{aligned}
$$

Fig. 1 Relation $\mathcal{R}$ for MAMF, where $\mathcal{R}_{\mathrm{s}}$ is a relation proving the validity of the sender's public/secret keys, $\mathcal{R}_{\mathrm{c,k}}^{\mathsf{d}}$ is a relation proving that (c, k_{J}) is generated via $\mathsf{Encap}_{\mathrm{c}}$ and $\mathsf{dEncap}_{\mathrm{k}}$ with the same randomness r, $\mathcal{R}_{\mathrm{c}}^*$ is a relation proving that c is a ciphertext output by $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^*, $\mathcal{R}_{\mathrm{ct}}$ is a relation proving the USPCE ciphertext is well-formed, $\mathcal{R}_{\mathrm{k}}^*$ is to prove the encapsulated key of the receiver is generated via $\mathsf{SamEncK}$, and $\mathcal{R}_{\mathrm{k}}^{\mathsf{d}*}$ is to prove the encapsulated key of the judge is generated via $\mathsf{dSamEncK}$. Note that the symbol "$\wedge_{\mathrm{eq}}$" represents an "EQUAL-AND$_l$" operation between two relations, signifying that part (e.g., $\boxed{t}$) of the sub-witnesses in the relations are equal. The formal definition and further discussions are presented in Appendix G.

Now, let's shift our focus to the forging algorithms Forge, RForge and JForge.

The relation $\mathcal{R}$ in Fig. 1 plays a pivotal role in the forging algorithms. Observe that the relation comprises three sub-relations connected by "OR" operations. The first sub-relation is crafted for the sender, ensuring that the message is genuinely sent by the sender and convincing the receiver that the judge can successfully trace the originator once the message is reported. The second and the third sub-relations are devised for the forging algorithms to ensure deniability.

Specifically, the algorithm Forge first invokes $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^* to generate an ill-formed encapsulated ciphertext c, and uniformly samples two encapsulated keys k_{r} and k_{J}, where k_{r} is sampled with $\mathsf{SamEncK}$ using randomness r_{k}^*. Following this, it uniformly chooses a tag t, and then calls the encryption algorithm of USPCE to encrypt t with randomness r_{USPCE}, using the message m as the item, to obtain a ciphertext c_{t}. Finally, it outputs a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$, where π is a NIZK proof (generated with witness $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$) for the relation $\mathcal{R}$.

The algorithm RForge first invokes $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^* to generate an ill-formed encapsulated ciphertext c, and computes an encapsulated key k_{r} by executing Decap to decapsulate c. Then, it chooses a random tag t, and samples k_{J} with $\mathsf{dSamEncK}$ using t and randomness r_{k}^*. Following this, it calls the encryption algorithm of USPCE to encrypt t with randomness r_{USPCE}, using the message m as the item, to obtain a ciphertext c_{t}. Finally, it outputs a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$, where π is a NIZK proof (generated with witness $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$) for the relation $\mathcal{R}$.

The algorithm JForge first invokes $\mathsf{Encap}_{\mathrm{c}}^*$ with randomness r_{c}^* to generate an ill-formed encapsulated ciphertext c, and computes an encapsulated key k_{J} by executing dDecap (with a random tag t) to decapsulate c. Then, it samples k_{r} with $\mathsf{SamEncK}$ using randomness r_{k}^*. Following this, it calls the encryption algorithm of USPCE to encrypt t with randomness r_{USPCE}, using the message m as the item, to obtain a ciphertext c_{t}. Finally, it outputs a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$,

where π is a NIZK proof (generated with witness $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$) for the relation $\mathcal{R}$.

In summary, we have presented a generic construction of MAMF from USPCE and dual HPS-KEM$^{\Sigma}$. By incorporating a concrete USPCE and a concrete dual HPS-KEM$^{\Sigma}$, we can derive a specific instantiation of MAMF.

Security analysis. We turn to show a high-level intuition that our MAMF framework achieves the required unforgeability, accountability, deniability, unframeability, untraceability, and confidentiality of sets.

Given the similarity in the security analysis of unforgeability and accountability, we will focus here on demonstrating how to achieve unforgeability.

Unforgeability requires that any adversary cannot generate a signature such that an honest receiver accepts it. Supposing that there is an adversary generating a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ such that an honest receiver accepts it, we have: (i) π is a valid proof for the relation $\mathcal{R}$, and (ii) $k_{\mathsf{r}} = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c)$. Observe that to generate the valid proof π for $\mathcal{R}$, the adversary needs to know witness $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$ or $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$.

- If the adversary knows $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$, it implies that sk_{s} is a valid secret key of the sender. Since the adversary possesses no information about the sender's secret key beyond the knowledge of the sender's public key, it is contradictory to SK-second-preimage resistance of dual HPS-KEM$^{\Sigma}$.
- If the adversary knows $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$, it implies that c is generated via $\mathsf{Encap}_{\mathsf{c}}^*$. The ciphertext unexplainability of dual HPS-KEM$^{\Sigma}$ guarantees that c is not well-formed with overwhelming probability. Thus, according to (ii), $(c, k_{\mathsf{r}}, r_{\mathsf{c}}^*)$ leads to a successful attack on universality of dual HPS-KEM$^{\Sigma}$.

Now, we turn to analyze universal deniability, receiver compromise deniability, and judge compromise deniability within our MAMF framework. Given the similarity in the security analysis of these deniability aspects, we will focus solely on demonstrating how judge compromise deniability is achieved.

Judge compromise deniability requires that any adversary with the judge's secret key cannot distinguish between the outputs $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ of Frank and JForge.

- Frank computes $(c \leftarrow \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r), k_{\mathsf{r}} \leftarrow \mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r), k_{\mathsf{J}} \leftarrow \mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t; r))$ with the same randomness r, where t is a random tag. On the other hand, JForge computes $c \leftarrow \mathsf{Encap}_{\mathrm{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*)$ and $k_{\mathsf{r}} \leftarrow \mathsf{SamEncK}(\mathsf{pp}; r_{\mathsf{k}}^*)$ with randomness r_{c}^* and r_{k}^*, respectively, and then decapsulates c by dDecap, using sk'_{J} and a random tag t, to obtain k_{J}.
  Note that for $c \leftarrow \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r)$, we obtain $k_{\mathsf{r}} = \mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r) = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c)$ and $k_{\mathsf{J}} = \mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t; r) = \mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t, c)$. The indistinguishability of dual HPS-KEM$^{\Sigma}$ guarantees that the tuple $(c, k_{\mathsf{r}}, k_{\mathsf{J}})$ output by Frank is indistinguishable from $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \widehat{k_{\mathsf{J}}})$, where $\widehat{c} \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*)$, $\widehat{k_{\mathsf{r}}} = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, \widehat{c})$ and $\widehat{k_{\mathsf{J}}} = \mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t, \widehat{c})$. Due to the smoothness of dual HPS-KEM$^{\Sigma}$, it guarantees that the tuple $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \widehat{k_{\mathsf{J}}})$ is indistinguishable from $(\widehat{c}, \tilde{k}_{\mathsf{r}}, \widehat{k_{\mathsf{J}}})$, where $\tilde{k}_{\mathsf{r}}$ is uniformly distributed over the encapsulated key space. According to the uniformity of sampled key by algorithm

$\mathsf{SamEncK}$ of dual HPS-KEM$^{\Sigma}$, $(\widehat{c}, \tilde{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}})$ is indistinguishable from that output by JForge. Thus, the output tuple $(c, k_{\mathsf{r}}, k_{\mathsf{J}})$ from Frank and that from JForge are indistinguishable.
- The ciphertext c_{t} output by Frank and that output by JForge are distributed identically.
- Frank generates a NIZK proof π for relation $\mathcal{R}$ with witness $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$, while JForge generates π for $\mathcal{R}$ with witness $(\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$. The zero knowledge property of NIZK guarantees that anyone cannot distinguish the proof output by Frank from that output by JForge.

Unframeability requires that any adversary (possessing the secret keys of the receiver, the legislative agency and the judge, but without the sender's secret key) cannot generate a signature such that both the receiver and the judge accept it. Suppose that there is an adversary generating a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ such that both the receiver and the judge accept it. The fact that the receiver accepts the signature implies: (i) π is a valid proof for the relation $\mathcal{R}$, and (ii) $k_{\mathsf{r}} = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c)$. Observe that to generate the valid proof π for $\mathcal{R}$, the adversary needs to know witness $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$ or $(\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$.

- If the adversary knows $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$, it implies that sk_{s} is a valid secret key of the sender. Similar to the previous analysis of unforgeability, it is contradictory to SK-second-preimage resistance of dual HPS-KEM$^{\Sigma}$.
- If the adversary knows $(\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, we turn our focus on the last two sub-relations of relation $\mathcal{R}$.
  - If $(\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$ satisfies

    $$(c, r^*_{\mathsf{c}}) \in \mathcal{R}^*_{\mathsf{c}} \ \wedge \ (k_{\mathsf{r}}, r^*_{\mathsf{k}}) \in \mathcal{R}^*_{\mathsf{k}} \ \wedge \ ((pk_{\mathsf{USPCE}}, m, c_{\mathrm{t}}), (t, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathsf{ct}},$$

    according to (ii), $(c, r^*_{\mathsf{c}}, k_{\mathsf{r}}, r^*_{\mathsf{k}})$ leads to a successful attack on the key unexplainability of dual HPS-KEM$^{\Sigma}$.
  - If $(\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$ satisfies

    $$(c, r^*_{\mathsf{c}}) \in \mathcal{R}^*_{\mathsf{c}} \wedge (k_{\mathsf{J}}, (\ t\ , r^*_{\mathsf{k}})) \in \mathcal{R}^{\mathsf{d}*}_{\mathsf{k}} \wedge_{\mathrm{eq}} ((pk_{\mathsf{USPCE}}, m, c_{\mathrm{t}}), (\ t\ , r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathsf{ct}},$$

    according to the fact that the judge accepts the signature (which further suggests $k_{\mathsf{J}} = \mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t, c)$), $(c, r^*_{\mathsf{c}}, k_{\mathsf{J}}, t, r^*_{\mathsf{k}})$ leads to a successful attack on the extended key unexplainability of dual HPS-KEM$^{\Sigma}$.

Next, we turn to analyze untraceability against judge and untraceability against agency within our MAMF framework. Given the similarity in the security analysis of these untraceability aspects, we will focus solely on demonstrating how untraceability against judge is achieved.

Untraceability against judge requires the existence of a simulator $\mathsf{SimFrank}$, such that any adversary with the judge's secret key cannot distinguish between the outputs $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ of Frank and $\mathsf{SimFrank}$, given that the message is not in the set of illegal messages.

– The algorithm $\mathsf{SimFrank}$ is constructed as follows. It computes $c \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*)$ and $k_{\mathsf{J}} \leftarrow \mathsf{dSamEncK}(\mathsf{pp}, t; r_{\mathsf{k}}^*)$ with randomness r_{c}^* and r_{k}^*, respectively, where t is a random tag, decapsulates c by Decap using sk_{r} to obtain k_{r}, and then computes c_{t} via encrypting t with the encryption algorithm of USPCE, using the message m as an item. After that, taking $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$ as the witness, it calls the proving algorithm of NIZK to generate a proof π. Finally, it outputs a signature $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$.
– Frank computes $(c \leftarrow \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r), k_{\mathsf{r}} \leftarrow \mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r), k_{\mathsf{J}} \leftarrow \mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t; r))$ with the same randomness r, where t is a random tag, and computes c_{t} via encrypting t with the encryption algorithm of USPCE, using the message m as an item.
Note that for $c \leftarrow \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r)$, we obtain $k_{\mathsf{r}} = \mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r) = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c)$ and $k_{\mathsf{J}} = \mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t; r) = \mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t, c)$. The indistinguishability of dual HPS-KEM$^{\Sigma}$ and the confidentiality against users of USPCE guarantee that the tuple $(c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ output by Frank is indistinguishable from $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \widehat{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$, where $\widehat{c} \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*)$, $\widehat{k_{\mathsf{r}}} = \mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, \widehat{c})$, $\widehat{k_{\mathsf{J}}} = \mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t, \widehat{c})$ and $\widehat{c_{\mathrm{t}}}$ is the ciphertext created via encrypting another random tag t' with the encryption algorithm of USPCE, using the message m as an item. Due to the special extended smoothness of dual HPS-KEM$^{\Sigma}$, it guarantees that the tuple $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \widehat{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$ is indistinguishable from $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \tilde{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$, where $\tilde{k_{\mathsf{J}}}$ is uniformly distributed over the encapsulated key space. According to the uniformity of sampled key by algorithm $\mathsf{dSamEncK}$ of dual HPS-KEM$^{\Sigma}$, $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \tilde{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$ is indistinguishable from that $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \overline{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$, where $\overline{k_{\mathsf{J}}} \leftarrow \mathsf{dSamEncK}(\mathsf{pp}, t; r_{\mathsf{k}}^*)$. The confidentiality against users of USPCE ensures that $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \overline{k_{\mathsf{J}}}, \widehat{c_{\mathrm{t}}})$ is indistinguishable from $(\widehat{c}, \widehat{k_{\mathsf{r}}}, \overline{k_{\mathsf{J}}}, c_{\mathrm{t}})$, which is the tuple output by $\mathsf{SimFrank}$. Thus, the output tuple $(c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}})$ from Frank and that from $\mathsf{SimFrank}$ are indistinguishable.
– Frank generates a NIZK proof π for relation $\mathcal{R}$ with witness $(sk_{\mathsf{s}}, t, r, \perp, \perp, r_{\mathsf{USPCE}})$, while $\mathsf{SimFrank}$ generates π for $\mathcal{R}$ with witness $(\perp, t, \perp, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})$. The zero knowledge property of NIZK guarantees that anyone cannot distinguish the proof output by Frank from that output by $\mathsf{SimFrank}$.

Roughly, confidentiality of sets requires the legislative agency's public key and the judge's public key will not disclose any information about the set of illegal messages (except for its size), which is trivially obtained from the confidentiality of sets of USPCE.

1.2 Discussions

One-time token for specific MAMF signature. In this paper, our focus is solely on illegal messages. It is worth noting that certain messages, like harassment messages and phishing links, may not universally qualify as illegal messages for all users. Consequently, these messages might not be encompassed within the set designated by the legislative agency. Therefore, without the assistance of the legislative agency, the moderator cannot ascertain the sender's identity in such scenarios. On the other hand, if the legislative agency provides tokens for the

messages, the moderator possesses the ability to identify all senders of these messages. To address this, a solution is to empower the legislative agency to generate a one-time token for a specific MAMF signature and a specific message, which is not in pre-defined set, such that the moderator can carry out content moderation for that specific signature and specific message. We stress that our scheme seamlessly accommodates this requirement, leveraging the inherent flexibility of our USPCE. Further elaboration on this aspect can be found in Appendix I.

MPC for the token generation. In our scheme, the token generation algorithm is invoked by legislative agency, which implicitly means that we assume that the agency would not augment message to illegal set arbitrarily. To mitigate trust in the agency, there exist some general methods. Essentially, the secret key used to generate tokens can be shared among multiple agencies using secret sharing techniques, and then secure multi-party computation (MPC) can be invoked to generate (one-time) tokens for messages deemed illegal by the majority.

Witness-only Sigma protocols. When building the Sigma protocol for the aforementioned relation $\mathcal{R}$, partially composed of sub-relations using $\wedge_{\text{eq}}$ operations, we introduce a novel property termed "witness-only" for Sigma protocols, which may have independent interests. Roughly speaking, if the prover in Sigma protocols can generate the commitment and response solely based on the input witness, without necessitating the use of the statement, then we characterize these Sigma protocols as witness-only. It's noteworthy that numerous Sigma protocols inherently possess this witness-only property. Subsequently, we illustrate the construction of a Sigma protocol for a relation composed of sub-relations using an $\wedge_{\text{eq}}$ operation, provided that there exists a witness-only Sigma protocol for each sub-relation. Additional details are available in Appendix G.

Predicate-based primitive. Similar to the construction of [BGJP23], our MAMF is constructed with polynomial-size sets of illegal messages. One might prefer to constructing MAMF with sets of illegal messages expressed with predicates, allowing the scheme to be applied to a broader range of scenarios. We leave it as an open problem to construct a practical MAMF without using cumbersome cryptographic tools (e.g., witness encryption or indistinguishability obfuscation).

1.3 Roadmap

Other related works are recalled in Appendix A. We recall some preliminaries in Sec. 2. Then in Sec. 3, we present the primitive of MAMF and formalize its security notions. Next, in Sec. 4 and in Sec. 5, we introduce primitives of USPCE and dual HPS-KEM$^{\Sigma}$, respectively. Taking USPCE and dual HPS-KEM$^{\Sigma}$ as building blocks, we provide a framework of constructing MAMF in Sec. 6.

2 Preliminaries

Throughout this paper, let λ denote the security parameter. For any $k \in \mathbb{N}$, let $[k] := \{1, 2, \cdots, k\}$. For a finite set S, we denote by $|S|$ the number of elements

in S, and denote by $a \leftarrow S$ the process of uniformly sampling a from S. For a distribution X, we denote by $a \leftarrow X$ the process of sampling a from X. For any probabilistic polynomial-time (PPT) algorithm Alg, let $\mathcal{RS}$ be the randomness space of Alg. We write $\mathsf{Alg}(x; r)$ for the process of Alg on input x with inner randomness $r \in \mathcal{RS}$, and use $y \leftarrow \mathsf{Alg}(x)$ to denote the process of running Alg on input x with $r \leftarrow \mathcal{RS}$, and assigning y the result. We write $\mathsf{negl}(\lambda)$ to denote a negligible function in λ and write $\mathsf{poly}(\lambda)$ to denote a polynomial.

For a polynomial-time relation $\mathcal{R} \subset \mathcal{Y} \times \mathcal{X}$, where $\mathcal{Y}$ is the statement space and $\mathcal{X}$ is the witness space, we say that x is a witness for y if $(y, x) \in \mathcal{R}$.

Due to space limitations, we have provided additional preliminaries in the appendices. Specifically, cryptographic assumptions are reviewed in Appendix B.1. Definitions of NIZK and Sigma protocols are revisited in Appendix B.2. The definition of cuckoo hash is outlined in Appendix B.3. Furthermore, a summary of set pre-constrained encryption is included in Appendix B.4.

3 Mild asymmetric message franking

In this section, we introduce a primitive known as *mild asymmetric message franking (MAMF)*, to establish mild content moderation for a messaging system, and formally define its security notions.

3.1 MAMF algorithms

Formally, an MAMF scheme $\mathsf{MAMF} = (\mathsf{Setup}, \mathsf{KG}_{\mathsf{Ag}}, \mathsf{KG}_{\mathsf{J}}, \mathsf{KG}_{\mathsf{u}}, \mathsf{Frank}, \mathsf{Verify}, \mathsf{TKGen}, \mathsf{Judge}, \mathsf{Forge}, \mathsf{RForge}, \mathsf{JForge})$ is a tuple of algorithms, encompassing four roles: a sender, a receiver, a legislative agency, and a judge. The scheme is associated with three public key spaces (for the legislative agency, the judge, and users, including senders and receivers, respectively), three corresponding secret key spaces, a message space $\mathcal{M}$, a token space $\mathcal{TK}$, and a signature space $\mathcal{SG}$. Without loss of generality, we assume that all public key inputs are in the corresponding public key space, all secret key inputs are in the corresponding secret key space, all m inputs are in $\mathcal{M}$, all tk inputs are in $\mathcal{TK}$, and all σ inputs are in $\mathcal{SG}$.

The detailed descriptions of the algorithms are as follows.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$: The setup algorithm takes the security parameter as input, and outputs a global public parameter pp.
- $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) \leftarrow \mathsf{KG}_{\mathsf{Ag}}(\mathsf{pp}, \mathsf{S})$: The key generation algorithm $\mathsf{KG}_{\mathsf{Ag}}$ takes pp and a set $\mathsf{S} \subseteq \mathcal{M}$ as input, and outputs a key pair $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}})$ for the legislative agency and an auxiliary parameter $\mathsf{ap}_{\mathsf{Ag}}$ for the judge's key generation.
- $(pk_{\mathsf{J}}, sk_{\mathsf{J}}) \leftarrow \mathsf{KG}_{\mathsf{J}}(\mathsf{pp}, pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$: The key generation algorithm KG_{J} takes $(\mathsf{pp}, pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$ as input, and outputs a key pair $(pk_{\mathsf{J}}, sk_{\mathsf{J}})$ for the judge. We assume that the well-formedness of the public key pk_{J} can be verified with the assistance of $\mathsf{ap}_{\mathsf{Ag}}$ and S.

- $(pk, sk) \leftarrow \mathsf{KG_u}(\mathsf{pp})$: The key generation algorithm $\mathsf{KG_u}$ takes pp as input, and outputs a key pair (pk, sk) for users. Below we usually use $(pk_\mathsf{s}, sk_\mathsf{s})$ (resp., $(pk_\mathsf{r}, sk_\mathsf{r})$) to denote the sender's (resp., the receiver's) public/secret key pair.
- $\sigma \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$: The franking algorithm takes the public parameter pp, a sender's secret key sk_s, a receiver's public key pk_r, the agency's public key pk_{Ag}, the judge's public key pk_J and a message m as input, and outputs a signature σ.
- $b \leftarrow \mathsf{Verify}(\mathsf{pp}, pk_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m, \sigma)$: The *deterministic* algorithm of the verification of the receiver takes $(\mathsf{pp}, pk_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J})$, a message m and a signature σ as input, and returns $b \in \{0, 1\}$, which indicates whether the receiver accepts the signature or not.
- $\mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$: The token generation algorithm, run by the agency, takes $(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_\mathsf{J})$ and a message m as input, and outputs a token tk.
- $b \leftarrow \mathsf{Judge}(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J}, m, \sigma, \mathsf{tk})$: The *deterministic* algorithm of verification of the judge takes $(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J})$, a message m, a signature σ and a token tk as input, and outputs a bit $b \in \{0, 1\}$. Note that, when $m \in \mathsf{S}$, the token tk can be $\perp$.
- $\sigma \leftarrow \mathsf{Forge}(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$: The universal forging algorithm, on input $(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J})$ and a message m, returns a "forged" signature σ.
- $\sigma \leftarrow \mathsf{RForge}(\mathsf{pp}, pk_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$: The receiver compromise forging algorithm takes $(\mathsf{pp}, pk_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J})$ and a message m as input, and returns a "forged" signature σ.
- $\sigma \leftarrow \mathsf{JForge}(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J}, m)$: The judge compromise forging algorithm takes $(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J})$ and a message m as input, and outputs a "forged" signature σ.

Correctness. For any normal signature generated by Frank, the correctness requires that (i) the receiver can call Verify to verify the signature successfully, and (ii) the judge can invoke Judge to validate a report successfully once they receive a valid report. The formal requirements are shown as follows.

Given any pp generated by Setup, any key pairs $(pk_\mathsf{s}, sk_\mathsf{s})$ and $(pk_\mathsf{r}, sk_\mathsf{r})$ output by $\mathsf{KG_u}$, any key pair $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$ for a set $\mathsf{S} \subseteq \mathcal{M}$ output by $\mathsf{KG}_{\mathsf{Ag}}$ (where S is selected by some authority, e.g., the agency), and any key pair $(pk_\mathsf{J}, sk_\mathsf{J})$ output by $\mathsf{KG_J}$ (with $(\mathsf{pp}, pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$ as input), we require that for any message $m \in \mathcal{M}$ and any $\sigma \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$, it holds with overwhelming probability that:

(1) $\mathsf{Verify}(\mathsf{pp}, pk_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m, \sigma) = 1$;
(2) if $m \in \mathsf{S}$, then $\mathsf{Judge}(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J}, m, \sigma, \mathsf{tk} = \perp) = 1$;
(3) if $m \notin \mathsf{S}$, then $\mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_\mathsf{J}, m)$, and $\mathsf{Judge}(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, pk_{\mathsf{Ag}}, sk_\mathsf{J}, m, \sigma, \mathsf{tk}) = 1$.

Remark 1. For the sake of simplicity, we posit the existence of an additional key verification algorithms $\mathsf{WellForm_u}$ and a token verification algorithm $\mathsf{WellForm_{tk}}$.

The algorithm $\mathsf{WellForm}_{\mathsf{u}}$ takes the public parameter and the key pairs of the receiver (or sender) as input, producing a bit b that signifies the well-formedness of the key pairs. The algorithm $\mathsf{WellForm}_{\mathsf{tk}}$ takes the public parameter, the judge's public key and the token as input, producing a bit b that signifies the well-formedness of the token.

3.2 Security notions for MAMF

We now formalize specific security notions for MAMF, including unforgeability, accountability, deniability, unframeability, untraceability, and confidentiality of sets.

Unforgeability. One of the paramount security requirements in secure messaging applications is the prevention of malicious impersonation. In essence, MAMF must guarantee that successful impersonation is thwarted, contingent upon the non-compromise of the respective individual's secret key.

Definition 1 (Unforgeability). *An MAMF scheme* MAMF *is* unforgeable*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Accountability. Adhering to the terminology established in AMF [TGL+19], we systematically formalize the security requirement pertaining to accountability as *receiver binding* and *sender binding*. Concretely, MAMF is to ensure that (i) no receivers can deceive the judge into accepting a message not genuinely sent by the sender, and (ii) no sender can produce a signature acceptable to the receiver while simultaneously rejected by the judge.

Now, we present the formal definitions as below.

Definition 2 (r-BIND). *An MAMF scheme* MAMF *is* receiver-binding*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Definition 3 (s-BIND). *An MAMF scheme* MAMF *is* sender-binding*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Deniability. To uphold deniability, MAMF needs to adhere to the secure properties of *universal deniability*, *receiver compromise deniability*, and *judge compromise deniability*.

Universal deniability indicates that any non-participating entity (i.e., lacking access to the sender's secret key, the receiver's secret key, or the judge's secret key) can generate a signature, indistinguishable from honestly-created signatures to other non-participating entities.

For receiver compromise deniability, the property requires that an entity with access to the receiver's secret key can generate a signature. This generated signature should be indistinguishable from honestly-created signatures to other entities with access to the corrupted secret key of the receiver.

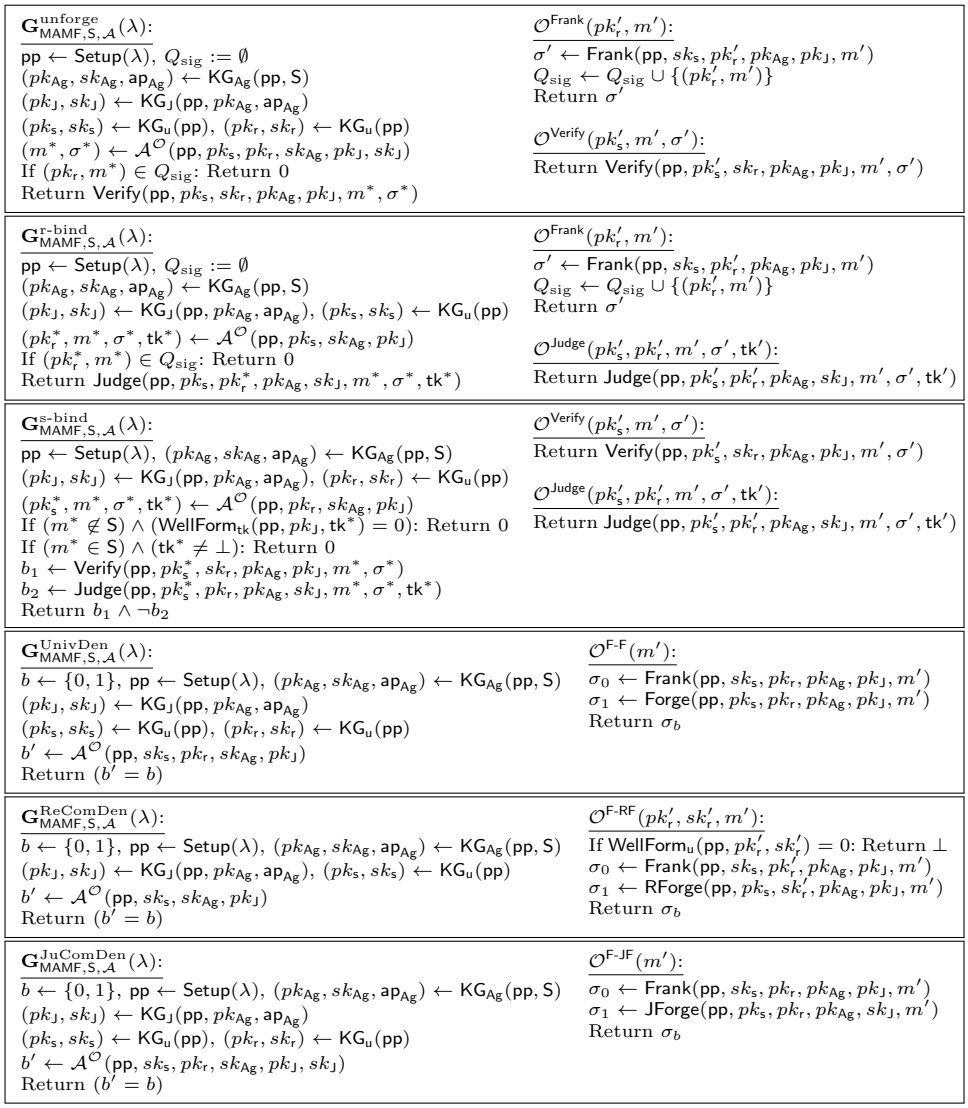

Fig. 2 Games for defining unforgeability, accountability and deniability of MAMF

As for judge compromise deniability, an entity with access to the judge's secret key should be capable of creating a signature. This signature should be indistinguishable from honestly-generated signatures for other entities with access to the judge's secret key.

The formal definitions are presented as follows.

Definition 4 (UnivDen). *An MAMF scheme* MAMF *is* universally deniable, *if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$, $\mathbf{Adv}^{\text{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$, *where* $\mathbf{G}^{\text{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Definition 5 (ReComDen). *An MAMF scheme* MAMF *is* receiver-compromise deniable*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{ReComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Definition 6 (JuComDen). *An MAMF scheme* MAMF *is* judge-compromise deniable*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{JuComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 2.*

Unframeability. The unframeability of MAMF requires that no party, even given a receiver's secret key and the judge's secret key, is able to produce a signature acceptable to both the receiver and the judge.

The formal definition is as follows.

Definition 7 (Unframeability). *An MAMF scheme* MAMF *is* unframeable*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 3.*

Untraceability. Ensuring untraceability constrains the capabilities of both the agency and the judge, thereby enhancing the assurance of sender privacy. Informally, untraceability implies an inability to discern the exact sender of a message. This concept is formalized into two distinct notions: untraceability against judge and untraceability against agency.

Untraceability against judge. When the message is not within the set S, untraceability against judge ensures that an entity possessing the judge's secret key cannot identify the sender of the message, without any assistance from the agency.

Definition 8 (Untraceability against judge). *An MAMF scheme* MAMF *has* untraceability against judge*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*, there is a simulator* $\mathsf{SimFrank}$*, such that* $\mathbf{Adv}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 3.*

Untraceability against agency. We also articulated untraceability against agency. In essence, a party with access to the agency's secret key is unable to discern the sender of a given message. The formal definition is articulated as follows.

Definition 9 (Untraceability against agency). *An MAMF scheme* MAMF *has* untraceability against agency*, if for any set* $\mathsf{S} \subseteq \mathcal{M}$ *and any PPT adversary* $\mathcal{A}$*, there is a simulator* $\mathsf{SimFrank}$*, such that* $\mathbf{Adv}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ *is shown in Fig. 3.*

Confidentiality of sets. We also consider the confidentiality of sets. It means the the public parameters and the public keys will not disclose any information about the pre-defined set S. The formal definition is outlined as follows.

Definition 10 (Confidentiality of sets). *An MAMF scheme* MAMF *supports* confidentiality of sets*, if for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,* $\mathbf{Adv}^{\text{conf-set}}_{\mathsf{MAMF},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{conf-set}}_{\mathsf{MAMF},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{conf-set}}_{\mathsf{MAMF},\mathcal{A}}(\lambda)$ *is shown in Fig. 3.*

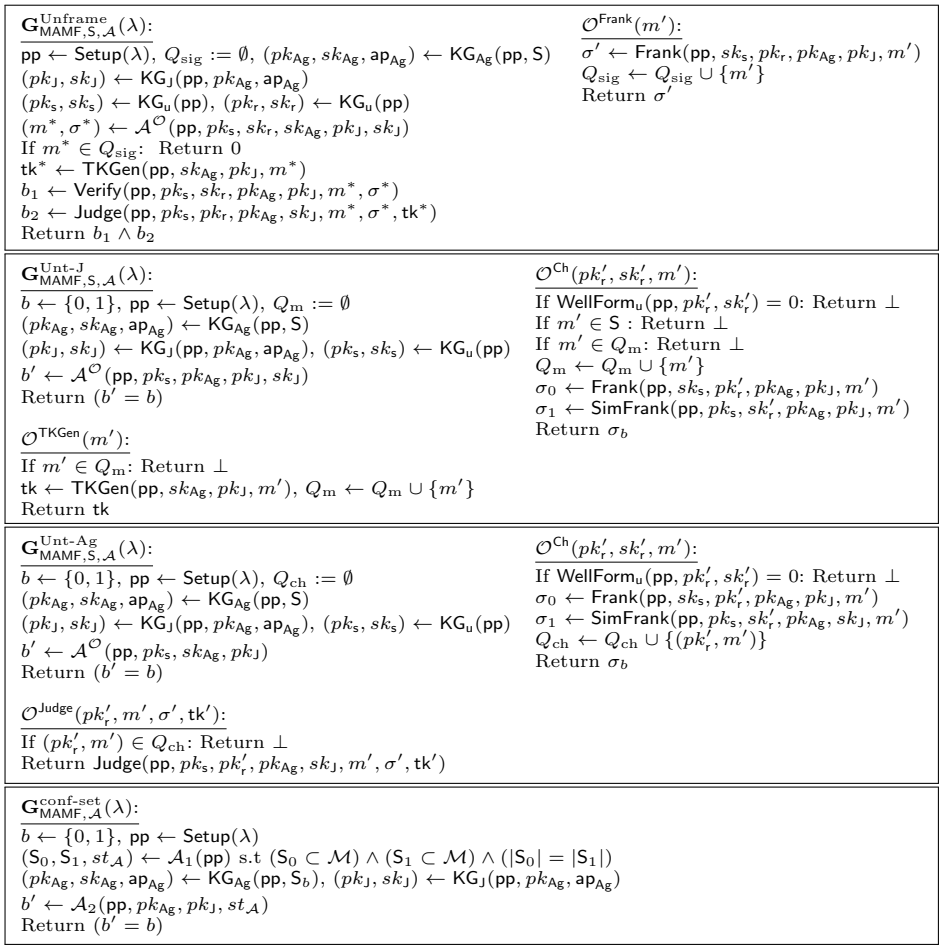

Fig. 3 Games for defining unframeability, untraceability, and confidentiality of sets of MAMF

Remark 2. Unlike AMF [TGL+19], where receiver binding and sender binding imply unforgeability, our notion differs. The unforgeability in [TGL+19] provides an adversary access to a judge oracle, while ours directly provides the judge's secret key. The unforgeability in [TGL+19] cannot prevent the receiver from accepting a signature forged by the judge (as discussed in [TGL+19, Appendix B], where signatures output by its JForge algorithm can be accepted by the receiver). Our unforgeability ensures that the receiver will not accept a signature forged by anyone else, including the agency and the judge, thus preventing deception.

In [TGL+19], the adversary in the judge compromise deniability game can access to *all* secret keys (including the sender's, the receiver's and the judge's). It means that the signature output by JForge can be accept by the receiver, which is unreasonable. In our model, the adversary only has access to the sender's and the judge's secret keys but *not to the receiver's secret key*. More importantly,

the definition of judge compromise deniability in [TGL+19] is contradictory to unframeability. Our deniability model makes space for unframeability.

Our notions have some areas to be strengthened, e.g., strong unforgeability. We believe the current definitions have grasped the key security requirements of MAMF. Some enhancement definitions can be considered for future work.

4 Universal set pre-constrained encryption

In this section, we introduce a new primitive, *universal set pre-constrained encryption (USPCE)*.

Definition. Let $\mathcal{U}$ denote a universe of elements, and $\mathcal{M}$ denote a message space. The universal set pre-constrained encryption USPCE contains five algorithms $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{TKGen}, \mathsf{Dec})$ and the details are as follows.

- $(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S})$: The setup algorithm, run by the authority, takes as input a security parameter λ and a set $\mathsf{S} \subseteq \mathcal{U}$ of size at most n, and outputs a public parameter pp, a auxiliary parameter ap and a master secret key msk.
- $(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap})$: The key generation algorithm is run by the users. It takes $(\mathsf{pp}, \mathsf{ap})$ as input, and outputs a public key pk and a secret key sk. We assume that the well-formedness of pk can be verified with the assistance of ap and S.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)$: The encryption algorithm takes (pp, pk), an item $x \in \mathcal{U}$ and a message $m \in \mathcal{M}$ as input, and outputs a ciphertext ct.
- $\mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, msk, x)$: The token generation algorithm takes (pp, msk, x) as input, and outputs a token tk for x.
- $m/S_{\mathrm{m}} \leftarrow \mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \mathsf{tk})$: The decryption algorithm takes $(\mathsf{pp}, sk, \mathsf{ct}, \mathsf{tk})$ as input, and outputs either a message m or a polynomial-size set $S_{\mathrm{m}} \subset \mathcal{M}$.

Given a set $\mathsf{S} \subseteq \mathcal{U}$, for any pp and msk generated by $\mathsf{Setup}(\lambda, \mathsf{S})$, we define a relation as follows:

$$\mathcal{R}_{\mathsf{ct}} = \{((pk, x, \mathsf{ct}), (m, r)) : \mathsf{ct} = \mathsf{Enc}(\mathsf{pp}, pk, x, m; r)\}. \tag{1}$$

We require that there is a *witness-only* Sigma protocol (please refer to the definition of witness-only in Appendix G) for the relation $\mathcal{R}_{\mathsf{ct}}$ in Eq. (1).

It also satisfies the following properties.

Definition 11 (Correctness). *An USPCE scheme* USPCE *is* correct*, if for any* $\lambda \in \mathbb{N}$*, any set* $\mathsf{S} \subset \mathcal{U}$*, and any* $m \in \mathcal{M}$*, it holds that*

– *when* $x \in \mathsf{S}$*:*

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S}) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)\end{array} : m \in S_{\mathrm{m}} = \mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \perp)\right] = 1 - \mathsf{negl}(\lambda);$$

$\underline{\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)}$:
$b \leftarrow \{0,1\}$, $(\mathsf{pp},\mathsf{ap},msk) \leftarrow \mathsf{Setup}(\lambda,\mathsf{S})$, $(pk,sk) \leftarrow \mathsf{KG}(\mathsf{pp},\mathsf{ap})$
$(m_0,m_1,x^*,st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{pp},msk,pk)$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp},pk,x^*,m_b)$, $b' \leftarrow \mathcal{A}_2(\mathsf{ct},st_{\mathcal{A}})$
Return $(b'=b)$

$\underline{\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)}$:
$b \leftarrow \{0,1\}$, $(\mathsf{pp},\mathsf{ap},msk) \leftarrow \mathsf{Setup}(\lambda,\mathsf{S})$, $Q_{\mathrm{x}} := \emptyset$, $U_{\mathrm{x}} := \emptyset$
$(pk,sk) \leftarrow \mathsf{KG}(\mathsf{pp},\mathsf{ap})$, $(m_0,m_1,x^*,st_{\mathcal{A}}) \leftarrow \mathcal{A}_1^{\mathcal{O}}(\mathsf{pp},pk,sk)$
If $(x^* \notin \mathcal{U}) \vee (x^* \in \mathsf{S}) \vee (x^* \in Q_{\mathrm{x}})$: Return $\perp$
$U_{\mathrm{x}} \leftarrow U_{\mathrm{x}} \cup \{x^*\}$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp},pk,x^*,m_b)$, $b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(\mathsf{ct},st_{\mathcal{A}})$
Return $(b'=b)$

$\underline{\mathcal{O}^{\mathsf{TKGen}}(x')}$:
If $x' \in U_{\mathrm{x}}$: Return $\perp$
$Q_{\mathrm{x}} \leftarrow Q_{\mathrm{x}} \cup \{x'\}$
Return $\mathsf{TKGen}(\mathsf{pp},msk,x')$

$\underline{\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)}$:
$b \leftarrow \{0,1\}$, $(\mathsf{S}_0,\mathsf{S}_1,st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\lambda)$ s.t $(\mathsf{S}_0 \subset \mathcal{U}) \wedge (\mathsf{S}_1 \subset \mathcal{U}) \wedge (|\mathsf{S}_0| = |\mathsf{S}_1|)$
$(\mathsf{pp},\mathsf{ap},msk) \leftarrow \mathsf{Setup}(\lambda,\mathsf{S}_b)$, $(pk,sk) \leftarrow \mathsf{KG}(\mathsf{pp},\mathsf{ap})$, $b' \leftarrow \mathcal{A}_2(\mathsf{pp},pk,st_{\mathcal{A}})$
Return $(b'=b)$

Fig. 4 Games $\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$, $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ and $\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)$ for USPCE

– *when* $x \notin \mathsf{S}$*:*

$$\Pr\left[\begin{array}{l}(\mathsf{pp},\mathsf{ap},msk) \leftarrow \mathsf{Setup}(\lambda,\mathsf{S}) \\ (pk,sk) \leftarrow \mathsf{KG}(\mathsf{pp},\mathsf{ap}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp},pk,x,m) \\ \mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp},msk,x)\end{array} : m = \mathsf{Dec}(\mathsf{pp},sk,\mathsf{ct},\mathsf{tk})\right] = 1 - \mathsf{negl}(\lambda).$$

Confidentiality against authority. Here, we address the matter of confidentiality against the authority. In a nutshell, the authority cannot obtain meaningful information about the message from a ciphertext.

Definition 12 (Confidentiality against authority). *An USPCE scheme* USPCE *has* confidentiality against authority*, if for any set* $\mathsf{S} \subseteq \mathcal{U}$ *and any PPT adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$*,* $\mathbf{Adv}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda) := |\Pr[\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)=1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ *is shown in Fig. 4.*

Confidentiality against users. We extend our considerations to confidentiality against users. Informally, It is required that, without the token for an item $x \notin \mathsf{S}$ given by the authority, any user cannot obtain meaningful information about the message from a ciphertext associated with x.

Definition 13 (Confidentiality against users). *An USPCE scheme* USPCE *has* confidentiality against users*, if for any set* $\mathsf{S} \subseteq \mathcal{U}$ *and any PPT adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$*,* $\mathbf{Adv}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda) := |\Pr[\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)=1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ *is shown in Fig. 4.*

Confidentiality of sets. Then, we delve into the concept of confidentiality of sets. It is required that the public parameters and a user's public key will not disclose any information about the pre-defined set S.

Definition 14 (Confidentiality of sets). *A USPCE scheme* USPCE *supports* confidentiality of sets*, if for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$*,* $\mathbf{Adv}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)=1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)$ *is shown in Fig. 4.*

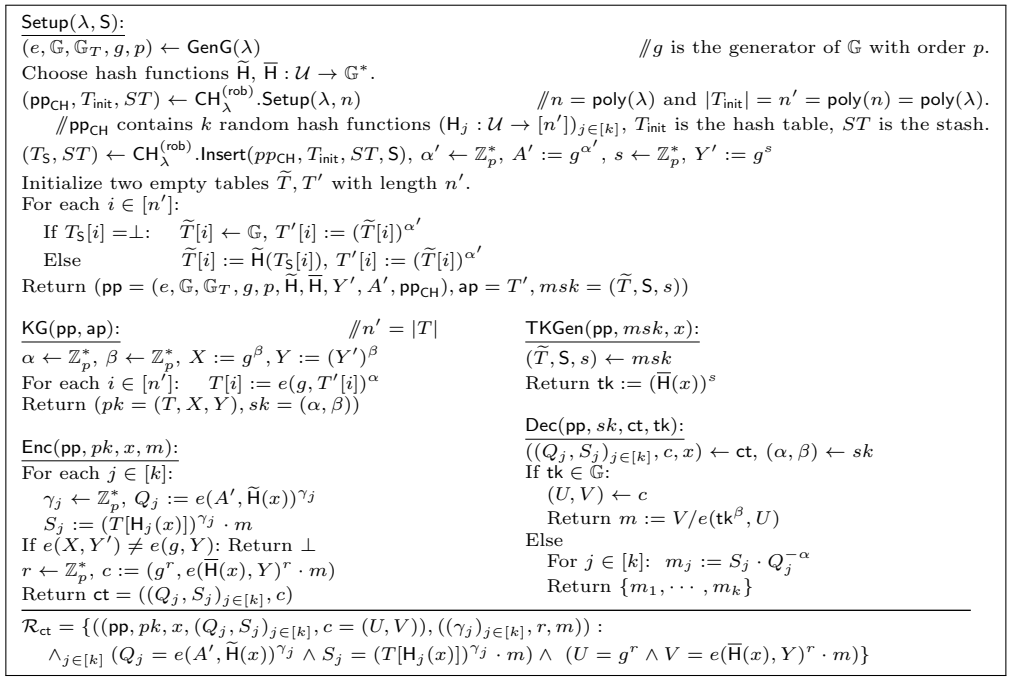

Fig. 5 A concrete USPCE scheme USPCE ($\mathcal{M} \subseteq \mathbb{G}_T$.)

Construction. Let $\mathsf{CH}_\lambda^{(\mathsf{rob})} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$ be an ϵ-robust cuckoo hashing scheme outlined in Appendix B.3, of which the negligibility of ϵ is ensured. More exactly, given the security parameter λ, the setup algorithm chooses $k = \lambda$ hash functions, the size of the hash table is $n' = 2 \cdot \lambda \cdot n$, and the size of the stash is 0. Let GenG be a group generation algorithm for bilinear maps, which takes the security parameter as input and outputs the group description $(e, \mathbb{G}, \mathbb{G}_T, g, p)$. Then, we provide a concrete USPCE scheme USPCE as shown in Fig. 5.

We show how to prove the relation $\mathcal{R}_{\mathsf{ct}}$, and the concrete relation $\mathcal{R}_{\mathsf{ct}}$ is presented in Fig. 5. We can prove the well-formedness of S_j and V using the Sigma protocols in Appendix F, while proving the well-formedness of Q_j and U using Schnorr's Sigma protocol [Sch89]. Furthermore, applying the "AND-EQUAL$_l$" operations in Appendix G over these sigma protocols, we can obtain a Sigma protocol for $\mathcal{R}_{\mathsf{ct}}$.

Remark 3. We can construct the algorithm $\mathsf{WellForm}_{\mathsf{tk}}$ mentioned in Remark 1 in this way: if $e(\mathsf{tk}, g) \neq e(\overline{\mathsf{H}}(x), Y')$, then return 0, otherwise return 1.

We analyze the correctness of USPCE as follows.

For any $\mathsf{S} \subset \mathcal{U}$, any $(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S})$, any $(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap})$, and any $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)$,

- when $x \in \mathsf{S}$, the properties of cuckoo hashing guarantee that x is inserted in one of locations (e.g., $\mathsf{H}_1(x), \ldots, \mathsf{H}_k(x)$) in T_{S}. Assuming x is located at

$\mathsf{H}_j(x)$ in the table, we obtain $\widetilde{\mathsf{H}}(x) = \widetilde{T}[\mathsf{H}_j(x)]$. Hence,

$$\begin{aligned} S_j \cdot Q_j^{-\alpha} &= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(A', \widetilde{\mathsf{H}}(x))^{-\alpha\gamma_j} = (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g^{\alpha'}, \widetilde{\mathsf{H}}(x))^{-\alpha\gamma_j} \\ &= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, (\widetilde{\mathsf{H}}(x))^{\alpha'})^{-\alpha\gamma_j} = (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, (\widetilde{T}[\mathsf{H}_j(x)])^{\alpha'})^{-\alpha\gamma_j} \\ &= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, T'[\mathsf{H}_j(x)])^{-\alpha\gamma_j} = (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot (T[\mathsf{H}_j(x)])^{-\gamma_j} = m \end{aligned}$$

Hence, it is affirmed that $m \in S_{\mathrm{m}} = \{m_1, \cdots, m_k\}$, where $k = \lambda$ is a polynomial, indicating that the set S_{m} is polynomial.

– when $x \notin \mathsf{S}$, for $\mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, msk, x)$, we obtain

$$\begin{aligned} V/e(\mathsf{tk}^{\beta}, U) &= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e((\overline{\mathsf{H}}(x))^{s\beta}, g^r) = e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), (g^s)^{\beta})^r \\ &= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), (Y')^{\beta})^r = e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), Y)^r = m \end{aligned}$$

Security analysis. Now, we show that the USPCE in Fig. 5 satisfies the aforementioned security requirements. Formally, we have the following theorem, the proof of which is given in Appendix C.

Theorem 1. USPCE *achieves confidentiality against authority, confidentiality against users, and confidentiality of sets.*

5 Dual HPS-KEM$^{\Sigma}$

In this section, we introduce a new primitive called *dual HPS-based KEM supporting Sigma protocols (dual HPS-KEM$^{\Sigma}$)*. We present a dual HPS-KEM$^{\Sigma}$ scheme based on the DDH assumption. Similar to [LZH+23], our scheme can also be extended to be based on the k-linear assumption [HK07,Sha07].

Definition. A dual HPS-KEM$^{\Sigma}$ scheme $\mathsf{dHPS\text{-}KEM}^{\Sigma} = (\mathsf{KEMSetup}, \mathsf{KG}, \mathsf{CheckKey}, \mathsf{Encap}_{\mathrm{c}}, \mathsf{Encap}_{\mathrm{c}}^*, \mathsf{Encap}_{\mathrm{k}}, \mathsf{Decap}, \mathsf{dEncap}_{\mathrm{k}}, \mathsf{dDecap}, \mathsf{SamEncK}, \mathsf{dSamEncK}, \mathsf{CheckCwel})$ is a tuple of algorithms associated with a secret key space $\mathcal{SK}$, an encapsulated key space $\mathcal{K}$, and a tag space $\mathcal{T}$, where $\mathsf{Encap}_{\mathrm{c}}$, $\mathsf{Encap}_{\mathrm{k}}$ and $\mathsf{dEncap}_{\mathrm{k}}$ have the same randomness space $\mathcal{RS}$. We use $\mathcal{RS}^*$ to denote the randomness space of $\mathsf{Encap}_{\mathrm{c}}^*$.

- $\mathsf{pp} \leftarrow \mathsf{KEMSetup}(\lambda)$: On input a security parameter λ, it outputs a public parameter pp.
- $(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp})$: On input the public parameter pp, it outputs a pair of public/secret keys (pk, sk).
- $b \leftarrow \mathsf{CheckKey}(\mathsf{pp}, sk, pk)$: On input the public parameter pp, a secret key sk and a public key pk, it outputs a bit b. Let $\mathcal{SK}_{\mathsf{pp},pk} := \{sk \in \mathcal{SK} \mid \mathsf{CheckKey}(\mathsf{pp}, sk, pk) = 1\}$.
- $c \leftarrow \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r)$: On input the public parameter pp with inner randomness $r \in \mathcal{RS}$, it outputs a well-formed ciphertext c. Let $\mathcal{C}_{\mathsf{pp}}^{\text{well-f}} := \{c = \mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r) \mid r \in \mathcal{RS}\}$.
- $c \leftarrow \mathsf{Encap}_{\mathrm{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*)$: On input the public parameter pp with inner randomness $r_{\mathsf{c}}^* \in \mathcal{RS}^*$, it outputs a ciphertext c. Let $\mathcal{C}_{\mathsf{pp}}^* := \{\mathsf{Encap}_{\mathrm{c}}^*(\mathsf{pp}; r_{\mathsf{c}}^*) \mid r_{\mathsf{c}}^* \in \mathcal{RS}^*\}$. We require $\mathcal{C}_{\mathsf{pp}}^{\text{well-f}} \subset \mathcal{C}_{\mathsf{pp}}^*$.

- $k \leftarrow \mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk; r)$: On input the public parameter pp and a public key pk with inner randomness $r \in \mathcal{RS}$, it outputs an encapsulated key $k \in \mathcal{K}$.
- $k' \leftarrow \mathsf{Decap}(\mathsf{pp}, sk, c)$: On input the public parameter pp, a secret key sk and a ciphertext c, and it outputs an encapsulated key $k' \in \mathcal{K}$.
- $k_{\mathsf{d}} \leftarrow \mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}, pk, t; r)$: On input the public parameter pp, a public key pk, and a tag $t \in \mathcal{T}$ with inner randomness $r \in \mathcal{RS}$, it outputs an encapsulated key $k \in \mathcal{K}$.
- $k'_{\mathsf{d}} \leftarrow \mathsf{dDecap}(\mathsf{pp}, sk, t, c)$: On input the public parameter pp, a secret key sk, a tag $t \in \mathcal{T}$ and a ciphertext c, it outputs an encapsulated key $k'_{\mathsf{d}} \in \mathcal{K}$.
- $k \leftarrow \mathsf{SamEncK}(\mathsf{pp}; r^*_{\mathsf{k}})$: On input the public parameter pp with inner randomness $r^*_{\mathsf{k}} \in \mathcal{RS}^*$, it outputs an encapsulated key $k \in \mathcal{K}$.
- $k_{\mathsf{d}} \leftarrow \mathsf{dSamEncK}(\mathsf{pp}, t; r^*_{\mathsf{k}})$: On input the public parameter pp and a tag $t \in \mathcal{T}$ with inner randomness $r^*_{\mathsf{k}} \in \mathcal{RS}^*$, it outputs an encapsulated key $k_{\mathsf{d}} \in \mathcal{K}$.
- $b \leftarrow \mathsf{CheckCwel}(\mathsf{pp}, c, r^*_{\mathsf{c}})$: On input the public parameter pp, a ciphertext c and a random number $r^*_{\mathsf{c}} \in \mathcal{RS}^*$, it outputs a bit b.

Correctness requirements are as follows.

(1) For any pp generated by $\mathsf{KEMSetup}(\lambda)$, and any (pk, sk) output by $\mathsf{KG}(\mathsf{pp})$, $\mathsf{CheckKey}(\mathsf{pp}, sk, pk) = 1$.
(2) For any pp generated by $\mathsf{KEMSetup}(\lambda)$, any (pk, sk) satisfying $\mathsf{CheckKey}(\mathsf{pp}, sk, pk) = 1$, any $t \in \mathcal{T}$, any randomness $r \in \mathcal{RS}$ and $c = \mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r)$, it holds that $\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, sk; r) = \mathsf{Decap}(\mathsf{pp}, sk, c)$, and $\mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}, pk, t; r) = \mathsf{dDecap}(\mathsf{pp}, sk, t, c)$.
(3) For any pp generated by $\mathsf{KEMSetup}(\lambda)$, and any c generated with $\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$, $\mathsf{CheckCwel}(\mathsf{pp}, c, r^*_{\mathsf{c}}) = 1$ if and only if $c \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$.

For any pp generated by $\mathsf{KEMSetup}(\lambda)$, we define some relations as follows:

$$
\begin{aligned}
\mathcal{R}_{\mathsf{s}} &= \{(pk, sk) : \mathsf{CheckKey}(\mathsf{pp}, sk, pk) = 1\}, \mathcal{R}^*_{\mathsf{c}} = \{(c, r^*_{\mathsf{c}}) : c = \mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r^*_{\mathsf{c}})\} \\
\mathcal{R}_{\mathsf{c,k}} &= \{((c, k, pk), r) : (c = \mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r)) \wedge (k = \mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk; r))\} \\
\mathcal{R}^{\mathsf{d}}_{\mathsf{c,k}} &= \{((c, k_{\mathsf{d}}, pk), (t, r)) : (c = \mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r)) \wedge (k_{\mathsf{d}} = \mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}, pk, t; r))\} \\
\mathcal{R}^*_{\mathsf{k}} &= \{(k, r^*_{\mathsf{k}}) : k = \mathsf{SamEncK}(\mathsf{pp}; r^*_{\mathsf{k}})\} \\
\mathcal{R}^{\mathsf{d}*}_{\mathsf{k}} &= \{(k_{\mathsf{d}}, (t, r^*_{\mathsf{k}})) : k_{\mathsf{d}} = \mathsf{dSamEncK}(\mathsf{pp}, t; r^*_{\mathsf{k}})\}
\end{aligned}
\tag{2}
$$

We require that for each relation in Eq. (2), there is a Sigma protocol. Note that for $\mathcal{R}^{\mathsf{d}}_{\mathsf{c,k}}$ and $\mathcal{R}^{\mathsf{d}*}_{\mathsf{k}}$, we further require a *witness-only* Sigma protocol (please refer to the definition of witness-only in Appendix G).

We also require that $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ should satisfy the following properties.

Definition 15 (Universality). $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is* universal*, if for any computationally unbounded adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 16 (Extended universality). $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is* extended universal*, if for any computationally unbounded adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

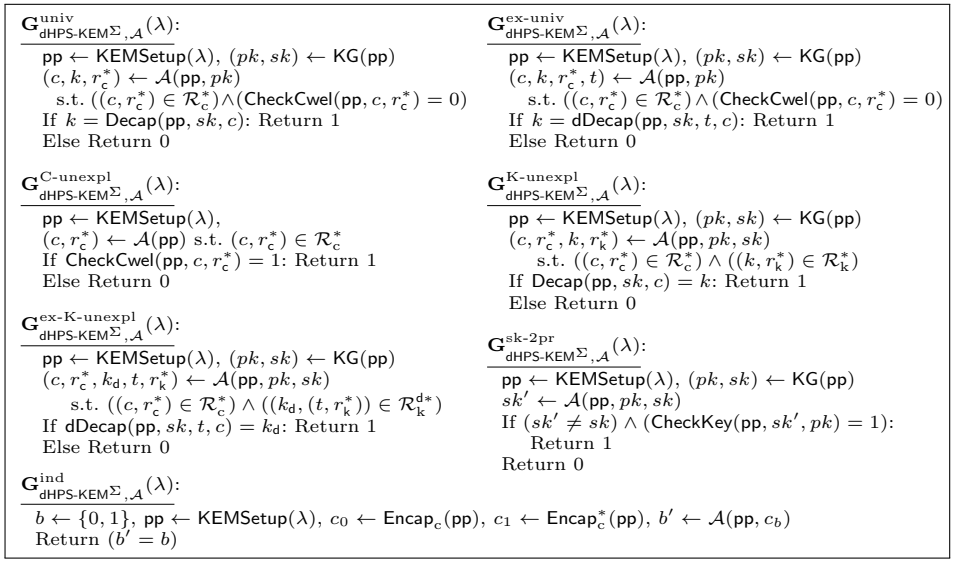

Fig. 6 Games for $\mathsf{dHPS\text{-}KEM}^\Sigma$

Definition 17 (Ciphertext unexplainability). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* ciphertext-unexplainable*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{C-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{C-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{C-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 18 (Key unexplainability). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* key-unexplainable*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 19 (Extended key unexplainability). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* extended key-unexplainable*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{ex-K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{ex-K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{ex-K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 20 (Indistinguishability). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* indistinguishable*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{ind}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\text{ind}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) = 1] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{ind}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 21 (SK-2PR). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* SK-second-preimage resistant*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $\mathbf{G}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

Definition 22 (Smoothness). $\mathsf{dHPS\text{-}KEM}^\Sigma$ *is* smooth*, if for any fixed* pp *generated by* $\mathsf{KEMSetup}$ *and any fixed* pk *generated by* KG*,* $\Delta((c,k),(c,k')) \leq \mathsf{negl}(\lambda)$*, where* $c \leftarrow \mathsf{Encap}_c^*(\mathsf{pp})$*,* $k \leftarrow \mathcal{K}$*,* $sk \leftarrow \mathcal{SK}_{\mathsf{pp},pk}$ *and* $k' = \mathsf{Decap}(\mathsf{pp}, sk, c)$*.*

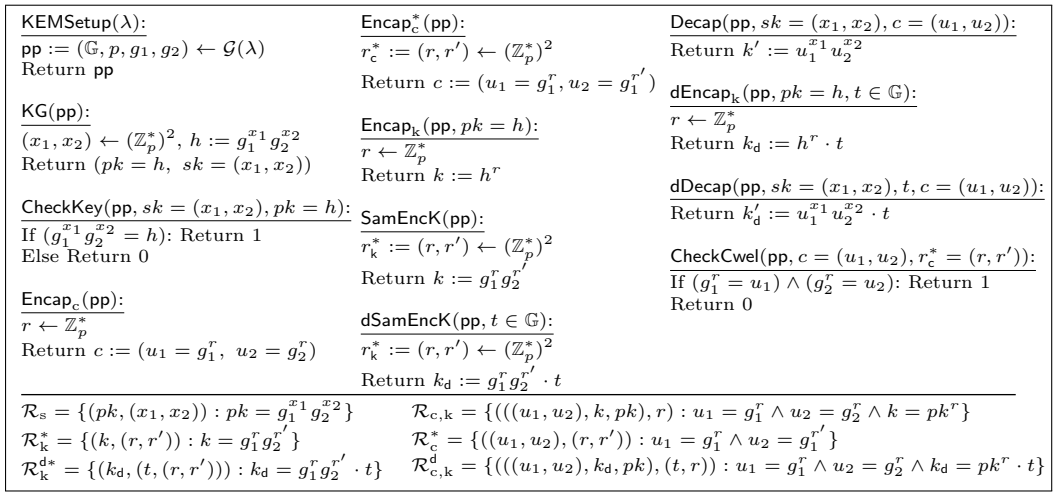

Fig. 7 Algorithm descriptions of a concrete $\mathsf{dHPS\text{-}KEM}^{\Sigma}$. There are Sigma protocols for relations $\mathcal{R}_{\mathsf{s}}$, $\mathcal{R}_{\mathsf{c,k}}$, $\mathcal{R}_{\mathsf{c}}^*$, $\mathcal{R}_{\mathsf{c,k}}^{\mathsf{d}}$, $\mathcal{R}_{\mathsf{k}}^*$ and $\mathcal{R}_{\mathsf{k}}^{\mathsf{d}*}$: Okamoto's Sigma protocol [Oka95] for $\mathcal{R}_{\mathsf{s}}$ and $\mathcal{R}_{\mathsf{k}}^*$, the Chaum-Pedersen protocol [CP92] for $\mathcal{R}_{\mathsf{c,k}}$, Schnorr's Sigma protocol [Sch89] for $\mathcal{R}_{\mathsf{c}}^*$, the Chaum-Pedersen protocol [CP92] and the Sigma protocol in Appendix F for $\mathcal{R}_{\mathsf{c,k}}^{\mathsf{d}}$ (requiring "AND-EQUAL$_l$" operations proposed in Appendix G), and Okamoto's Sigma protocol [Oka95] and the Sigma protocol in Appendix F for $\mathcal{R}_{\mathsf{k}}^{\mathsf{d}*}$ (the obtained Sigma protocol is witness-only as discussed in Appendix G).

Definition 23 (Extended smoothness). $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is* extended smooth, *if for any fixed* pp *generated by* $\mathsf{KEMSetup}$ *and any fixed* pk *generated by* KG, $\Delta((c, k, t), (c, k', t)) \leq \mathsf{negl}(\lambda)$, *where* $c \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp})$, $k \leftarrow \mathcal{K}$, $t \leftarrow \mathcal{T}$, $sk \leftarrow \mathcal{SK}_{\mathsf{pp},pk}$ *and* $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, c)$.

Definition 24 (Special extended smoothness). $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is* special extended smooth, *if for any fixed* pp *generated by* $\mathsf{KEMSetup}$ *and any fixed* (pk, sk) *generated by* KG, $\Delta((c, k), (c, k')) \leq \mathsf{negl}(\lambda)$, *where* $c \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp})$, $k \leftarrow \mathcal{K}$, $t \leftarrow \mathcal{T}$ *and* $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, c)$.

Definition 25 (Uniformity of sampled keys). $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *has* uniformity of sampled keys, *if for any* pp *generated by* $\mathsf{KEMSetup}$ *and any* $t \in \mathcal{T}$, *it holds that* $\Delta(k, k') = 0$ *and* $\Delta(k, k'') = 0$, *where* $k \leftarrow \mathcal{K}$, $k' \leftarrow \mathsf{SamEncK}(\mathsf{pp})$ *and* $k'' \leftarrow \mathsf{dSamEncK}(\mathsf{pp}, t)$.

Construction. Here, we present a concrete construction of dual $\mathsf{HPS\text{-}KEM}^{\Sigma}$, which satisfies all the aforementioned security properties. Let $\mathcal{G}$ be a group generation algorithm, taking λ as input and outputting $(\mathbb{G}, p, g_1, g_2)$, where $\mathbb{G}$ is a prime-order group, p is the order of $\mathbb{G}$, and g_1, g_2 are two random generators of $\mathbb{G}$. Our construction $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ is shown in Fig. 7.

It is clear that the construction in Fig. 7 satisfies *correctness*. The relations $\mathcal{R}_{\mathsf{s}}$, $\mathcal{R}_{\mathsf{c,k}}$, $\mathcal{R}_{\mathsf{c}}^*$, $\mathcal{R}_{\mathsf{c,k}}^{\mathsf{d}}$, $\mathcal{R}_{\mathsf{k}}^*$ and $\mathcal{R}_{\mathsf{k}}^{\mathsf{d}*}$ are presented in Fig. 7.

Remark 4. The algorithm $\mathsf{WellForm_u}$ mentioned in Remark 1 can be built in this way: given $sk = (x_1, x_2)$, if $pk \neq g_1^{x_1} g_2^{x_2}$, then return 0, otherwise return 1.

For security properties, we present the following theorem, the proof of which is given in Appendix D.

Theorem 2. *The above scheme* $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *achieves universality, extended universality, smoothness, extended smoothness, special extended smoothness, and uniformity of sampled keys. Furthermore,*

- $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is ciphertext-unexplainable, key-unexplainable and extended key-unexplainable under the DL assumption.*
- $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is indistinguishable under the DDH assumption.*
- $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *is SK-second-preimage resistant under the DL assumption.*

6 General construction of MAMF

In this section, we present a framework for constructing MAMF using USPCE and dual HPS-KEM$^{\Sigma}$.

Let $\mathsf{dHPS\text{-}KEM}^{\Sigma} = (\mathsf{KEMSetup}, \mathsf{KG}, \mathsf{CheckKey}, \mathsf{Encap_c}, \mathsf{Encap_c^*}, \mathsf{Encap_k}, \mathsf{Decap}, \mathsf{dEncap_k}, \mathsf{dDecap}, \mathsf{SamEncK}, \mathsf{dSamEncK}, \mathsf{CheckCwel})$ be a dual HPS-KEM$^{\Sigma}$ scheme, where $\mathcal{RS}$ denotes the randomness space of $\mathsf{Encap_c}$, $\mathsf{Encap_k}$ and $\mathsf{dEncap_k}$, $\mathcal{RS}^*$ denotes the randomness space of $\mathsf{Encap_c^*}$, $\mathsf{SamEncK}$ and $\mathsf{dSamEncK}$, $\mathcal{T}$ denotes the tag space, and $\mathcal{K}$ denotes the encapsulated key space.

Let $\mathsf{USPCE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{TKGen}, \mathsf{Dec})$ be a USPEC scheme with a universe of elements $\mathcal{U}$, a token space $\mathcal{TK}$ and a message space $\mathcal{M}$.

Our generic MAMF scheme $\mathsf{MAMF} = (\mathsf{Setup}, \mathsf{KG_{Ag}}, \mathsf{KG_J}, \mathsf{KG_u}, \mathsf{Frank}, \mathsf{Verify}, \mathsf{TKGen}, \mathsf{Judge}, \mathsf{Forge}, \mathsf{RForge}, \mathsf{JForge})$ is presented in Fig. 8. It's worth noting that the message space of MAMF $\mathcal{M} = \mathsf{USPCE}.\mathcal{U}$ and $\mathsf{USPCE}.\mathcal{M} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$.

We mainly introduce the algorithm Frank. It calls $\mathsf{Encap_c}$, $\mathsf{Encap_k}$, and $\mathsf{dEncap_k}$ of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ to generate a well-formed ciphertext and encapsulated keys, respectively. Subsequently, it invokes Enc of USPCE to encrypt the tag used for the generation of the encapsulated key of the judge. Afterward, it utilizes a NIZK proof algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ to create a NIZK proof. The relation $\mathcal{R}$ is defined as follows (which is also introduced in Fig. 1):

$$
\begin{aligned}
\mathcal{R} = \{&((\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m), (sk_{\mathsf{s}}, t, r, r_{\mathsf{c}}^*, r_{\mathsf{k}}^*, r_{\mathsf{USPCE}})) : \\
&((pk_{\mathsf{s}}, sk_{\mathsf{s}}) \in \mathcal{R}_{\mathsf{s}} \wedge (((c, k_{\mathsf{J}}, pk_{\mathsf{J}}), (\boxed{t}, r)) \in \mathcal{R}_{\mathsf{c,k}}^{\mathsf{d}} \ \wedge_{\mathrm{eq}} \ ((pk_{\mathsf{USPCE}}, m, c_{\mathsf{t}}), (\boxed{t}, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathsf{ct}})) \\
&\vee((c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathsf{c}}^* \ \wedge \ (k_{\mathsf{r}}, r_{\mathsf{k}}^*) \in \mathcal{R}_{\mathsf{k}}^* \ \wedge \ ((pk_{\mathsf{USPCE}}, m, c_{\mathsf{t}}), (t, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathsf{ct}}) \\
&\vee((c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathsf{c}}^* \ \wedge \ ((k_{\mathsf{J}}, (\boxed{t}, r_{\mathsf{k}}^*)) \in \mathcal{R}_{\mathsf{k}}^{\mathsf{d}*} \ \wedge_{\mathrm{eq}} \ ((pk_{\mathsf{USPCE}}, m, c_{\mathsf{t}}), (\boxed{t}, r_{\mathsf{USPCE}})) \in \mathcal{R}_{\mathsf{ct}}))\},
\end{aligned}
$$

For the NIZK proof system $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ utilized in Fig. 8, we construct it as follows. It's noteworthy that for every sub-relation (i.e., $\mathcal{R}_{\mathsf{s}}$, $\mathcal{R}_{\mathsf{c,k}}^{\mathsf{d}}$, $\mathcal{R}_{\mathsf{c}}^*$, $\mathcal{R}_{\mathsf{k}}^*$, $\mathcal{R}_{\mathsf{k}}^{\mathsf{d}*}$ and $\mathcal{R}_{\mathsf{ct}}$), the dual HPS-KEM$^{\Sigma}$ scheme and USPCE ensure the existence of a Sigma protocol. Utilizing the technique of trivially combining Sigma protocols for "AND/OR" operations [BS20, Sec. 19.7] and the "AND-EQUAL$_l$" operations (introduced in Appendix G), a new Sigma protocol for the relation

$\underline{\mathsf{Setup}(\lambda)}$: Return $\mathsf{pp} := \mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KEMSetup}(\lambda)$

$\underline{\mathsf{KG}_{\mathsf{Ag}}(\mathsf{pp}, \mathsf{S})}$: Return $(pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}, sk_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$

$\underline{\mathsf{KG}_{\mathsf{J}}(\mathsf{pp}, pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})}$:
$(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\mathsf{pp}_{\mathsf{KEM}})$, $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$
Return $(pk_{\mathsf{J}} = (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}), sk_{\mathsf{J}} = (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}}))$

$\underline{\mathsf{KG}_{\mathsf{u}}(\mathsf{pp})}$: Return $(pk, sk) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\mathsf{pp}_{\mathsf{KEM}})$

$\underline{\mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $r \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r)$, $k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk_{\mathsf{r}}; r)$
$t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk'_{\mathsf{J}}, t; r)$, $r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (sk_{\mathsf{s}}, t, r, \bot, \bot, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$, $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

$\underline{\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m, \sigma)}$:
$(\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}) \leftarrow \sigma$
$y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi, y) = 0$:
  Return 0
If $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}_{\mathsf{KEM}}, sk_{\mathsf{r}}, c) = k_{\mathsf{r}}$:
  Return 1
Return 0

$\underline{\mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$\mathsf{tk} \leftarrow \mathsf{USPCE.TKGen}(\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, m)$
Return tk

$\underline{\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m, \sigma, \mathsf{tk})}$:
$(sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}}) \leftarrow sk_{\mathsf{J}}$, $(\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}) \leftarrow \sigma$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi, y) = 0$: Return 0
If $\mathsf{tk} \neq \bot$:
  $t' \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c_{\mathsf{t}}, \mathsf{tk})$
  If $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}_{\mathsf{KEM}}, sk'_{\mathsf{J}}, t', c) = k_{\mathsf{J}}$: Return 1
  Return 0
If $\mathsf{tk} = \bot$:
  $S_{\mathsf{t}} \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c_{\mathsf{t}}, \bot)$
  For $t' \in S_{\mathsf{t}}$:
    If $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}_{\mathsf{KEM}}, sk'_{\mathsf{J}}, t', c) = k_{\mathsf{J}}$: Return 1
  Return 0

$\underline{\mathsf{Forge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r^*_{\mathsf{c}})$, $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$
$k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(pp; r^*_{\mathsf{k}})$, $t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$
$r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (\bot, t, \bot, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$, $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

$\underline{\mathsf{RForge}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r^*_{\mathsf{c}})$, $k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}_{\mathsf{KEM}}, sk_{\mathsf{r}}, c)$
$t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(pp, t; r^*_{\mathsf{k}})$
$r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (\bot, t, \bot, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$, $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

$\underline{\mathsf{JForge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m)}$:
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $(sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}}) \leftarrow sk_{\mathsf{J}}$, $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r^*_{\mathsf{c}})$
$r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(pp; r^*_{\mathsf{k}})$, $t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}_{\mathsf{KEM}}, sk'_{\mathsf{J}}, t, c)$
$r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (\bot, t, \bot, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$, $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

Fig. 8 Algorithms of MAMF

$\mathcal{R}$ is obtained (implied by [BS20, Sec. 19.7] and Theorem 6 in Appendix G). Subsequently, employing the Fiat-Shamir transform, we derive a NIZK proof system $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{R}$ in the random oracle model.

Correctness analysis. For any signature $\sigma \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)$, we parse $\sigma = (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$, and let $y := (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$.

We first analyze the output of Verify as follows: (i) the correctness of $\mathsf{NIZK}^{\mathcal{R}}$ guarantees that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(k_{\mathsf{r}}, \pi, y) = 1$; (ii) the correctness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that $\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c) = k_{\mathsf{r}}$. So, Verify will return 1.

Next, we analyze the output of Judge as follows: (i) the correctness of $\mathsf{NIZK}^{\mathcal{R}}$ guarantees that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(k_{\mathsf{r}}, \pi, y) = 1$; (ii) the correctness of USPCE guarantees that $t = \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c_{\mathsf{t}}, \mathsf{tk})$, where $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$; (iii) the correctness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that $\mathsf{Decap}(pp, sk'_{\mathsf{J}}, t, c) = k_{\mathsf{J}}$. Therefore, Judge will also return 1.

In fact, the second point (ii) of the correctness of Judge should be divided into two cases as the definition of correctness in Sec. 3. It can be trivially guaranteed by the correctness of USPCE, so we omit the details here.

Security analysis. We have the following theorem, the proof of which is placed in Appendix E due to space limitations.

Theorem 3. *If the USPCE scheme* USPCE *satisfies the properties defined in Sec. 4, the dual HPS-KEM*$^{\Sigma}$ *scheme* $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ *satisfies the properties defined in Sec. 5, and* $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ *is a Fiat-Shamir NIZK proof system for* $\mathcal{R}$*, then our scheme* MAMF *achieves the properties defined in Sec. 3.2.*

Concrete construction and improvements. Plugging the concrete USPCE in Sec. 4 and the concrete dual HPS-KEM$^{\Sigma}$ in Sec. 5 into our framework, we obtain a concrete MAMF scheme. Notably, our concrete USPCE is based on the DBDH assumption, featuring a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and our concrete dual HPS-KEM$^{\Sigma}$ is based on the DDH assumption. To integrate them into our framework, we require that the concrete dual HPS-KEM$^{\Sigma}$ is built over $\mathbb{G}_T$.

In Appendix H, we present some improvements on the concrete MAMF. Because of the algebraic structure of our USPCE and dual HPS-KEM$^{\Sigma}$, there exist Sigma protocols proving the well-formedness of the USPCE ciphertext and of the encapsulated key for the judge simultaneously. Therefore, we let the franking algorithm directly call the encryption algorithm of USPCE to encrypt the encapsulated key for the judge, instead of the tag t, and then change the relation $\mathcal{R}$ accordingly. Furthermore, by proving the same statements in different sub-relations simultaneously, we propose an enhancement to the Sigma protocol for the relation $\mathcal{R}$. As a result, it reduces about 2/3 of the space overhead for the response of the Sigma protocol, which implies a smaller signature size. More exact comparison can be found in Table 1 in Appendix H.

In our MAMF scheme, if the legislative agency has provided tokens for some messages, then the judge possesses the ability to identify all senders of these messages. However, it may not be suitable for all scenarios, as discussed in Introduction. We propose a solution in Appendix I, empowering the legislative agency to generate a one-time token for a specific MAMF signature and a specific message, such that the judge can carry out content moderation for that specific signature and specific message.

References

AAB+15. Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats: mandating insecurity by requiring government access to all data and communications. *J. Cybersecur.*, 1(1):69–79, 2015.

AC20. Thomas Attema and Ronald Cramer. Compressed-protocol theory and practical application to plug & play secure algorithmics. In *CRYPTO 2020*, pages 513–543. Springer, 2020.

AJJM22. Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Pre-constrained encryption. In *ITCS 2022*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

BBM+21. Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The apple psi system. *Apple, Inc., Tech. Rep*, 2021.

BBS+15. Adam Bates, Kevin RB Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan Wallach. Accountable wiretapping–or–i know they can hear you now. *J. Comput. Secur.*, 23(2):167–195, 2015.

BGJP23. James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. End-to-end secure messaging with traceability only for illegal content. In *EUROCRYPT 2023*, pages 35–66. Springer, 2023.

BS20. Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

CD98. Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *CRYPTO 1998*, pages 424–441. Springer, 1998.

CL06. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO 2006*, pages 78–96. Springer, 2006.

CP92. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO 1992*, pages 89–105. Springer, 1992.

CS97. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO 1997*, pages 410–424. Springer, 1997.

CT18. Long Chen and Qiang Tang. People who live in glass houses should not throw stones: Targeted opening message franking schemes. Cryptology ePrint Archive, Report 2018/994, 2018.

DB96. Dorothy E Denning and Dennis K Branstad. A taxonomy for key escrow encryption systems. *Commun. ACM*, 39(3):34–40, 1996.

DGRW18. Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In *CRYPTO 2018*, pages 155–186. Springer, 2018.

Fac16a. Facebook. Facebook messenger app. 2016. https://www.messenger.com/.

Fac16b. Facebook. Messenger secret conversations technical whitepaper. 2016. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf.

FBI. FBI. Going dark. https://www.fbi.gov/services/operational-technology/going-dark, accessed in January 2024.

FKMQ+23. Valerie Fetzer, Michael Klooß, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. Universally composable auditable surveillance. In *ASIACRYPT 2023*, pages 453–487. Springer, 2023.

FKP23. Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT 2023*, pages 166–200. Springer, 2023.

FPS+18. Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel Weitzner. Practical accountability of secret processes. In *USENIX Security 2018*, pages 657–674, 2018.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, pages 186–194. Springer, 1986.

GGHAK22. Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking Sigmas: A Framework to Compose Σ-Protocols for Disjunctions. In *EUROCRYPT 2022*, pages 458–487. Springer, 2022.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO 2013*, pages 536–553. Springer, 2013.

GKVL21. Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems. In *EUROCRYPT 2021*, pages 553–583. Springer, 2021.

GLR17. Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *CRYPTO 2017*, pages 66–97. Springer, 2017.

GP17. Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist? A cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 99–110, 2017.

GQ88. Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *CRYPTO 1988*, pages 216–231. Springer, 1988.

HK07. Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO 2007*, pages 553–571. Springer, 2007.

IAV22. Rawane Issa, Nicolas Alhaddad, and Mayank Varia. Hecate: Abuse reporting in secure messengers with sealed sender. In *USENIX Security 2022*, pages 2335–2352, 2022.

KFB14. Joshua Kroll, Edward Felten, and Dan Boneh. Secure protocols for accountable warrant execution. *See http://www.cs.princeton.edu/felten/warrant-paper.pdf*, 2014.

Kho13. Megha Khosla. Balls into bins made faster. In *ESA 2013*, pages 601–612, 2013.

KL95. Joe Kilian and Tom Leighton. Fair cryptosystems, revisited: A rigorous approach to key-escrow. In *CRYPTO 1995*, pages 208–221. Springer, 1995.

KMW09. Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.

KZW+18. Joshua A Kroll, Joe Zimmerman, David J Wu, Valeria Nikolaenko, Edward W Felten, and Dan Boneh. Accountable cryptographic access control. In *Workshop, CRYPTO*, 2018.

LRC14. Jia Liu, Mark D Ryan, and Liqun Chen. Balancing societal security and individual privacy: Accountable escrow system. In *CSF 2014*, pages 427–440. IEEE, 2014.

LV18. Iraklis Leontiadis and Serge Vaudenay. Private message franking with after opening privacy. Cryptology ePrint Archive, Report 2018/938, 2018. https://eprint.iacr.org/2018/938.

LZH+23. Junzuo Lai, Gongxian Zeng, Zhengan Huang, Siu Ming Yiu, Xin Mu, and Jian Weng. Asymmetric group message franking: Definitions and constructions. In *EUROCRYPT 2023*, pages 67–97. Springer, 2023.

Oka95. Tatsuaki Okamoto. An efficient divisible electronic cash scheme. In *CRYPTO 1995*, pages 438–451. Springer, 1995.

PEB21. Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. In *CCS 2021*, page 1484–1506, 2021.

PR04. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

PVMB19. Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Austin Bos. Sampl: Scalable auditability of monitoring processes using public ledgers. In *CCS 2019*, pages 2249–2266, 2019.

Sav18. Stefan Savage. Lawful device access without mass surveillance risk: A technical design discussion. In *CCS 2018*, pages 1761–1774, 2018.

Sca19. Alessandra Scafuro. Break-glass encryption. In *PKC 2019*, pages 34–62. Springer, 2019.

Sch89. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO 1989*, pages 239–252. Springer, 1989.

Sha07. Hovav Shacham. A Cramer-Shoup Encryption Scheme from the Linear Assumption and from Progressively Weaker Linear Variants. Cryptology ePrint Archive, Report 2007/074, 2007.

SSW19. Sacha Servan-Schreiber and Archer Wheeler. Judge, jury & encryptioner: Exceptional access with a fixed social cost. *arXiv preprint arXiv:1912.05620*, 2019.

Tar18. Jamie Tarabay. Australian government passes contentious encryption law. *The New York Times*, 2018.

TGL+19. Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In *CRYPTO 2019*, pages 222–250. Springer, 2019.

TMR19. Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In *CCS 2019*, pages 413–430, 2019.

Wal22. Stefan Walzer. Insertion time of random walk cuckoo hashing below the peeling threshold. In *ESA 2022*, pages 87:1–87:11, 2022.

WV18. Charles Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *EuroS&P 2018*, pages 288–306. IEEE, 2018.

Yeo23. Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In *CRYPTO 2023*, pages 197–230, 2023.

YY98. Adam Young and Moti Yung. Auto-recoverable auto-certifiable cryptosystems. In *EUROCRYPT 1998*, pages 17–31. Springer, 1998.

Appendix

A Other related works

Here, we revisit some other related works.

Message franking. The concept of message franking was initially introduced by Facebook [Fac16a,Fac16b]. They are two kinds of message franking. One is symmetric message franking (SMF) [GLR17,DGRW18,LV18,CT18], using symmetric-key encryption. As a result, the moderator is the platform. Another one is asymmetric message franking (AMF) [TGL+19], which is firstly formalized in [TGL+19] for the E2E scenarios. Then Lai et al. proposed a definition and construction of asymmetric group message franking (AGMF) [LZH+23] based on HPS-KEM$^{\Sigma}$. However, these schemes predominantly emphasize source tracing and do not explicitly consider abuse resistance and retrospective content moderation.

Traceback Systems. There are some works [TMR19,PEB21,IAV22], extending the reach of message franking, focusing on "traceback" mechanisms. For example, the work [TMR19] allows to reveal a chain of forwarded messages (path traceback) or the entire forwarding tree (tree traceback).

Set pre-constrained encryption. In [AJJM22], Ananth et al. introduce the concept of *pre-constrained encryption (PCE)*, where the owner of the master secret key does not possess "full" decryption power. Instead, its decryption capabilities are constrained in a pre-specified manner during the system setup. In [BGJP23], Bartusek et al. aim to constrain the decryption power with respect to a set, thereby defining and constructing set pre-constrained encryption (SPCE) schemes. Their construction builds upon the recent Apple PSI protocol [BBM+21]. However, as discussed earlier, their SPCE definition cannot meet the requirements of our MAMF.

Key escrow and accountable access. Since the 1990s, various papers [KL95,YY98,DB96], have explored different forms of key escrow mechanisms in diverse domains. However, such schemes are vulnerable to potential abuse and mass surveillance. In 2018, [Sav18] addressed lawful device access while safeguarding against mass surveillance through physical means. Yet, extending these countermeasures to messaging or telephony software presents notable challenges. Over the last decade, academic attention also focuses on accountability or auditable logs for surveillance actions [LRC14,BBS+15,KFB14,KZW+18,GP17,FPS+18,WV18,SSW19,PVMB19,FKMQ+23] For example, Frankle et al. [FPS+18] employed ledgers for accountability in search procedures, though the method cannot be applied to E2E setting. There are also some unconventional proposals [WV18,Sca19]. Wright and Varia [WV18] proposed a construction utilizing cryptographic puzzles for limited access, but with a high monetary cost.

B Preliminaries: Cryptographic assumptions, NIZK, Sigma protocols, cuckoo hashing and SPCE

B.1 Cryptographic assumptions

Let $\mathbb{G}$ be a cyclic group of prime order p and g be the generator of $\mathbb{G}$. Let $(\mathbb{G}_1, \mathbb{G}_T)$ be a bilinear groups of prime order p with an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, and let h be a generator of $\mathbb{G}_1$. Let GenG_1 be a

group generation algorithm for common cyclic groups, which takes the security parameter as input and outputs the group description $(\mathbb{G}, g, p)$. Let GenG_2 be a group generation algorithm for bilinear maps, which takes the security parameter as input and outputs the group description $(e, \mathbb{G}_1, \mathbb{G}_T, g, p)$.

Definition 26. (The DL assumption). *We say that the* discrete logarithm (DL) assumption *holds for* $\mathbb{G}$*, if for any PPT adversary* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},\mathcal{A}}(\lambda) = 1]$ *is negligible, where* $\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},\mathcal{A}}(\lambda)$ *is shown in Fig. 9.*

Definition 27. (The DDH assumption). *We say that the* decisional Diffie-Hellman (DDH) assumption *holds for* $\mathbb{G}$*, if for any PPT adversary* $\mathcal{D}$*,* $\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{D}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{D}}(\lambda) = 1] - \frac{1}{2}|$ *is negligible, where* $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{D}}(\lambda)$ *is in Fig. 9.*

Definition 28. (The DBDH assumption). *We say that the* decisional Bilinear Diffie-Hellman (DBDH) assumption *holds for* $(\mathbb{G}_1, \mathbb{G}_T)$*, if for any PPT adversary* $\mathcal{D}$*,* $\mathbf{Adv}^{\mathrm{dbdh}}_{\mathbb{G}_1,\mathbb{G}_T}(\mathcal{D}) := |\Pr[\mathbf{G}^{\mathrm{dbdh}}_{\mathbb{G}_1,\mathbb{G}_T,\mathcal{D}}(\lambda) = 1] - \frac{1}{2}|$ *is negligible, where* $\mathbf{G}^{\mathrm{dbdh}}_{\mathbb{G}_1,\mathbb{G}_T,\mathcal{D}}(\lambda)$ *is in Fig. 9.*

$\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},\mathcal{A}}(\lambda)$:	$\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{D}}(\lambda)$:	$\mathbf{G}^{\mathrm{dbdh}}_{\mathbb{G}_1,\mathbb{G}_T,\mathcal{D}}(\lambda)$:
$(\mathbb{G}, g, p) \leftarrow \mathsf{GenG}_1(\lambda)$	$(\mathbb{G}, g, p) \leftarrow \mathsf{GenG}_1(\lambda)$	$(e, \mathbb{G}, \mathbb{G}_T, g, p) \leftarrow \mathsf{GenG}_2(\lambda)$
$x \leftarrow \mathbb{Z}_p^*$	$c \leftarrow \{0,1\}$	$d \leftarrow \{0,1\}$
$x' \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^x)$	$(a, b) \leftarrow (\mathbb{Z}_p^*)^2$	$(a, b, c) \leftarrow (\mathbb{Z}_p^*)^3$
Return $(x = x')$	If $c = 1$: $Z = g^{ab}$	If $d = 1$: $r = e(h, h)^{abc}$
	Else: $Z \leftarrow \mathbb{G}$	Else: $r \leftarrow \mathbb{G}_T$
	$c' \leftarrow \mathcal{D}(\mathbb{G}, p, g, g^a, g^b, Z)$	$d' \leftarrow \mathcal{D}(e, \mathbb{G}_1, \mathbb{G}_T, p, h, h^a, h^b, h^c, r)$
	Return $(c = c')$	Return $(d = d')$

Fig. 9 Games for the DL, DDH and DBDH assumptions

B.2 NIZK and Sigma protocols

Now we recall the definitions of non-interactive zero knowledge (NIZK) proof system *in the random oracle model*, Sigma protocol, and the Fiat-Shamir heuristic [FS86] as follows. For convenience, the recalled NIZK is a variant integrating the notion of signature of knowledge in [CS97,CL06,TGL+19] and the notion of NIZK in [BS20].

NIZK proof system. Let $\mathcal{M}$ be a message space. For a witness space $\mathcal{X}$ and a statement space $\mathcal{Y}$, let $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$ be a relation. A NIZK proof scheme $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ for witness-statement relation $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$ is a pair of PPT algorithms associated with a message space $\mathcal{M}$ and a proof space Π.

- $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(m, y, x)$: The prove algorithm takes $(m, y, x) \in \mathcal{M} \times \mathcal{Y} \times \mathcal{X}$ as input, and outputs a proof $\pi \in \Pi$.
- $b \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(m, \pi, y)$: The verification algorithm takes $(m, \pi, y) \in \mathcal{M} \times \Pi \times \mathcal{Y}$ as input, and outputs a bit $b \in \{0, 1\}$.

It is required to satisfies *completeness, existential soundness, and zero-knowledge in the random oracle model*. The formal definitions are recalled as follows.

- **Completeness.** For all $m \in \mathcal{M}$ and all $(y, x) \in \mathcal{R}$, we always have $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(m, \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(m, y, x), y) = 1$.
- **Existential soundness.** For any PPT adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda)$ is negligible, where $\mathbf{Adv}^{\mathrm{sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda)$ is the probability that $\mathcal{A}$ outputs $(m, y) \in \mathcal{M} \times \mathcal{Y}$ and $\pi \in \Pi$, such that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(m, \pi, y) = 1$ and $(x', y) \notin \mathcal{R}$ for all $x' \in \mathcal{X}$.
- **Zero-knowledge.** There is a PPT simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{Prove}}, \mathcal{S}_{\mathsf{ro}})$, such that for any PPT adversary $\mathcal{A}$, the advantage

$$\mathbf{Adv}^{\mathrm{zk}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{G}^{\mathrm{real}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{G}^{\mathrm{ideal}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) = 1]\right|$$

  is negligible, where $\mathbf{G}^{\mathrm{real}}_{\mathsf{NIZK},\mathcal{A}}$ and $\mathbf{G}^{\mathrm{ideal}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}$ are both in Fig. 10. Suppose that $\mathsf{NIZK}^{\mathcal{R}}$ makes use of a hash function Hash, and the hash function Hash with output length len in Fig. 10 is modeled as a random oracle (a local array H is employed).

$\mathbf{G}^{\mathrm{real}}_{\mathsf{NIZK},\mathcal{A}}(\lambda)$:	$\mathbf{G}^{\mathrm{ideal}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda)$:
$b \leftarrow \mathcal{A}^{\mathcal{O}}(\lambda)$	$b \leftarrow \mathcal{A}^{\mathcal{O}}(\lambda)$
Return b	Return b
$\mathcal{O}^{\mathsf{Prove}}(m, y, x)$:	$\mathcal{O}^{\mathsf{Prove}}(m, y, x)$:
If $(y, x) \notin \mathcal{R}$: Return $\perp$	If $(y, x) \notin \mathcal{R}$: Return $\perp$
$\pi \leftarrow \mathsf{Prove}(m, y, x)$	$(st, \pi) \leftarrow \mathcal{S}_{\mathsf{Prove}}(st, m, y)$
Return π	Return π
$\mathcal{O}^{\mathrm{ro}}(str)$:	$\mathcal{O}^{\mathrm{ro}}(str)$:
If $H[str] = \perp$:	$(st, r) \leftarrow \mathcal{S}_{\mathsf{ro}}(st, str)$
$r \leftarrow \{0,1\}^{len}$; $H[str] := r$	Return r
Return $H[str]$	

Fig. 10 Games for defining zero knowledge of $\mathsf{NIZK}^{\mathcal{R}}$

Sigma protocol. A Sigma protocol for $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$ consists of two efficient interactive protocol algorithms (P, V), where $P = (P_1, P_2)$ is the prover and $V = (V_1, V_2)$ is the verifier, associated with a challenge space $\mathcal{CL}$. Specifically, for any $(y, x) \in \mathcal{R}$, the input of the prover (resp., verifier) is (y, x) (resp., y). The prover first computes $(\mathsf{cm}, \mathsf{aux}) \leftarrow P_1(y, x)$ and sends the commitment cm to the verifier. The verifier (i.e., V_1) returns a challenge $\mathsf{cl} \leftarrow \mathcal{CL}$. Then the prover replies with $\mathsf{z} \leftarrow P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux})$. Receiving z, the verifier (i.e., V_2) outputs $b \in \{0, 1\}$. The tuple $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ is called a *conversation*. We require that V does not make any random choices other than the selection of cl. For any

fixed $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$, if the final output of $V(y)$ is 1, $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ is called an *accepting conversation* for y. Correctness requires for all $(y, x) \in \mathcal{R}$, when $P(y, x)$ and $V(y)$ interact with each other, the final output of $V(y)$ is always 1.

The corresponding security notions are as follows.

Definition 29. (Knowledge soundness). *We say that a Sigma protocol (P, V) for $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$ provides* knowledge soundness*, if there is an efficient deterministic algorithm* Ext *such that on input $y \in \mathcal{Y}$ and two accepting conversations $(\mathsf{cm}, \mathsf{cl}, \mathsf{z}), (\mathsf{cm}, \mathsf{cl}', \mathsf{z}')$ where $\mathsf{cl} \neq \mathsf{cl}'$, Ext always outputs an $x \in \mathcal{X}$ satisfying $(y, x) \in \mathcal{R}$.*

Definition 30. (Special HVZK). *We say that a Sigma protocol (P, V) for $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$ with challenge space $\mathcal{CL}$ is* special honest verifier zero knowledge (special HVZK)*, if there is a PPT simulator $\mathcal{S}$ which takes $(y, \mathsf{cl}) \in \mathcal{Y} \times \mathcal{CL}$ as input and satisfies the following properties:*

(i) for all $(y, \mathsf{cl}) \in \mathcal{Y} \times \mathcal{CL}$, $\mathcal{S}$ always outputs a pair $(\mathsf{cm}, \mathsf{z})$ such that $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ is an accepting conversation for y;

(ii) for all $(y, x) \in \mathcal{R}$, the tuple $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$, generated via $\mathsf{cl} \leftarrow \mathcal{CL}$ and $(\mathsf{cm}, \mathsf{z}) \leftarrow \mathcal{S}(y, \mathsf{cl})$, has the same distribution as that of a transcript of a conversation between $P(y, x)$ and $V(y)$.

The Fiat-Shamir heuristic. Let $\mathcal{M}$ be a message space, and $(P, V) = ((P_1, P_2), (V_1, V_2))$ be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{Y} \times \mathcal{X}$, where its conversations $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ belong to some space $\mathcal{CM} \times \mathcal{CL} \times \mathcal{Z}$. Let $\mathsf{Hash} : \mathcal{M} \times \mathcal{CM} \rightarrow \mathcal{CL}$ be a hash function. The Fiat-Shamir non-interactive proof system $\mathsf{NIZK}_{\mathrm{FS}} = (\mathsf{Prove}_{\mathrm{FS}}, \mathsf{Verify}_{\mathrm{FS}})$, with proof space $\Pi = \mathcal{CM} \times \mathcal{Z}$, is as follows:

- $\mathsf{Prove}_{\mathrm{FS}}(m, y, x)$: On input $(m, y, x) \in \mathcal{M} \times \mathcal{Y} \times \mathcal{X}$, this algorithm firstly generates $(\mathsf{cm}, \mathsf{aux}) \leftarrow P_1(y, x)$ and $\mathsf{cl} = \mathsf{Hash}(m, \mathsf{cm}, y)$, and then computes $\mathsf{z} \leftarrow P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux})$. Finally, it outputs $\pi = (\mathsf{cm}, \mathsf{z})$.
- $\mathsf{Verify}_{\mathrm{FS}}(m, (\mathsf{cm}, \mathsf{z}), y)$: On input $(m, (\mathsf{cm}, \mathsf{z}), y) \in \mathcal{M} \times (\mathcal{CM} \times \mathcal{Z}) \times \mathcal{Y}$, this algorithm firstly computes $\mathsf{cl} = \mathsf{Hash}(m, \mathsf{cm}, y)$, and then runs $V_2(y)$ to check whether $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ is a valid conversation for y. If so, $\mathsf{Verify}_{\mathrm{FS}}$ returns 1; otherwise, it returns 0.

According to [FS86,BS20], $\mathsf{NIZK}_{\mathrm{FS}}$ is an NIZK proof system if Hash is modeled as a random oracle. To be noted, in order to reduce the size of π, we replace cm with cl (i.e., we have $\pi = (\mathsf{z}, \mathsf{cl})$), following [BS20].

B.3 A definition of cuckoo hashing [FKP23]

Cuckoo hashing (CH) [PR04] is a technique to store a set of m elements from a large universe $\mathcal{X}$ into a linear-size data structure that allows efficient memory accesses. Here, we define the notion of cuckoo hashing schemes, following [FKP23], which is also a variant of the one recently offered by Yeo [Yeo23]. The difference is that we use *deterministic* insert algorithm Insert instead.

In a nutshell, a cuckoo hashing scheme inserts n elements $x_1, \ldots, x_n \in \mathcal{X}$ in a vector T (with size $n' = \mathsf{poly}(n)$) so that each element x_i can be found exactly once in T, or in a stash set S. The efficient memory access comes from the fact that for a given x one can efficiently compute the k indices $i_1, \ldots, i_k$ such that $x \in \{T[i_1], \ldots, T[i_k]\} \cup S$. The idea of cuckoo hashing constructions is to sample k random hash functions $H_1, \ldots, H_k : \mathcal{X} \to [n']$ and use them to allocate x in one of the k indices $H_1(x), \ldots, H_k(x)$ (if the k positions are not available, then allocate x in the stash S). Each construction uses a specific algorithm to search the index allocated to x, requiring to move existing elements whenever a position is going to be allocated to another element. The most efficient algorithms are local search allocation [Kho13] and random walks [Wal22,Yeo23].

A cuckoo hashing scheme is defined as follows.

Definition 31. (Cuckoo hashing schemes). *A cuckoo hashing scheme* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Insert}, \mathsf{Lookup})$ *for a value space* $\mathcal{X}$ *consists of the following algorithms:*

- $\mathsf{Setup}(\lambda, n) \to (\mathsf{pp}, T, S)$*: It is a probabilistic algorithm that on input the security parameter* λ *and a bound* n *on the number of insertions, outputs a public parameter* pp *(indicating* $k \geq 2$ *hash functions), an empty vector* T *with* n' *entries (with* n' *a multiple of* k*), along with an empty stash set* S*, (denoting* $s \geq 0$ *its size, at this point,* $s = 0$*);*
- $\mathsf{Insert}(\mathsf{pp}, T, S, X = (x_1, \ldots, x_m)) \to (T', S')$*: It is a* deterministic *algorithm that on input vector* T *where each non-empty component contains an element in* $\mathcal{X} \in \mathsf{pp}$*, inserts each* $x_1, \ldots, x_m \in \mathcal{X}$ *in the vector exactly once and returns the updated vector with moved elements,* T'*,* S'*.*
- $\mathsf{Lookup}(\mathsf{pp}, x) \to (i_1, \ldots, i_k)$*: It is a* deterministic *algorithm that on input public parameter* pp *and* $x \in \mathcal{X}$*, returns* $(i_1, \ldots, i_k)$*, the candidate indices where* x *could be stored.*

Following the definition in [FKP23], here we also define the correctness of a cuckoo hashing scheme in a combined way. More exactly, we define the correctness of cuckoo hashing by looking at the probability that

- either the insertion algorithm Insert fails (i.e., the construction error probability in [Yeo23]),
- or if it does not fail, an inserted element is not stored in the appropriate indices returned by Lookup (i.e., the query error probability in [Yeo23]).

Definition 32. (Correctness). *A cuckoo hashing scheme* CH *is* ϵ*-correct if for any* n*, any set of* $m \leq n$ *items* $x_1, \ldots, x_m \in \mathcal{X}$ *such that* $x_i \neq x_j$ *for all* $i \neq j$ *and any* $l \in [m]$*:*

$$\Pr\left[\begin{array}{l|c} (\mathsf{pp}, T, S) \leftarrow \mathsf{Setup}(\lambda, n) & T' = \bot \\ (T', S') \leftarrow \mathsf{Insert}(\mathsf{pp}, T, S, X = (x_1, \ldots, x_m)) & : \quad \vee (T' \neq \bot \ \wedge \\ (i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x_l) & x_l \notin \{T'[i_1], \ldots, T'[i_k]\} \cup S') \end{array}\right] \leq \epsilon.$$

Robust cuckoo hashing. We also consider robust cuckoo hashing, which was introduced by Yeo [Yeo23]. In a nutshell, a PPT adversary is given the public

parameter pp (i.e., given the sampled hash functions) and aims to produce a set X that will fail to allocate. This models the scenario where an adversary has explicit access to the hash functions before choosing the set of elements.

Definition 33. (Robustness). *A cuckoo hashing scheme* CH *is* ϵ*-robust if for any* n*, any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{ll} (\mathsf{pp}, T, S) \leftarrow \mathsf{Setup}(\lambda, n) & \\ (X = (x_1, \ldots, x_m), l) \leftarrow \mathcal{A}(\mathsf{pp}) & T' = \bot \\ \quad \text{s.t. } \forall i \neq j \in [m] : x_i \neq x_j & : \quad \vee (T' \neq \bot \ \wedge \\ (T', S') \leftarrow \mathsf{Insert}(\mathsf{pp}, T, S, X) & x_l \notin \{T'[i_1], \ldots, T'[i_k]\} \cup S') \\ (i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x_l) & \end{array}\right] \leq \epsilon.$$

Efficiency parameters of cuckoo hashing. For our applications, the following parameters will dictate the efficiency of a cuckoo hashing scheme:

- k, the number of possible indices (and of hash functions);
- n', the size of the table T; s, the size of the stash S;
- d, the number of changes in the table (i.e., number of evictions) after a single insertion.

While in most constructions, the parameters k and n' are fixed at Setup time, in some cuckoo hashing schemes the values of s and d may depend on the randomness and the choice of inputs. As in the case of correctness vs. robustness, we define s and d in the average case (i.e., for any set of inputs and for random and independent execution of Setup) or in the worst case (i..e, for adversarial choice of inputs after seeing pp).

Remark 5. As stated in [FKP23], the cuckoo hashing schemes defined here are, overall, probabilistic with the probability taken over the choice of pp. Once pp is fixed, everything is *deterministic*; Insert and Lookup, that take pp as input, are *deterministic* algorithms. The definition above differs from the one in [Yeo23] in the following aspects. Firstly, one can keep inserting elements, while [Yeo23] considers the static case in which the set is hashed all at once. Second, each entry of T here can store a single element, whereas in [Yeo23], they consider a more general case where it can store $l \geq 1$ elements, which occurs in some constructions.

Existing cuckoo hashing schemes. The following theorem encompasses a few existing cuckoo hashing schemes.

Theorem 4. *For a security parameter* λ *and an upper bound* n*, there exist the following cuckoo hashing schemes:*

- CH_2 *[KMW09] where* $k = 2$*,* $n' = 2kn$*, that achieves* $\mathsf{negl}(\lambda)$*-correctness, and average case* $s = \log n$*,* $d = O(1)$ *.*
- $\mathsf{CH}^{(\mathsf{rob})}$ *[KMW09, Yeo23] where* $k = 2$*,* $n' = 2kn$*, that achieves* $\mathsf{negl}(\lambda)$*-robustness, and worst case* $s = n$*,* $d = O(1)$ *in the random oracle model.*
- $\mathsf{CH}^{(\mathsf{rob})}_{\lambda}$ *[Yeo23] where* $k = \lambda$*,* $n' = 2\lambda n$*, that achieves* $\mathsf{negl}(\lambda)$*-robustness, and worst case* $s = 0$*,* $d = \lambda$ *in the random oracle model.*

B.4 Set pre-constrained encryption

Let's recall the primitive of *set pre-constrained encryption (SPCE)*, following the definition in [BGJP23].

Let $\mathcal{U}$ denote a universe of elements, $\mathcal{M}$ denote a message space. The set pre-constrained encryption USPCE contains three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and details are as follows.

1. $(pk, sk) \leftarrow \mathsf{Gen}(\lambda, \mathsf{S})$: the parameter generation algorithm takes as input a security parameter λ and a set $\mathsf{S} \in \mathcal{U}$ of size at most n, and outputs a public key pk and a secret key sk.
2. $\mathsf{ct} \leftarrow \mathsf{Enc}(pk, x, m)$: the encryption algorithm takes as input a public key pk, an item $x \in \mathcal{U}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext ct.
3. $m/\perp \leftarrow \mathsf{Dec}(sk, \mathsf{ct})$: the decryption algorithm takes as input a secret key sk and a ciphertext ct, and outputs either a message m or a bot symbol $\perp$.

It also satisfies the following properties.

Definition 34. (Correctness). *An set pre-constrained encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* correct*, if for any* $\lambda \in \mathbb{N}$ *and* $\mathsf{S} \subset \mathcal{U}$*, it holds that with overwhelming probability, for any* $x \in \mathsf{S}$ *and* $m \in \mathcal{M}$*:*

$$\Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(\lambda, \mathsf{S}) \ : \ \mathsf{Dec}(sk, \mathsf{Enc}(pk, x, m)) = m\right] = 1 - \mathsf{negl}(\lambda).$$

We say the scheme is perfectly correct*, if for any* $\lambda \in \mathbb{N}$*,* $\mathsf{S} \subset \mathcal{U}$*, and for any* $m \in \mathcal{M}$*, it hold that*

- *for every* $x \in \mathsf{S}$*,*

$$\Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(\lambda, \mathsf{S}) \ : \ \mathsf{Dec}(sk, \mathsf{Enc}(pk, x, m)) = m\right] = 1.$$

- *for every* $x \notin \mathsf{S}$*,*

$$\Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(\lambda, \mathsf{S}) \ : \ \mathsf{Dec}(sk, \mathsf{Enc}(pk, x, m)) \in \{m, \perp\}\right] = 1.$$

For other security properties, please refer to [BGJP23].

C Proof of Theorem 1

C.1 Proof of confidentiality against authority

Proof. We use a sequence of games to show that USPCE satisfies confidentiality against authority.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$.

Game $\mathbf{G}_\varrho$ $(1 \leq \varrho \leq k)$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

- Pick $b \in \{0, 1\}$ randomly.

– For $1 \leq j \leq \varrho$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– For $\varrho < j \leq k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j := e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
– Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c := (g^r, e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

Game $\mathbf{G}_{k+1}$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

– For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– Choose $c_1 \leftarrow \mathbb{G},\ c_2 \leftarrow \mathbb{G}_T$ randomly, and set $c := (c_1, c_2)$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

We prove these games are indistinguishable in the following lemmas. It is clear that the adversary has no advantage in **Game $\mathbf{G}_{k+1}$**. Therefore, we conclude that the advantage of the adversary in $\mathbf{G}^{\text{conf-au}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ is negligible.

Lemma 1. *If the hash function $\widetilde{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then for $1 \leq \varrho \leq k$, **Game $\mathbf{G}_{\varrho-1}$** and **Game $\mathbf{G}_{\varrho}$** are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game $\mathbf{G}_{\varrho-1}$** and **Game $\mathbf{G}_{\varrho}$** with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game $\mathbf{G}_{\varrho-1}$** or **Game $\mathbf{G}_{\varrho}$** as follows.

$\mathcal{B}$ first chooses a hash function $\overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. $\mathcal{B}$ also maintains a list $L_{\widetilde{\mathsf{H}}}$, where the list is empty initially. Then, $\mathcal{B}$ computes $A' = g^{\alpha'}$ and $Y' = g^s$, where $\alpha', s \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, $\mathcal{B}$ chooses $t_i \leftarrow \mathbb{Z}_p^*$ and computes $\widetilde{T}[i] = (g^{z_2})^{t_i}$. Note that, if $T_{\mathsf{S}}[i] \neq \perp$, $\mathcal{B}$ sets $\widetilde{\mathsf{H}}(T_{\mathsf{S}}[i]) = (g^{z_2})^{t_i}$ implicitly and adds the record $(T_{\mathsf{S}}[i], t_i, (g^{z_2})^{t_i})$ to the list $L_{\widetilde{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\widetilde{\mathsf{H}}$. Next, $\mathcal{B}$ sets $\alpha = z_1$ implicitly, and computes $X = g^{\beta}, Y = (Y')^{\beta}$, where $\beta \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, it computes $T[i] = e(g^{z_1}, (g^{z_2})^{t_i})^{\alpha'}$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \overline{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}})$, $msk = (\widetilde{T}, \mathsf{S}, s)$, $pk = (T, X, Y)$ to the adversary $\mathcal{A}$.

When $\mathcal{A}$ issues a random oracle $\widetilde{\mathsf{H}}$ query on x, $\mathcal{B}$ answers as follows. If there is some $(x, t \in \mathbb{Z}_p^*, h \in \mathbb{G}) \in L_{\widetilde{\mathsf{H}}}$, $\mathcal{B}$ returns h; otherwise, $\mathcal{B}$ samples $t_x \leftarrow \mathbb{Z}_p^*$, adds (x, t_x, g^{t_x}) to $L_{\widetilde{\mathsf{H}}}$, and returns g^{t_x}.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^*. $\mathcal{B}$ picks $b \in \{0,1\}$ randomly and proceeds as follows.

– For $1 \leq j \leq \varrho - 1$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.

– For $j = \varrho$, find some (x^*, t_{x^*}, h_{x^*}) in $L_{\widetilde{\mathsf{H}}}$, and set $\gamma_j = z_3$ implicitly. Then, let $i = \mathsf{H}_j(x^*)$ and compute $Q_j = e(g^{z_3}, h_{x^*})^{\alpha'} = e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j = Z^{\alpha' t_i} \cdot m_b$. Note that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $S_j = (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then S_j also is a random element of $\mathbb{G}_T$.
– For $\varrho < j \leq k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j := e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
– Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c := (g^r, e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

Observe that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho-1}$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$. Any non-negligible advantage of $\mathcal{A}$ is converted to a non-negligible advantage of $\mathcal{B}$.

Lemma 2. *If the hash function $\overline{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then **Game** $\boldsymbol{G}_k$ and **Game** $\boldsymbol{G}_{k+1}$ are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_{\varrho}$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_k$ or **Game** $\mathbf{G}_{k+1}$ as follows.

$\mathcal{B}$ first chooses a hash function $\widetilde{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. Then, $\mathcal{B}$ computes $A' = g^{\alpha'}$ and $Y' = g^s$, where $\alpha', s \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, if $T_{\mathsf{S}}[i] = \perp$, $\mathcal{B}$ chooses $\widetilde{T}[i] \leftarrow \mathbb{G}$; else $\mathcal{B}$ sets $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$. $\mathcal{B}$ also maintains a list $L_{\overline{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\overline{\mathsf{H}}$, where the list is empty initially. Next, $\mathcal{B}$ choose $\alpha \leftarrow \mathbb{Z}_p^*$ and computes $X = g^{z_1}, Y = (g^{z_1})^s$, where it sets $\beta = z_1$ implicitly. For each $i \in [n']$, it computes $T[i] = e(A', \widetilde{T}[i])^\alpha$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \widetilde{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}})$, $msk = (\widetilde{T}, \mathsf{S}, s)$, $pk = (T, X, Y)$ to the adversary $\mathcal{A}$.

When $\mathcal{A}$ issues a random oracle $\overline{\mathsf{H}}$ query on x, $\mathcal{B}$ answers as follows. If there is some $(x, t_x \in \mathbb{Z}_p^*, h_x \in \mathbb{G}) \in L_{\overline{\mathsf{H}}}$, $\mathcal{B}$ returns h_x; otherwise, $\mathcal{B}$ samples $t_x \leftarrow \mathbb{Z}_p^*$, adds $(x, t_x, h_x = (g^{z_2})^{t_x})$ to $L_{\overline{\mathsf{H}}}$, and returns h_x.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^*. $\mathcal{B}$ searches $L_{\overline{\mathsf{H}}}$ for a record $(x^*, t_{x^*}, h_{x^*} = g^{z_2})^{t_{x^*}}, coin)$. Then, $\mathcal{B}$ picks $b \in \{0,1\}$ randomly and proceeds as follows.

– For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– Compute $c = (c_1 = g^{z_3}, c_2 = Z^{t_{x^*}} \cdot m_b)$. Note that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $c_2 = e(\overline{\mathsf{H}}(x^*), Y)^{z_3} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then c_2 also is a random element of $\mathbb{G}_T$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

Observe that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_k$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{k+1}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$.

C.2 Proof of confidentiality against users

Proof. We use a sequence of games to show that USPCE satisfies confidentiality against users.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$.

Game $\mathbf{G}_\varrho (1 \le \varrho \le k)$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

- Pick $b \in \{0,1\}$ randomly.
- For $1 \le j \le \varrho$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
- For $\varrho < j \le k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j := e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
- Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c := (g^r, e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
- Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

Game $\mathbf{G}_{k+1}$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

- For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
- Choose $c_1 \leftarrow \mathbb{G}$, $c_2 \leftarrow \mathbb{G}_T$ randomly, and set $c := (c_1, c_2)$.
- Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

We prove these games are indistinguishable in the following lemmas. It is clear that the adversary has no advantage in **Game** $\mathbf{G}_{k+1}$. Therefore, we conclude that the advantage of the adversary in $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ is negligible.

Lemma 3. *If the hash function $\widetilde{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then for $1 \le \varrho \le k$, **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_\varrho$ are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_\varrho$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_{\varrho-1}$ or **Game** $\mathbf{G}_\varrho$ as follows.

$\mathcal{B}$ first chooses a hash function $\overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_\lambda.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_\lambda.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. $\mathcal{B}$ also maintains a list $L_{\widetilde{\mathsf{H}}}$, where the list is empty initially. Then, $\mathcal{B}$ sets $A' = g^{z_1}$ and computes $Y' = g^s$, where $s \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, $\mathcal{B}$ chooses $t_i \leftarrow \mathbb{Z}_p^*$ and computes $\widetilde{T}[i] = (g^{z_2})^{t_i}$. Note that, if $T_{\mathsf{S}}[i] \neq \perp$, $\mathcal{B}$ sets $\widetilde{\mathsf{H}}(T_{\mathsf{S}}[i]) = (g^{z_2})^{t_i}$ implicitly and adds the record

$(T_{\mathsf{S}}[i], t_i, (g^{z_2})^{t_i})$ to the list $L_{\widetilde{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\widetilde{\mathsf{H}}$. Next, $\mathcal{B}$ choose $\alpha, \beta \leftarrow \mathbb{Z}_p^*$ and computes $X = g^\beta, Y = (Y')^\beta$. For each $i \in [n']$, it computes $T[i] = e(A', (g^{z_2})^{t_i})^\alpha$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \overline{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y), sk = (\alpha, \beta)$ to the adversary $\mathcal{A}$.

$\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as follows:

- $\widetilde{\mathsf{H}}$ query on x: If there is some $(x, t \in \mathbb{Z}_p^*, h \in \mathbb{G}) \in L_{\widetilde{\mathsf{H}}}$, $\mathcal{B}$ returns h; otherwise, $\mathcal{B}$ samples $t_x \leftarrow \mathbb{Z}_p^*$, adds (x, t_x, g^{t_x}) to $L_{\widetilde{\mathsf{H}}}$, and returns g^{t_x}.
- TKGen query on x: $\mathcal{B}$ returns $(\overline{\mathsf{H}}(x))^s$.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^* such that $x^* \notin \mathsf{S}$. $\mathcal{B}$ picks $b \in \{0, 1\}$ randomly and proceeds as follows.

- For $1 \leq j \leq \varrho - 1$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
- For $j = \varrho$, find some $(x^*, t_{x^*}, g^{t_{x^*}})$ in $L_{\widetilde{\mathsf{H}}}$, and set $\gamma_j = z_3$ implicitly. Then, let $i = \mathsf{H}_j(x^*)$ and compute $Q_j = e(A', g^{z_3})^{t_{x^*}} = e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j = Z^{\alpha t_i} \cdot m_b$. Note that, if $Z = e(g, g)^{z_1 z_2 z_3}$, then $S_j = (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then S_j also is a random element of $\mathbb{G}_T$.
- For $\varrho < j \leq k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j := e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
- Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c := (g^r, e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
- Return $\mathsf{ct} = ((Q_j, S_j)_{j \in [k]}, c)$.

Observe that, if $Z = e(g, g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho-1}$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_\varrho$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g, g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$. Any non-negligible advantage of $\mathcal{A}$ is converted to a non-negligible advantage of $\mathcal{B}$.

Lemma 4. *If the hash function $\overline{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then **Game** $\boldsymbol{G}_k$ and **Game** $\boldsymbol{G}_{k+1}$ are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_\varrho$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g, g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_k$ or **Game** $\mathbf{G}_{k+1}$ as follows.

$\mathcal{B}$ first chooses a hash function $\widetilde{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j \in [k]}$. Then, $\mathcal{B}$ computes $A' = g^{\alpha'}$ and sets $Y' = g^{z_1}$, where $\alpha' \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, if $T_{\mathsf{S}}[i] = \perp$, $\mathcal{B}$ chooses $\widetilde{T}[i] \leftarrow \mathbb{G}$; else $\mathcal{B}$ sets $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$. $\mathcal{B}$ also maintains a list $L_{\overline{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\overline{\mathsf{H}}$, where the list is empty initially. Next, $\mathcal{B}$ choose $\alpha, \beta \leftarrow \mathbb{Z}_p^*$ and computes $X = g^\beta, Y = (Y')^\beta$. For each $i \in [n']$, it computes $T[i] = e(A', \widetilde{T}[i])^\alpha$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \widetilde{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y), sk = (\alpha, \beta)$ to the adversary $\mathcal{A}$.

$\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as follows:

– $\overline{\mathsf{H}}$ query on x: If there is some $(x, t_x, h_x, coin) \in L_{\overline{\mathsf{H}}}$, $\mathcal{B}$ returns h_x; otherwise, $\mathcal{B}$ picks $coin \in \{0,1\}$ at random such that $\mathsf{Pr}[coin = 0] = \rho$. ($\rho$ will be determined later.) Then, randomly chooses $t_x \leftarrow \mathbb{Z}_p^*$. The record $(x, t_x, h_x = (g^{z_2})^{coin} \cdot g^{t_x}, coin)$ is added to $L_{\overline{\mathsf{H}}}$ and h_x is sent to $\mathcal{A}$.
– TKGen query on x: $\mathcal{B}$ searches $L_{\overline{\mathsf{H}}}$ for a record $(x, t_x, h_x, coin)$. If $coin = 1$, it aborts and terminates; otherwise $\mathcal{B}$ returns $(g^{z_1})^{t_x}$.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^* such that $x^* \notin \mathsf{S}$. $\mathcal{B}$ searches $L_{\overline{\mathsf{H}}}$ for a record $(x^*, t_{x^*}, h_{x^*}, coin)$. If $coin = 0$, it aborts and terminates; otherwise $\mathcal{B}$ picks $b \in \{0,1\}$ randomly and proceeds as follows.

– For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– Compute $c = (c_1 = g^{z_3}, c_2 = Z \cdot e(g^{z_3}, g^{z_1})^{\beta t_{x^*}} \cdot m_b)$. Note that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $c_2 = e(\overline{\mathsf{H}}(x^*), Y)^{z_3} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then c_2 also is a random element of $\mathbb{G}_T$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c)$.

The probability that $\mathcal{B}$ does not abort during the simulation is given by $\rho^{q_{tken}}(1-\rho)$ which is maximized at $\rho = 1 - 1/(q_{tken}+1)$, where q_{tken} denotes the number of TKGen queries by the adversary $\mathcal{A}$. Now, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_k$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{k+1}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$.

C.3 Proof of confidentiality of sets

Proof. We use a sequence of games to show that USPCE supports confidentiality of sets.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)$. Specifically, when the adversary submits two sets $\mathsf{S}_0, \mathsf{S}_1$, the challenger picks $b \in \{0,1\}$ randomly and proceeds as follows.

– Run $(e, \mathbb{G}, \mathbb{G}_T, g, p) \leftarrow \mathsf{GenG}(\lambda)$.
– Choose hash functions $\widetilde{\mathsf{H}}, \overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$.
– Run $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S}_b)$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$.
– Choose $\alpha', s \leftarrow \mathbb{Z}_p^*$ and compute $A' = g^{\alpha'}$, $Y' = g^s$.
– For each $i \in [n']$, if $T_{\mathsf{S}}[i] = \perp$, choose $\widetilde{T}[i] \leftarrow \mathbb{G}$; else set $\widetilde{T}[i] = \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$.
– Choose $\alpha, \beta \leftarrow \mathbb{Z}_p^*$ and compute $X = g^{\beta}, Y = (Y')^{\beta}$.
– For each $i \in [n']$, set $T[i] = e(g, \widetilde{T}[i])^{\alpha'\alpha}$.
– Send $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \widetilde{\mathsf{H}}, \overline{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}}), pk = (T, X, Y)$ to the adversary.

Game $\mathbf{G}_\varrho (1 \le \varrho \le n')$: This game is the same as $\mathbf{G}_0$ except that the parameter T in the public key pk is generated as follows.

– For $1 \le j \le \varrho$, choose $T[j] \leftarrow \mathbb{G}_T$ randomly.

– For $\varrho < j \leq n'$, set $T[j] = e(g, \widetilde{T}[i])^{\alpha'\alpha}$.

We prove these games are indistinguishable in the following lemma. It is clear that the adversary has no advantage in **Game** $\mathbf{G}_{n'}$. Therefore, we conclude that the advantage of the adversary in $\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{A}}(\lambda)$ is negligible.

Lemma 5. *If the hash function $\widetilde{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then for $1 \leq \varrho \leq n'$, **Game** $\boldsymbol{G}_{\varrho-1}$ and **Game** $\boldsymbol{G}_{\varrho}$ are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_{\varrho}$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_{\varrho-1}$ or **Game** $\mathbf{G}_{\varrho}$ as follows.

$\mathcal{B}$ first chooses a hash function $\overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}^{(\mathsf{rob})}_{\lambda}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S}_b)$, where $b \in \{0,1\}$ is chosen randomly, and $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. $\mathcal{B}$ also maintains a list $L_{\widetilde{\mathsf{H}}}$, where the list is empty initially. Then, $\mathcal{B}$ proceeds as follows.

- Set $A' = g^{z_1}$ and compute $Y' = g^s$, where $s \leftarrow \mathbb{Z}^*_p$. Note that, it sets $\alpha' = z_1$ implicitly.
- Set $\alpha = z_2$ implicitly.
- Choose $\beta \leftarrow \mathbb{Z}^*_p$ and compute $X = g^{\beta}, Y = (Y')^{\beta}$.
- For $1 \leq j \leq \varrho - 1$, choose $T[j] \leftarrow \mathbb{G}_T$ randomly.
- For $j = \varrho$, if $T_{\mathsf{S}}[j] = \perp$, choose $T[j] \leftarrow \mathbb{G}$; else set $T[j] = Z^{t_j}$, where $t_j \leftarrow \mathbb{Z}^*_p$. Note that, if $T_{\mathsf{S}}[j] \neq \perp$, set $\widetilde{\mathsf{H}}(T_{\mathsf{S}}[j]) = (g^{z_3})^{t_j}$ implicitly and add the record $(T_{\mathsf{S}}[j], t_j, (g^{z_3})^{t_j})$ to the list $L_{\widetilde{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\widetilde{\mathsf{H}}$. Observe that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $T_j = e(g, \widetilde{T}[i])^{\alpha'\alpha}$; otherwise Z is a random element of $\mathbb{G}_T$, then T_j also is a random element of $\mathbb{G}_T$.
- For $\varrho < j \leq k$, if $T_{\mathsf{S}}[j] = \perp$, choose $T[j] \leftarrow \mathbb{G}$; else set $T[j] = e(g^{z_1}, g^{z_2})^{t_j}$, where $t_j \leftarrow \mathbb{Z}^*_p$. Note that, if $T_{\mathsf{S}}[j] \neq \perp$, set $\widetilde{\mathsf{H}}(T_{\mathsf{S}}[j]) = g^{t_j}$ implicitly and add the record $(T_{\mathsf{S}}[j], t_j, g^{t_j})$ to the list $L_{\widetilde{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\widetilde{\mathsf{H}}$.
- Send $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, p, \overline{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y)$ to the adversary $\mathcal{A}$.

When $\mathcal{A}$ issues a random oracle $\widetilde{\mathsf{H}}$ query on x, $\mathcal{B}$ answers as follows. If there is some $(x, t \in \mathbb{Z}^*_p, h \in \mathbb{G}) \in L_{\widetilde{\mathsf{H}}}$, $\mathcal{B}$ returns h; otherwise, $\mathcal{B}$ samples $t_x \leftarrow \mathbb{Z}^*_p$, adds (x, t_x, g^{t_x}) to $L_{\widetilde{\mathsf{H}}}$, and returns g^{t_x}.

Observe that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho-1}$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$. Any non-negligible advantage of $\mathcal{A}$ is converted to a non-negligible advantage of $\mathcal{B}$.

D Proof of Theorem 2

D.1 Proof of universality

Proof. For any computationally unbounded adversary $\mathcal{A}$ attacking universality of dual HPS-KEM$^{\Sigma}$, let $(\mathsf{pp} = (\mathbb{G}, p, g_1, g_2), pk = g_1^{x_1} g_2^{x_2})$ be $\mathcal{A}$'s input, where $(pk, sk = (x_1, x_2))$ are generated by $\mathsf{KG}(\mathsf{pp})$. Denote by $a := \log_{g_1} g_2$. Let $(c = (u_1, u_2), k, r^* = (r, r'))$ be $\mathcal{A}$'s final output satisfying $((c, r^*) \in \mathcal{R}_{\mathrm{c}}^*) \wedge (\mathsf{CheckCwel}(\mathsf{pp}, c, r^*) = 0)$.

Note that $(c, r^*) \in \mathcal{R}_{\mathrm{c}}^*$ implies that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$. On the other hand, since $\mathcal{C}_{\mathsf{pp}}^{\text{well-f}} = \{(g_1^{\tilde{r}}, g_2^{\tilde{r}}) \mid \tilde{r} \in \mathbb{Z}_p^*\} = \{(g_1^{\tilde{r}}, g_1^{a\tilde{r}}) \mid \tilde{r} \in \mathbb{Z}_p^*\}$, we derive that $r' \neq ar$.

As a result,

$$\begin{aligned}\mathsf{Decap}(\mathsf{pp}, sk, c) = u_1^{x_1} u_2^{x_2} &= g_1^{rx_1} g_1^{r'x_2} = g_1^{r(x_1+ax_2)+r'x_2-rax_2} \\ &= (g_1^{x_1} g_2^{x_2})^r \cdot g_1^{(r'-ra)x_2} = pk^r \cdot g_1^{(r'-ra)x_2}.\end{aligned}$$

Notice that $sk = (x_1, x_2)$ is uniformly sampled from $(\mathbb{Z}_p^*)^2$, and the only information that $\mathcal{A}$ has about sk is $\log_{g_1} pk = x_1 + ax_2$. Thus, from $\mathcal{A}$'s point of view, given (pp, pk), x_2 is still uniformly distributed, which implies that $\mathsf{Decap}(\mathsf{pp}, sk, c) = pk^r \cdot g_1^{(r'-ra)x_2}$ is also uniformly distributed.

Hence, $\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = \Pr[k = \mathsf{Decap}(\mathsf{pp}, sk, c)]$ is negligible, concluding the proof of this theorem. □

D.2 Proof of extended universality

Proof. For any computationally unbounded adversary $\mathcal{A}$ attacking extended universality of dual HPS-KEM$^{\Sigma}$, let $(\mathsf{pp} = (\mathbb{G}, p, g_1, g_2), pk = g_1^{x_1} g_2^{x_2})$ be $\mathcal{A}$'s input, where $(pk, sk = (x_1, x_2))$ are generated by $\mathsf{KG}(\mathsf{pp})$. Denote by $a := \log_{g_1} g_2$. Let $(c = (u_1, u_2), k, r^* = (r, r'), t)$ be $\mathcal{A}$'s final output satisfying $((c, r^*) \in \mathcal{R}_{\mathrm{c}}^*) \wedge (\mathsf{CheckCwel}(\mathsf{pp}, c, r^*) = 0)$.

Note that $(c, r^*) \in \mathcal{R}_{\mathrm{c}}^*$ implies that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$. On the other hand, since $\mathcal{C}_{\mathsf{pp}}^{\text{well-f}} = \{(g_1^{\tilde{r}}, g_2^{\tilde{r}}) \mid \tilde{r} \in \mathbb{Z}_p^*\} = \{(g_1^{\tilde{r}}, g_1^{a\tilde{r}}) \mid \tilde{r} \in \mathbb{Z}_p^*\}$, we derive that $r' \neq ar$.

As a result,

$$\begin{aligned}\mathsf{dDecap}(\mathsf{pp}, sk, t, c) = u_1^{x_1} u_2^{x_2} \cdot t &= g_1^{rx_1} g_1^{r'x_2} \cdot t = g_1^{r(x_1+ax_2)+r'x_2-rax_2} \cdot t \\ &= (g_1^{x_1} g_2^{x_2})^r g_1^{(r'-ra)x_2} \cdot t = pk^r \cdot g_1^{(r'-ra)x_2} \cdot t.\end{aligned}$$

Notice that $sk = (x_1, x_2)$ is uniformly sampled from $(\mathbb{Z}_p^*)^2$, and the only information that $\mathcal{A}$ has about sk is $\log_{g_1} pk = x_1 + ax_2$. Thus, from $\mathcal{A}$'s point of view, given (pp, pk), x_2 is still uniformly distributed, which implies that $\mathsf{dDecap}(\mathsf{pp}, sk, t, c) = pk^r \cdot g_1^{(r'-ra)x_2} \cdot t$ is also uniformly distributed.

Hence, $\mathbf{Adv}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = \Pr[k = \mathsf{dDecap}(\mathsf{pp}, sk, t, c)]$ is negligible, concluding the proof of this theorem. □

D.3 Proof of smoothness

Proof. For any fixed $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$ and any fixed $pk = h$ generated by KG, let $a := \log_{g_1} g_2$, $b := \log_{g_1} h$. Then, $\mathcal{SK}_{pp,pk} = \{(x_1, x_2) \in (\mathbb{Z}_p^*)^2 \mid x_1 + ax_2 = b\}$.

Note that the ciphertext space of $\mathsf{Encap}_{\mathsf{c}}^*$ is $\mathcal{C}^* = (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2$, where $1_{\mathbb{G}}$ is the identity element of $\mathbb{G}$, and the encapsulated key space $\mathcal{K} = \mathbb{G}$. For all $\hat{c} \in (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2$, we parse $\hat{c} = (\hat{u}_1, \hat{u}_2)$, and write $S_1 := \{(\hat{u}_1, \hat{u}_2) \in (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2 \mid \log_{g_1} \hat{u}_2 \neq a \log_{g_1} \hat{u}_1\}$ and $S_2 := \{(\hat{u}_1, \hat{u}_2) \in (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2 \mid \log_{g_1} \hat{u}_2 = a \log_{g_1} \hat{u}_1\}$. So,

$$\begin{aligned}
\Delta((c,k),(c,k')) &= \frac{1}{2} \sum_{(\hat{c},\hat{k}) \in \mathcal{C}^* \times \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| \\
&= \frac{1}{2} \sum_{\hat{c} \in S_1} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| \\
&\quad + \frac{1}{2} \sum_{\hat{c} \in S_2} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]|. \quad (3)
\end{aligned}$$

We present the following two lemmas with postponed proofs.

Lemma 6. $\sum_{\hat{c} \in S_1} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| = 0.$

Lemma 7. $\sum_{\hat{c} \in S_2} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| = \frac{2}{p}.$

Combining Eq. (3), Lemma 6 and Lemma 7, we obtain $\Delta((c,k),(c,k')) = \frac{1}{p}$, concluding the proof of this theorem.

So what remains is to prove the above two lemmas.

Proof (of Lemma 6). For any $\hat{c} = (\hat{u}_1, \hat{u}_2) \in S_1$ and any $\hat{k} \in \mathcal{K} = \mathbb{G}$, we have $\Pr[(c,k) = (\hat{c},\hat{k})] = \frac{1}{(p-1)^2 p}$, and $\Pr[(c,k') = (\hat{c},\hat{k})] = \frac{1}{(p-1)^2} \Pr[k' = \hat{k} \mid c = \hat{c}]$.

Note that $c = (g_1^r, g_1^{r'}) = \hat{c}$ implies $r = \log_{g_1} \hat{u}_1$ and $r' = \log_{g_1} \hat{u}_2$. Since $\hat{c} \in S_1$, we obtain $r' \neq ar$. We also notice that $sk = (x_1, x_2)$ is uniformly sampled from $\mathcal{SK}_{pp,pk}$, so the distribution of sk can be seen as "uniformly sampling x_2 from $\mathbb{Z}_p^*$, and letting $x_1 = b - ax_2$". As a result, given a fixed $c = \hat{c}$ (i.e., given fixed $r = \log_{g_1} \hat{u}_1$ and $r' = \log_{g_1} \hat{u}_2$), when $sk \leftarrow \mathcal{SK}_{pp,pk}$, $k' = \mathsf{Decap}(\mathsf{pp}, sk, c) = g_1^{rx_1} g_1^{r'x_2} = g_1^{r(b-ax_2)+r'x_2} = h^r g_1^{(r'-ar)x_2}$ is uniformly distributed over $\mathcal{K}$. Hence, $\Pr[k' = \hat{k} \mid c = \hat{c}] = \frac{1}{p}$.

So we conclude that for any $\hat{c} \in S_1$ and any $\hat{k} \in \mathcal{K}$, $\Pr[(c,k') = (\hat{c},\hat{k})] = \frac{1}{(p-1)^2 p} = \Pr[(c,k) = (\hat{c},\hat{k})]$. □

Proof (of Lemma 7). For any $\hat{c} = (\hat{u}_1, \hat{u}_2) \in S_2$ and any $\hat{k} \in \mathcal{K} = \mathbb{G}$, we have $\Pr[(c,k) = (\hat{c},\hat{k})] = \frac{1}{(p-1)^2 p}$, and $\Pr[(c,k') = (\hat{c},\hat{k})] = \frac{1}{(p-1)^2} \Pr[k' = \hat{k} \mid c = \hat{c}]$.

Note that $c = (g_1^r, g_1^{r'}) = \hat{c}$ implies $r = \log_{g_1} \hat{u}_1$ and $r' = \log_{g_1} \hat{u}_2$. Since $\hat{c} \in S_2$, we obtain $r' = ar$. Thus, given a fixed $c = \hat{c}$ (i.e., given fixed $r = \log_{g_1} \hat{u}_1$ and $r' = \log_{g_1} \hat{u}_2$), we derive that $k' = \mathsf{Decap}(\mathsf{pp}, sk, c) = g_1^{rx_1} g_1^{r'x_2} =$

$g_1^{r(b-ax_2)+r'x_2} = h^r g_1^{(r'-ar)x_2} = h^r = h^{\log_{g_1} \hat{u}_1}$, which is also fixed (since $pk = h$ and $\hat{u}_1$ are both fixed values).

Hence,

$$\begin{aligned}
&\sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k)=(\hat{c},\hat{k})] - \Pr[(c,k')=(\hat{c},\hat{k})]| \\
&= \sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\frac{1}{(p-1)^2 p} - \frac{1}{(p-1)^2}\Pr[k'=\hat{k} \mid c=\hat{c}]| \\
&= \sum_{\hat{c}\in S_2}(\sum_{\hat{k}\neq h^{\log_{g_1}\hat{u}_1}} |\frac{1}{(p-1)^2 p} - 0| + |\frac{1}{(p-1)^2 p} - \frac{1}{(p-1)^2}\cdot 1|) \\
&= \sum_{\hat{c}\in S_2}((p-1)\frac{1}{(p-1)^2 p} + \frac{1}{(p-1)p}) = \sum_{\hat{c}\in S_2}\frac{2}{(p-1)p} = \frac{2}{p}.
\end{aligned}$$

□

Thus, we complete the proof. □

D.4 Proof of extended smoothness

Proof. For any fixed $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$ and any fixed $pk = h$ generated by KG, let $a := \log_{g_1} g_2$, $b := \log_{g_1} h$. Then, $\mathcal{SK}_{pp,pk} = \{(x_1, x_2) \in (\mathbb{Z}_p^*)^2 \mid x_1 + a x_2 = b\}$.

Note that the ciphertext space of $\mathsf{Encap}_{\mathrm{c}}^*$ is $\mathcal{C}^* = (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2$, where $1_{\mathbb{G}}$ is the identity element of $\mathbb{G}$, the encapsulated key space $\mathcal{K} = \mathbb{G}$, and the tag space $\mathcal{T} = \mathbb{G}$. For all $\hat{c} \in (\mathbb{G}\setminus\{1_{\mathbb{G}}\})^2$, we parse $\hat{c} = (\hat{u}_1, \hat{u}_2)$, and write $S_1 := \{(\hat{u}_1, \hat{u}_2) \in (\mathbb{G}\setminus\{1_{\mathbb{G}}\})^2 \mid \log_{g_1}\hat{u}_2 \neq a\log_{g_1}\hat{u}_1\}$ and $S_2 := \{(\hat{u}_1, \hat{u}_2) \in (\mathbb{G}\setminus\{1_{\mathbb{G}}\})^2 \mid \log_{g_1}\hat{u}_2 = a\log_{g_1}\hat{u}_1\}$. So,

$$\begin{aligned}
&\Delta((c,k,t),(c,k',t)) \\
&= \frac{1}{2}\sum_{(\hat{c},\hat{k},\hat{t})\in\mathcal{C}^*\times\mathcal{K}\times\mathcal{T}} |\Pr[(c,k,t)=(\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t)=(\hat{c},\hat{k},\hat{t})]| \\
&= \frac{1}{2}\sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_1}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k,t)=(\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t)=(\hat{c},\hat{k},\hat{t})]| \\
&\quad + \frac{1}{2}\sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k,t)=(\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t)=(\hat{c},\hat{k},\hat{t})]|. \qquad (4)
\end{aligned}$$

We present the following two lemmas with postponed proofs.

Lemma 8. $\sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_1}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k,t)=(\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t)=(\hat{c},\hat{k},\hat{t})]| = 0$.

Lemma 9. $\sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k,t)=(\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t)=(\hat{c},\hat{k},\hat{t})]| = \frac{2}{p}$.

Combining Eq. (4), Lemma 8 and Lemma 9, we obtain $\Delta((c,k,t),(c,k',t)) = \frac{1}{p}$, concluding the proof of this theorem.

So what remains is to prove the above two lemmas.

Proof (of Lemma 8). For any $\hat{c} = (\hat{u}_1, \hat{u}_2) \in S_1$, any $\hat{k} \in \mathcal{K} = \mathbb{G}$ and any $\hat{t} \in \mathcal{T} = \mathbb{G}$, we have $\Pr[(c,k,t) = (\hat{c},\hat{k},\hat{t})] = \frac{1}{(p-1)^2p^2}$, and $\Pr[(c,k',t) = (\hat{c},\hat{k},\hat{t})] = \frac{1}{(p-1)^2p}\Pr[k' = \hat{k} \mid (c = \hat{c}) \wedge (t = \hat{t})]$.

Note that $c = (g_1^r, g_1^{r'}) = \hat{c}$ implies $r = \log_{g_1}\hat{u}_1$ and $r' = \log_{g_1}\hat{u}_2$. Since $\hat{c} \in S_1$, we obtain $r' \neq ar$. We also notice that $sk = (x_1, x_2)$ is uniformly sampled from $\mathcal{SK}_{pp,pk}$, so the distribution of sk can be seen as "uniformly sampling x_2 from $\mathbb{Z}_p^*$, and letting $x_1 = b - ax_2$". As a result, given a fixed $c = \hat{c}$ (i.e., given fixed $r = \log_{g_1}\hat{u}_1$ and $r' = \log_{g_1}\hat{u}_2$) and a fixed $t = \hat{t}$, when $sk \leftarrow \mathcal{SK}_{pp,pk}$, $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, c) = g_1^{rx_1} g_1^{r'x_2} \cdot \hat{t} = g_1^{r(b-ax_2)+r'x_2} \cdot \hat{t} = h^r g_1^{(r'-ar)x_2} \cdot \hat{t}$ is uniformly distributed over $\mathcal{K}$. Hence, $\Pr[k' = \hat{k} \mid (c = \hat{c}) \wedge (t = \hat{t})] = \frac{1}{p}$.

So we conclude that for any $\hat{c} \in S_1$, any $\hat{k} \in \mathcal{K}$ and any $\hat{t} \in \mathcal{T}$, $\Pr[(c,k',t) = (\hat{c},\hat{k},\hat{t})] = \frac{1}{(p-1)^2p^2} = \Pr[(c,k,t) = (\hat{c},\hat{k},\hat{t})]$. □

Proof (of Lemma 9). For any $\hat{c} = (\hat{u}_1, \hat{u}_2) \in S_2$, any $\hat{k} \in \mathcal{K} = \mathbb{G}$ and any $\hat{t} \in \mathcal{T} = \mathbb{G}$, we have $\Pr[(c,k,t) = (\hat{c},\hat{k},\hat{t})] = \frac{1}{(p-1)^2p^2}$, and $\Pr[(c,k',t) = (\hat{c},\hat{k},\hat{t})] = \frac{1}{(p-1)^2p}\Pr[k' = \hat{k} \mid (c = \hat{c}) \wedge (t = \hat{t})]$.

Note that $c = (g_1^r, g_1^{r'}) = \hat{c}$ implies $r = \log_{g_1}\hat{u}_1$ and $r' = \log_{g_1}\hat{u}_2$. Since $\hat{c} \in S_2$, we obtain $r' = ar$. Thus, given a fixed $c = \hat{c}$ (i.e., given fixed $r = \log_{g_1}\hat{u}_1$ and $r' = \log_{g_1}\hat{u}_2$) and a fixed $t = \hat{t}$, we derive that $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, c) = g_1^{rx_1} g_1^{r'x_2} \cdot \hat{t} = g_1^{r(b-ax_2)+r'x_2} \cdot \hat{t} = h^r g_1^{(r'-ar)x_2} \cdot \hat{t} = h^r \cdot \hat{t} = h^{\log_{g_1}\hat{u}_1} \cdot \hat{t}$, which is also fixed (since $pk = h$, $\hat{u}_1$ and $\hat{t}$ are all fixed values).

Hence,

$$
\begin{aligned}
&\sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\Pr[(c,k,t) = (\hat{c},\hat{k},\hat{t})] - \Pr[(c,k',t) = (\hat{c},\hat{k},\hat{t})]| \\
&= \sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\sum_{\hat{k}\in\mathcal{K}} |\frac{1}{(p-1)^2p^2} - \frac{1}{(p-1)^2p}\Pr[k' = \hat{k} \mid (c = \hat{c}) \wedge (t = \hat{t})]| \\
&= \sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\Big(\sum_{\hat{k}\neq h^{\log_{g_1}\hat{u}_1}\cdot\hat{t}} |\frac{1}{(p-1)^2p^2} - 0| + |\frac{1}{(p-1)^2p^2} - \frac{1}{(p-1)^2p}\cdot 1|\Big) \\
&= \sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}((p-1)\frac{1}{(p-1)^2p^2} + \frac{1}{(p-1)p^2}) = \sum_{\hat{t}\in\mathcal{T}}\sum_{\hat{c}\in S_2}\frac{2}{(p-1)p^2} = \frac{2}{p}.
\end{aligned}
$$

□

Thus, we complete the proof. □

D.5 Proof of special extended smoothness

Proof. Note that the ciphertext space of $\mathsf{Encap}_{\mathsf{c}}^*$ is $\mathcal{C}^* = (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2$, where $1_{\mathbb{G}}$ is the identity element of $\mathbb{G}$, the encapsulated key space $\mathcal{K} = \mathbb{G}$, and the tag space $\mathcal{T} = \mathbb{G}$.

For any fixed $\hat{c} \in (\mathbb{G} \setminus \{1_{\mathbb{G}}\})^2$, we can parse $\hat{c} = (\hat{u}_1, \hat{u}_2)$ for some fixed values $\hat{u}_1, \hat{u}_2$. Thus, for uniformly sampled $t \leftarrow \mathcal{T}$, we obtain $\mathsf{dDecap}(\mathsf{pp}, sk, t, \hat{c}) = \hat{u}_1^{x_1}\hat{u}_2^{x_2} \cdot t$. In other words, when $t \leftarrow \mathcal{T}$, $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, \hat{c})$ is uniformly distributed over $\mathbb{G}$.

So we trivially obtain $\Delta((c, k), (c, k')) = 0$, where $c \leftarrow \mathsf{Encap}_{\mathsf{c}}^*(\mathsf{pp})$, $k \leftarrow \mathcal{K}$, $t \leftarrow \mathcal{T}$ and $k' = \mathsf{dDecap}(\mathsf{pp}, sk, t, c)$. □

D.6 Proof of ciphertext unexplainability

Proof. Assume that there is a PPT adversary $\mathcal{A}$ winning the game of ciphertext unexplainability with non-negligible probability. We show a PPT algorithm $\mathcal{B}$, making use of $\mathcal{A}$ to solve the DL problem with non-negligible probability, as follows.

Given $(\mathbb{G}, p, g, g^a)$, $\mathcal{B}$ first sets $g_1 = g$ and $g_2 = g^a$, and sends the public parameter $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$ to $\mathcal{A}$. Receiving $\mathcal{A}$'s output $(c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathsf{c}}^*$, $\mathcal{B}$ parses $c = (u_1, u_2)$ and $r_{\mathsf{c}}^* = (r, r')$. Note that $(c, r^*) \in \mathcal{R}_{\mathsf{c}}^*$ guarantees that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$. When $\mathcal{A}$ wins the game of ciphertext unexplainability, $c \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, which implies $u_1 = g_1^r$ and $u_2 = g_2^r$. In this case, we obtain $u_2 = g_2^r = g_1^{r'}$. Therefore, $\mathcal{B}$ can output $a = \frac{r'}{r}$ as the solution of the DL problem. □

D.7 Proof of key unexplainability

Proof. Assume that there is a PPT adversary $\mathcal{A}$ winning the game of key unexplainability with non-negligible probability. We show a PPT algorithm $\mathcal{B}$, making use of $\mathcal{A}$ to solve the DL problem with non-negligible probability, as follows.

Given $(\mathbb{G}, p, g, g^a)$, $\mathcal{B}$ firstly sets $g_1 = g$ and $g_2 = g^a$, samples $(x_1, x_2) \leftarrow \mathbb{Z}_p^*$, and sets $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$, $pk = g_1^{x_1}g_2^{x_2}$ and $sk = (x_1, x_2)$. Then, $\mathcal{B}$ sends (pp, pk, sk) to $\mathcal{A}$. Receiving $\mathcal{A}$'s output $(c, r_{\mathsf{c}}^*, k, r_{\mathsf{k}}^*)$, $\mathcal{B}$ parses $c = (u_1, u_2)$, $r_{\mathsf{c}}^* = (r, r')$ and $r_{\mathsf{k}}^* = (\widetilde{r}, \widetilde{r}')$. Note that $(c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathsf{c}}^*$ guarantees that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$, and $(k, r_{\mathsf{k}}^*) \in \mathcal{R}_{\mathsf{k}}^*$ guarantees that $k = g_1^{\widetilde{r}}g_2^{\widetilde{r}'} = g_1^{\widetilde{r}+a\widetilde{r}'}$. When $\mathcal{A}$ wins the game of key unexplainability, $\mathsf{Decap}(\mathsf{pp}, sk, c) = k$, which implies $k = u_1^{x_1}u_2^{x_2} = g_1^{rx_1+r'x_2}$. Hence, $g_1^{\widetilde{r}+a\widetilde{r}'} = g_1^{rx_1+r'x_2}$. $\mathcal{B}$ can output $a = \frac{rx_1+r'x_2-\widetilde{r}}{\widetilde{r}'}$ as the solution of the DL problem. □

D.8 Proof of extended key unexplainability

Proof. Assume that there is a PPT adversary $\mathcal{A}$ winning the game of extended key unexplainability with non-negligible probability. We show a PPT algorithm $\mathcal{B}$, making use of $\mathcal{A}$ to solve the DL problem with non-negligible probability, as follows.

Given $(\mathbb{G}, p, g, g^a)$, $\mathcal{B}$ firstly sets $g_1 = g$ and $g_2 = g^a$, samples $(x_1, x_2) \leftarrow \mathbb{Z}_p^*$, and sets $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$, $pk = g_1^{x_1} g_2^{x_2}$ and $sk = (x_1, x_2)$. Then, $\mathcal{B}$ sends (pp, pk, sk) to $\mathcal{A}$. Receiving $\mathcal{A}$'s output $(c, r_{\mathsf{c}}^*, k_{\mathsf{d}}, t, r_{\mathsf{k}}^*)$, $\mathcal{B}$ parses $c = (u_1, u_2)$, $r_{\mathsf{c}}^* = (r, r')$ and $r_{\mathsf{k}}^* = (\widetilde{r}, \widetilde{r}')$. Note that $(c, r_{\mathsf{c}}^*) \in \mathcal{R}_{\mathsf{c}}^*$ guarantees that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$, and $(k_{\mathsf{d}}, (t, r_{\mathsf{k}}^*)) \in \mathcal{R}_{\mathsf{k}}^{\mathsf{d}*}$ guarantees that $k_{\mathsf{d}} = g_1^{\widetilde{r}} g_2^{\widetilde{r}'} \cdot t = g_1^{\widetilde{r}+a\widetilde{r}'} \cdot t$. When $\mathcal{A}$ wins the game of extended key unexplainability, $\mathsf{dDecap}(\mathsf{pp}, sk, t, c) = k_{\mathsf{d}}$, which implies $k_{\mathsf{d}} = u_1^{x_1} u_2^{x_2} \cdot t = g_1^{rx_1 + r'x_2} \cdot t$. Hence, $g_1^{\widetilde{r}+a\widetilde{r}'} = g_1^{rx_1+r'x_2}$. $\mathcal{B}$ can output $a = \frac{rx_1 + r'x_2 - \widetilde{r}}{\widetilde{r}'}$ as the solution of the DL problem. □

D.9 Proof of indistinguishability

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ winning the game of indistinguishability with non-negligible probability. It is easy to construct a PPT algorithm $\mathcal{B}$ that makes use of $\mathcal{A}$ to solve the DDH problem with non-negligible probability. Algorithm $\mathcal{B}$ is given a random tuple $(\mathbb{G}, p, g, g^a, g^b, Z)$, where $Z = g^{ab}$ or Z is uniformly and independently sampled in $\mathbb{G}$. $\mathcal{B}$ runs $\mathcal{A}$ as follows.

$\mathcal{B}$ first sets $g_1 = g$, $g_2 = g^a$, $u_1 = g^b$, $u_2 = Z$. Then, it sends the public parameter $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$ and the encapsulated ciphertext $c = (u_1, u_2)$ to the adversary $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit and $\mathcal{B}$ also outputs the bit.

Observe that, if $Z = g^{ab}$, then $u_1 = g_1^b, u_2 = g_2^b$, and from the perspective of the adversary the distribution of the ciphertext $c = (u_1, u_2)$ is identical to the distribution of the well-formed encapsulated ciphertext generated by $\mathsf{Encap}_{\mathsf{c}}$. If Z is a random element in $\mathbb{G}$, then u_1, u_2 are random elements in $\mathbb{G}$, and from the perspective of the adversary the distribution of the ciphertext $c = (u_1, u_2)$ is identical to the distribution of the ciphertext generated by $\mathsf{Encap}_{\mathsf{c}}^*$. Therefore, if $\mathcal{A}$ can win the game of indistinguishability with non-negligible probability, $\mathcal{B}$ can make use of $\mathcal{A}$ to solve the DDH problem with non-negligible probability. □

D.10 Proof of SK-second-preimage resistance

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ winning the game of SK-second-preimage resistance with non-negligible probability. It is easy to construct a PPT algorithm $\mathcal{B}$ that makes use of $\mathcal{A}$ to solve the DL problem with non-negligible probability. Algorithm $\mathcal{B}$ is given a random tuple $(\mathbb{G}, p, g, g^a)$, and runs $\mathcal{A}$ as follows.

$\mathcal{B}$ first sets $g_1 = g$ and $g_2 = g^a$. Next, it chooses $x_1, x_2 \in \mathbb{Z}_p^*$ uniformly at random, and generates a pair of public/secret keys $(pk = g_1^{x_1} g_2^{x_2}, sk = (x_1, x_2))$. Then, $\mathcal{B}$ sends the public parameter $\mathsf{pp} = (\mathbb{G}, p, g_1, g_2)$ and the pair of public/secret keys (pk, sk) to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs a secret key $sk' = (x_1', x_2')$. If $\mathcal{A}$ wins the game of SK-second-preimage resistance, we have $sk' \neq sk$ and $\mathsf{CheckKey}(\mathsf{pp}, sk', pk) = 1$. That is to say, $g_1^{x_1'} g_2^{x_2'} = g_1^{x_1} g_2^{x_2}$ and $x_1' \neq x_1, x_2' \neq x_2$. Therefore, $\mathcal{B}$ can output $a = (x_1 - x_1')/(x_2' - x_2)$ as the solution of the DL problem. □

D.11 Proof of uniformity of sampled keys

Proof. It is evident that for any pp generated by $\mathsf{KEMSetup}$ and any $t \in \mathcal{T}$, both $k' \leftarrow \mathsf{SamEncK}(\mathsf{pp})$ and $k'' \leftarrow \mathsf{dSamEncK}(\mathsf{pp}, t)$ are uniformly distributed over $\mathcal{K}$. So $\Delta(k, k') = 0$ and $\Delta(k, k'') = 0$ where $k \leftarrow \mathcal{K}$. □

E Proof for Theorem 3

E.1 Proof of unforgeability

Proof. For any PPT adversary $\mathcal{A}$ attacking the unforgeability of MAMF, we denote $\mathcal{A}$'s input as $(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, sk_{\mathsf{J}})$, and $\mathcal{A}$'s final output as (m^*, σ^*). Then, we parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*)$. Let Q_{sig} denote the messages that $\mathcal{A}$ has submitted to $\mathcal{O}^{\mathsf{Frank}}$. Since $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ is a NIZK proof obtained via applying the Fiat-Shamir transform to Sigma protocols, we can further parse $\widehat{\pi} = (\widehat{\mathsf{cm}}, \widehat{\mathsf{z}})$. Note that $\mathcal{A}$ can query the random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$.

We present the following claim.

Claim. If $\mathcal{A}$ wins the game $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ during the generation of the responses to $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Frank}}$-oracle queries in $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Proof (of Claim). Assume that $\mathcal{A}$ wins the game $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, and $\mathcal{A}$'s final output satisfies $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y}) = (pk'_{\mathsf{r}}, \mathsf{cm}', y')$, where $(pk'_{\mathsf{r}}, \mathsf{cm}', y')$ is extracted from some signature $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}})$ output by the $\mathcal{O}^{\mathsf{Frank}}$ oracle (on some query (pk'_{r}, m') from $\mathcal{A}$).

Then, we have $pk_{\mathsf{r}} = pk'_{\mathsf{r}}$ and

$$(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*) = \widehat{y} = y' = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m'),$$

which implies

$$(pk_{\mathsf{r}}, m^*) = (pk'_{\mathsf{r}}, m') \in Q_{\mathrm{sig}}.$$

So in this case $\mathcal{A}$ cannot win the game $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, contradicting the assumption. □

Without loss of generality, we assume that (i) $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output (m^*, σ^*); and (ii) for each of $\mathcal{A}$'s verification query $(pk'_{\mathsf{s}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}))$ (parsing $\pi' = (\mathsf{cm}', \mathsf{z}')$ and letting $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m'))$, $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \mathsf{cm}', y')$ before submitting $(pk'_{\mathsf{s}}, m', \sigma')$ to $\mathcal{O}^{\mathsf{Verify}}$. This assumption holds without loss of generality, because if $\mathcal{A}$ does not make these queries, we can easily construct another adversary, based on $\mathcal{A}$, that makes these types of random-oracle queries.

Hence, in this case, if $\mathcal{A}$ wins the game $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ queries the random oracle on it.

Let evt denote the event that $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$ where $(pk_{\mathsf{r}}, m^*) \notin Q_{\mathrm{sig}}$. Then, we obtain

$$\mathbf{Adv}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{evt}].$$

If we can show that $\Pr[\mathsf{evt}]$ is negligible, then we finish this proof.

Assume that $\Pr[\mathsf{evt}]$ is non-negligible.

Note that $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$ implies that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{\pi}, \widehat{y}) = 1$. Since $\mathsf{NIZK}^{\mathcal{R}}$ is a NIZK proof system obtained via the Fiat-Shamir transform from a Sigma protocol, according to a rewinding lemma [BS20, Lemma 19.2] and knowledge soundness of the Sigma protocol, a witness $\widehat{x}$ for $\widehat{y}$ (satisfying $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$) can be extracted with non-negligible probability. The reason is as follows.

Let q_{ro} denote the total number of random oracle queries (in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$) made by $\mathcal{A}$. Since we assume that $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output (m^*, σ^*), for $j \in [q_{\mathrm{ro}}]$, let $\mathsf{evt}^{(j)}$ denote the event that evt occurs and $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ is $\mathcal{A}$'s j-th random oracle query. Obviously, $\Pr[\mathsf{evt}] = \sum_{j=1}^{q_{\mathrm{ro}}} \Pr[\mathsf{evt}^{(j)}]$. So the fact that $\Pr[\mathsf{evt}]$ is non-negligible implies that there must be some $j^* \in [q_{\mathrm{ro}}]$, such that $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible. On the other hand, when $\mathsf{evt}^{(j^*)}$ occurs, we can rewind back to the moment when $\mathcal{A}$ made its j^*-th random oracle query, and respond with a fresh and uniformly sampled value for this query (since the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ makes its j^*-th random oracle query). If $\mathsf{evt}^{(j^*)}$ occurs again, we can use the knowledge soundness of the Sigma protocol to extract a valid witness $\widehat{x}$ for $\widehat{y}$. Since $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible, the rewinding lemma [BS20, Lemma 19.2] guarantees that the witness can be extracted successfully with non-negligible probability.

Hence, let $\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}$ (resp., $\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}$) denote the event that evt occurs and a witness $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ (resp., $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$) for $\widehat{y}$ is successfully extracted.

Since $\Pr[\mathsf{evt}]$ is non-negligible, we derive that at least one of $\Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}]$ and $\Pr[\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

Case 1: If $\Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

We show a PPT adversary $\mathcal{B}$ attacking the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk}, \tilde{sk})$, $\mathcal{B}$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}\mathsf{KG}(\tilde{\mathsf{pp}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}\mathsf{KG}(\tilde{\mathsf{pp}})$. Then, $\mathcal{B}$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$, $sk_{\mathsf{J}} := (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$, and $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) := (\tilde{pk}, \tilde{sk})$. Then, with these parameters, $\mathcal{B}$ simulates $\mathbf{G}^{\mathrm{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that $\mathcal{B}$ can answer $\mathcal{A}$'s oracle queries by itself. Receiving $\mathcal{A}$'s final output (m^*, σ^*), if $\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}$ occurs, $\mathcal{B}$ returns $\widehat{sk_{\mathsf{s}}}$; otherwise, $\mathcal{B}$ returns a random secret key.

That is the construction of $\mathcal{B}$. Now we analyze $\mathcal{B}$'s advantage.

We note that $\mathcal{B}$ wins if and only if $\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}$ occurs and $\widehat{sk}_\mathsf{s} \neq sk_\mathsf{s}$, i.e.,

$$\begin{aligned}\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{B}}(\lambda) &= \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk}_\mathsf{s} \neq sk_\mathsf{s})] \\ &= \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk}_\mathsf{s} = sk_\mathsf{s})] \\ &\geq \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}].\end{aligned}$$

Next, we turn to analyze $\Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}]$. From $\mathcal{A}$'s point of view, the information on sk_s beyond pk_s is released only in the responses returned by $\mathcal{O}^{\mathsf{Frank}}$. $\mathcal{O}^{\mathsf{Frank}}$ will not provide any information on sk_s beyond pk_s except with negligible probability, because of the zero-knowledge property of $\mathsf{NIZK}^\mathcal{R}$ (note that in Frank, sk_s is only used as a component of the witness to generate the NIZK proof). Hence,

$$\Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] \leq \mathsf{negl}(\lambda).$$

So $\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{B}}(\lambda)$ is non-negligible, contradicting the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^\Sigma$.

Case 2: If $\Pr[\mathbf{evt}_{(\perp,\widehat{t},\perp,\widehat{r_\mathsf{c}^*},\widehat{r_\mathsf{k}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

We show the proof with a sequence of games.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$. Specifically, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$, $(pk_\mathsf{J}, sk_\mathsf{J})$, $(pk_\mathsf{s}, sk_\mathsf{s})$ and $(pk_\mathsf{r}, sk_\mathsf{r})$, and initializes a set $Q_{\text{sig}} := \emptyset$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^\mathcal{R}$). Subsequently, the challenger sends $(\mathsf{pp}, pk_\mathsf{s}, pk_\mathsf{r}, sk_{\mathsf{Ag}}, pk_\mathsf{J}, sk_\mathsf{J})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, the challenger returns cl to $\mathcal{A}$; otherwise, it samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_\mathsf{r}, m')}$: The challenger generates $\sigma' \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_\mathsf{s}, pk'_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m')$, sets $Q_{\text{sig}} := Q_{\text{sig}} \cup \{(pk'_\mathsf{r}, m')\}$, and returns σ' to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_\mathsf{s}, m', \sigma')}$: The challenger returns $\mathsf{Verify}(\mathsf{pp}, pk'_\mathsf{s}, sk_\mathsf{r}, pk_{\mathsf{Ag}}, pk_\mathsf{J}, m', \sigma')$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output (m^*, σ^*), the challenger checks whether $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r_\mathsf{c}^*},\widehat{r_\mathsf{k}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs. If so, the challenger outputs 1; otherwise, it outputs 0. In the following, we use $\mathbf{G}_i \Rightarrow 1$ to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ ($i \in \{0, 1, 2\}$).

Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_\mathsf{r}, \widehat{k}_\mathsf{J}, \widehat{c}_\mathsf{t})$. Note that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r_\mathsf{c}^*},\widehat{r_\mathsf{k}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\widehat{c} = \mathsf{dHPS\text{-}KEM}^\Sigma.\mathsf{Encap}^*_\mathsf{c}(\mathsf{pp}_{\mathsf{KEM}}; \widehat{r_\mathsf{c}^*})$.

Since $\mathbf{G}_0 = \mathbf{G}^{\text{unforge}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, we derive

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r_\mathsf{c}^*},\widehat{r_\mathsf{k}^*},\widehat{r_{\mathsf{USPCE}}})}]. \tag{5}$$

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs, the challenger returns 0 if $\widehat{c} \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$. Note that the challenger can use algorithm $\mathsf{CheckCwel}$ to check whether $\widehat{c} \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$ (with the help of $\widehat{r_{\mathsf{c}}^*}$) efficiently. The ciphertext unexplainability of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_0 \Rightarrow 1]| \leq \mathsf{negl}(\lambda). \tag{6}$$

Game $\mathbf{G}_2$: This game is the same as $\mathbf{G}_1$, except that when $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Verify}}$ on $(pk'_{\mathsf{s}}, m', \sigma')$, the challenger generates the response as follows:

(i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$.
(ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 0$, return 0 to $\mathcal{A}$.
(iii) Check whether $c' \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$ or not (with the help of some unbounded algorithm):
- If $c' \notin \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, return 0 to $\mathcal{A}$ directly.
- If $c' \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, find $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r') = c'$. Then, the challenger checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') = k'_{\mathsf{r}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.

We stress that $\mathbf{G}_2$ is an inefficient game.

Let bad denote the event that "$\mathcal{A}$ submits a verification query $(pk'_{\mathsf{s}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}))$ satisfying that (i) $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 1$ (where $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$), (ii) $c' \in \mathcal{C}^*_{pp} \backslash \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, and (iii) $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c') = k'_{\mathsf{r}}$." Note that from $\mathcal{A}$'s point of view, $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical except that bad occurs. The universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that the probability that bad occurs is negligible (note that when $c' \in \mathcal{C}^*_{pp} \setminus \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, an unbounded algorithm can trivially find r' satisfying $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r')$). Hence,

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad}] \leq \mathsf{negl}(\lambda). \tag{7}$$

Next, we show an unbounded adversary $\mathcal{B}'$, which simulates $\mathbf{G}_2$ for $\mathcal{A}$, attacking the universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Receiving $(\tilde{\mathsf{pp}}, \tilde{pk})$, $\mathcal{B}'$ initializes a set $Q_{\text{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$. Then, $\mathcal{B}'$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$, $sk_{\mathsf{J}} := (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$ and $pk_{\mathsf{r}} := \tilde{pk}$. $\mathcal{B}'$ maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). With these parameters, $\mathcal{B}'$ simulates $\mathbf{G}_2$ for $\mathcal{A}$, answering $\mathcal{A}$'s oracle queries as follows:

– $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, $\mathcal{B}'$ returns cl; otherwise, $\mathcal{B}'$ samples $\underline{\mathsf{cl} \leftarrow \mathcal{CL}}$, adds (str, cl) to L_{ro}, and returns cl.
– $\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathsf{r}}, m')}$: $\mathcal{B}'$ generates $\sigma' \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, sets $Q_{\text{sig}} := \underline{Q_{\text{sig}} \cup \{(pk'_{\mathsf{r}}, m')\}}$, and returns σ' to $\mathcal{A}$.

– $\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathsf{s}}, m', \sigma')$: $\mathcal{B}'$ generates the response as follows:
  (i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$.
  (ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 0$, $\mathcal{B}'$ returns 0 to $\mathcal{A}$.
  (iii) $\mathcal{B}'$ checks whether $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ or not (with the help of some unbounded algorithm):
    - If $c' \notin \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, $\mathcal{B}'$ returns 0 to $\mathcal{A}$ directly.
    - If $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, $\mathcal{B}'$ finds $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r') = c'$. Then, $\mathcal{B}'$ checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') = k'_{\mathsf{r}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output (m^*, σ^*), since $\mathcal{B}'$ cannot check whether evt occurs by itself (because it does not have sk_{r}), it proceeds as follows.

(1) If $(pk_{\mathsf{r}}, m^*) \in Q_{\text{sig}}$, then $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output.
(2) Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}}, m^*)$.
(3) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{y}, \widehat{\pi}) = 0$, $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output.
(4) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{y}, \widehat{\pi}) = 1$, extract a witness $\widehat{x}$ for $\widehat{y}$ (via the rewinding technique) such that $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$:
  - If $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output.
  - If $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}'$ firstly checks whether $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ or not. If so, $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output. Otherwise, $\mathcal{B}'$ returns $(\widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{r^*_{\mathsf{c}}})$ as its final output.

That is the construction of $\mathcal{B}'$.

It is evident that $\mathcal{B}'$ perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$. Note that evt occurs if and only if $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$ (where $(pk_{\mathsf{r}}, m^*) \notin Q_{\text{sig}}$), which implies that $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, \widehat{c}) = \widehat{k}_{\mathsf{r}}$.

So we obtain

$$\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma}, \mathcal{B}'}(\lambda) \geq \Pr[\mathbf{G}_2 \Rightarrow 1]. \tag{8}$$

Combining equations (5)-(8), we obtain that

$$\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma}, \mathcal{B}'}(\lambda) \geq \Pr[\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}] - \mathsf{negl}(\lambda),$$

which is also non-negligible, contradicting the universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$. □

E.2 Proof of receiver binding

Proof. For any PPT adversary $\mathcal{A}$ attacking the receiver-binding property of MAMF, we denote $\mathcal{A}$'s input as $(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$, and $\mathcal{A}$'s final output as $(pk^*_{\mathsf{r}}, m^*, \sigma^*, \mathsf{tk}^*)$. Then, we parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}},$

$\widehat{c}_{\mathrm{t}}, m^*$). Let Q_{sig} denote the tuples that $\mathcal{A}$ has submitted to $\mathcal{O}^{\mathsf{Frank}}$. Since $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ is a NIZK proof obtained applying the Fiat-Shamir transform to Sigma protocols, we can further parse $\widehat{\pi} = (\widehat{\mathsf{cm}}, \widehat{\mathsf{z}})$. Note that $\mathcal{A}$ can query the random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$.

We present the following claim, the proof of which is almost the same as that of the claim in Appendix E.1. So we omit it here.

Claim. If $\mathcal{A}$ wins the game $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ during the generation of the responses to $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Frank}}$-oracle queries in $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Without loss of generality, we assume that (i) $\mathcal{A}$ has queried the random oracle on $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output $(pk^*_{\mathsf{r}}, m^*, \sigma^*)$; and (ii) for each of $\mathcal{A}$'s judge query $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}), \mathsf{tk}')$ (parsing $\pi' = (\mathsf{cm}', \mathsf{z}')$ and letting $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m')$), $\mathcal{A}$ has queried the random oracle on $(pk'_{\mathsf{r}}, \mathsf{cm}', y')$ before submitting $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{O}^{\mathsf{Judge}}$. This assumption holds without loss of generality, because if $\mathcal{A}$ does not make these queries, we can easily construct another adversary, based on $\mathcal{A}$, that makes these types of random-oracle queries.

Hence, in this case, if $\mathcal{A}$ wins the game $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ queries the random oracle on it.

Let evt denote the event that $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk^*_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1$ where $(pk^*_{\mathsf{r}}, m^*) \notin Q_{\mathrm{sig}}$.

Obviously, we have

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \Pr[\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{evt}].$$

If we can show that $\Pr[\mathsf{evt}]$ is negligible, then we finish this proof.

Assume that $\Pr[\mathsf{evt}]$ is non-negligible.

Note that $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk^*_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1$ implies that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}$ $(pk^*_{\mathsf{r}}, \widehat{\pi}, \widehat{y}) = 1$. Since $\mathsf{NIZK}^{\mathcal{R}}$ is a NIZK proof system obtained via the Fiat-Shamir transform from a Sigma protocol, according to a rewinding lemma [BS20, Lemma 19.2] and knowledge soundness of the Sigma protocol, a witness $\widehat{x}$ for $\widehat{y}$ (satisfying $\widehat{x} = (\widehat{sk}_{\mathsf{s}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$) can be extracted with non-negligible probability. The reason is as follows.

Let q_{ro} denote the total number of random oracle queries (in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$) made by $\mathcal{A}$. Since we assume that $\mathcal{A}$ has queried the random oracle on $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output $(pk^*_{\mathsf{r}}, m^*, \sigma^*, \mathsf{tk}^*)$, for $j \in [q_{\mathrm{ro}}]$, let $\mathsf{evt}^{(j)}$ denote the event that evt occurs and $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ is $\mathcal{A}$'s j-th random oracle query. Obviously, $\Pr[\mathsf{evt}] = \sum_{j=1}^{q_{\mathrm{ro}}} \Pr[\mathsf{evt}^{(j)}]$. So the fact that $\Pr[\mathsf{evt}]$ is non-negligible implies that there must be some $j^* \in [q_{\mathrm{ro}}]$, such that $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible. On the other hand, when $\mathsf{evt}^{(j^*)}$ occurs, we can rewind back to the moment when $\mathcal{A}$ made its j^*-th random oracle query, and respond with a fresh and uniformly sampled value for this query (since the challenger does not program the random oracle on $(pk^*_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ makes its j^*-th random oracle query). If $\mathsf{evt}^{(j^*)}$ occurs again, we can use the knowledge soundness of the Sigma

protocol to extract a valid witness $\widehat{x}$ for $\widehat{y}$. Since $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible, the rewinding lemma [BS20, Lemma 19.2] guarantees that the witness can be extracted successfully with non-negligible probability.

Hence, let $\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}$ (resp., $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_\mathsf{c}},\widehat{r^*_\mathsf{k}},\widehat{r_{\mathsf{USPCE}}})}$) denote the event that evt occurs and a witness $\widehat{x} = (\widehat{sk}_\mathsf{s}, \widehat{t}, \widehat{r}, \perp, \perp, \widehat{r_{\mathsf{USPCE}}})$ (resp., $\widehat{x} = (\perp, \widehat{t}, \perp, \widehat{r^*_\mathsf{c}}, \widehat{r^*_\mathsf{k}}, \widehat{r_{\mathsf{USPCE}}})$) for $\widehat{y}$ is successfully extracted. Since $\Pr[\mathsf{evt}]$ is non-negligible, we derive that at least one of $\Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}]$ and $\Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_\mathsf{c}},\widehat{r^*_\mathsf{k}},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

Case 1: If $\Pr[\mathbf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

We show a PPT adversary $\mathcal{B}$ attacking the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^\Sigma$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk}, \tilde{sk})$, $\mathcal{B}$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_\mathsf{Ag}, sk_\mathsf{Ag}, \mathsf{ap}_\mathsf{Ag}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk'_\mathsf{J}, sk'_\mathsf{J}) \leftarrow \mathsf{dHPS\text{-}KEM}^\Sigma.\mathsf{KG}(\tilde{\mathsf{pp}})$. $\mathcal{B}$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_\mathsf{J} := (pk_{\mathsf{USPCE}}, pk'_\mathsf{J})$, $sk_\mathsf{J} := (sk_{\mathsf{USPCE}}, sk'_\mathsf{J})$, and $(pk_\mathsf{s}, sk_\mathsf{s}) := (\tilde{pk}, \tilde{sk})$. Then, with these parameters, $\mathcal{B}$ simulates $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that $\mathcal{B}$ can answer $\mathcal{A}$'s oracle queries by itself. Receiving $\mathcal{A}$'s final output $(pk^*_\mathsf{r}, m^*, \sigma^*, \mathsf{tk}^*)$, if $\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\mathcal{B}$ returns $\widehat{sk}_\mathsf{s}$; otherwise, $\mathcal{B}$ returns a random secret key.

That is the construction of $\mathcal{B}$. Now we analyze $\mathcal{B}$'s advantage.

We note that $\mathcal{B}$ wins if and only if $\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}$ occurs and $\widehat{sk}_\mathsf{s} \neq sk_\mathsf{s}$, i.e.,

$$\begin{aligned}
\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{B}}(\lambda) &= \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk}_\mathsf{s} \neq sk_\mathsf{s})] \\
&= \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk}_\mathsf{s} = sk_\mathsf{s})] \\
&\geq \Pr[\mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}].
\end{aligned}$$

Next, we turn to analyze $\Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}]$. From $\mathcal{A}$'s point of view, the information on sk_s beyond pk_s is released only in the responses returned by $\mathcal{O}^{\mathsf{Frank}}$. $\mathcal{O}^{\mathsf{Frank}}$ will not provide any information on sk_s beyond pk_s except with negligible probability, because of the zero-knowledge property of $\mathsf{NIZK}^\mathcal{R}$ (note that during the execution of Frank, the secret key is only used as a component of the witness to generate the NIZK proof). Hence,

$$\Pr[\widehat{sk}_\mathsf{s} = sk_\mathsf{s} \mid \mathsf{evt}_{(\widehat{sk}_\mathsf{s},\widehat{t},\widehat{r},\perp,\perp,\widehat{r_{\mathsf{USPCE}}})}] \leq \mathsf{negl}(\lambda).$$

So $\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^\Sigma,\mathcal{B}}(\lambda)$ is non-negligible, contradicting the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^\Sigma$.

Case 2: If $\Pr[\mathbf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_\mathsf{c}},\widehat{r^*_\mathsf{k}},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

We show the proof with a sequence of games.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$. Specifically, the challenger generates pp, $(pk_\mathsf{Ag}, sk_\mathsf{Ag}, \mathsf{ap}_\mathsf{Ag})$, $(pk_\mathsf{J}, sk_\mathsf{J})$ and $(pk_\mathsf{s}, sk_\mathsf{s})$, and initializes a set $Q_{\mathrm{sig}} := \emptyset$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (we use $\mathcal{CL}$ to denote the range of the hash function modelled as a

random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Subsequently, the challenger sends $(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, the challenger returns cl to $\mathcal{A}$; otherwise, it samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathsf{r}}, m')}$: The challenger generates $\sigma' \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, sets $Q_{\mathrm{sig}} := Q_{\mathrm{sig}} \cup \{(pk'_{\mathsf{r}}, m')\}$, and returns σ' to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')}$: The challenger returns $\mathsf{Judge}(\mathsf{pp}, pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(pk^*_{\mathsf{r}}, m^*, \sigma^*, \mathsf{tk}^*)$, the challenger checks whether $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs. If so, the challenger outputs 1; otherwise, it outputs 0. In the following, we use $\mathbf{G}_i \Rightarrow 1$ to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ ($i \in \{0, 1, 2\}$).

Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}})$. Note that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\widehat{c} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathrm{c}}(\mathsf{pp}; \widehat{r^*_{\mathsf{c}}})$.

Since $\mathbf{G}_0 = \mathbf{G}^{\text{r-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, we derive

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}]. \tag{9}$$

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, the challenger returns 0 if $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$. Note that the challenger can use algorithm $\mathsf{CheckCwel}$ to check whether $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ (with the help of $\widehat{r^*_{\mathsf{c}}}$) efficiently. The ciphertext unexplainability of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_0 \Rightarrow 1]| \leq \mathsf{negl}(\lambda). \tag{10}$$

Game $\mathbf{G}_2$: This game is the same as $\mathbf{G}_1$, except that when $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Judge}}$ on $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')$, the challenger generates the response as follows:

(i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m')$.
(ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk'_{\mathsf{r}}, \pi', y') = 0$, return 0 to $\mathcal{A}$.
(iii) Check whether $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ or not (with the help of some unbounded algorithm):
  - If $c' \notin \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, return 0 to $\mathcal{A}$ directly.
  - If $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, find $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r') = c'$. Then, the challenger proceeds as follows:
    - If $\mathsf{tk}' \neq \perp$, it computes $t' \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \mathsf{tk}')$, where $\mathsf{pp}_{\mathsf{USPCE}} = pk_{\mathsf{Ag}}$ and sk_{USPCE} is from sk_{J}. It checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = k'_{\mathsf{J}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.
    - If $\mathsf{tk}' = \perp$, it computes $S_{\mathrm{t}} \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \perp)$. If there is some $t' \in S_{\mathrm{t}}$ satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = k'_{\mathsf{J}}$, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.

We stress that $\mathbf{G}_2$ is an inefficient game.

Let bad denote the event that "$\mathcal{A}$ submits a judge query $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}), \mathsf{tk}')$ satisfying that (i) $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk'_{\mathsf{r}}, \pi', y') = 1$ (where $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m')$), (ii) $c' \in \mathcal{C}^*_{pp} \backslash \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, and (iii) $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c') = k'_{\mathsf{J}}$, where $t' \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \mathsf{tk}')$ when $\mathsf{tk}' \neq \perp$, or $t' \in S_{\mathrm{t}} \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \perp)$ when $\mathsf{tk}' = \perp$." Note that from $\mathcal{A}$'s point of view, $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical except that bad occurs. The extended universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that the probability that bad occurs is negligible (note that when $c' \in \mathcal{C}^*_{pp} \setminus \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, an unbounded algorithm can trivially find r' satisfying $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathrm{c}}(\mathsf{pp}; r')$). Hence

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad}] \leq \mathsf{negl}(\lambda). \tag{11}$$

Next, we show an unbounded adversary $\mathcal{B}'$, which simulates $\mathbf{G}_2$ for $\mathcal{A}$, attacking the extended universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Receiving $(\tilde{\mathsf{pp}}, \tilde{pk})$, $\mathcal{B}'$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$ and runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$. Subsequently, $\mathcal{B}'$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$ and $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, \tilde{pk})$, and runs $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$. $\mathcal{B}'$ maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, with these parameters, $\mathcal{B}'$ simulates $\mathbf{G}_2$ for $\mathcal{A}$, answering $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, $\mathcal{B}'$ returns cl; otherwise, $\mathcal{B}'$ samples $\underline{\mathsf{cl} \leftarrow \mathcal{CL}}$, adds (str, cl) to L_{ro}, and returns cl.
- $\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathsf{r}}, m')}$: $\mathcal{B}'$ generates $\sigma' \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, sets $Q_{\mathrm{sig}} := \underline{Q_{\mathrm{sig}} \cup \{(pk'_{\mathsf{r}}, m')\}}$, and returns σ' to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')}$: $\mathcal{B}'$ generates the response as follows.
  - (i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m')$.
  - (ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk'_{\mathsf{r}}, \pi', y') = 0$, return 0 to $\mathcal{A}$.
  - (iii) $\mathcal{B}'$ checks whether $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ or not:
    - If $c' \notin \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, $\mathcal{B}'$ returns 0 to $\mathcal{A}$ directly.
    - If $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, $\mathcal{B}'$ find $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r') = c'$. Then, $\mathcal{B}'$ proceeds as follows:
      - If $\mathsf{tk}' \neq \perp$, it computes $t' \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \mathsf{tk}')$, where $\mathsf{pp}_{\mathsf{USPCE}} = pk_{\mathsf{Ag}}$ and sk_{USPCE} is from sk_{J}. It checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = k'_{\mathsf{J}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.
      - If $\mathsf{tk}' = \perp$, it computes $S_{\mathrm{t}} \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \perp)$. If there is some $t' \in S_{\mathrm{t}}$ satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = k'_{\mathsf{J}}$, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(pk^*_{\mathsf{r}}, m^*, \sigma^*, \mathsf{tk}^*)$, since $\mathcal{B}'$ cannot check whether evt occurs by itself (because it does not have sk'_{J}), it proceeds as follows.

(1) If $(pk_{\mathsf{r}}^*, m^*) \in Q_{\mathrm{sig}}$, then $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r_{\mathsf{ran}}^*, t_{\mathsf{ran}})$ as its final output.
(2) Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*)$.
(3) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}^*, \widehat{y}, \widehat{\pi}) = 0$, then $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r_{\mathsf{ran}}^*, t_{\mathsf{ran}})$ as its final output.
(4) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}^*, \widehat{y}, \widehat{\pi}) = 1$, extract a witness $\widehat{x}$ for $\widehat{y}$ (via the rewinding technique) such that $\widehat{x} = (\widehat{sk}_{\mathsf{s}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r_{\mathsf{c}}^*}, \widehat{r_{\mathsf{k}}^*}, \widehat{r_{\mathsf{USPCE}}})$:
- If $\widehat{x} = (\widehat{sk}_{\mathsf{s}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r_{\mathsf{ran}}^*, t_{\mathsf{ran}})$ as its final output.
- If $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r_{\mathsf{c}}^*}, \widehat{r_{\mathsf{k}}^*}, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}'$ firstly checks whether $\widehat{c} \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$. If so, $\mathcal{B}'$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r_{\mathsf{ran}}^*, t_{\mathsf{ran}})$ as its final output. Otherwise, $\mathcal{B}'$ proceeds as follows:
  - If $\mathsf{tk}^* \neq \bot$, it computes $\widehat{t'} = \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, \widehat{c}_{\mathrm{t}}, \mathsf{tk}^*)$, and returns $(\widehat{c}, \widehat{k}_{\mathsf{J}}, \widehat{r_{\mathsf{c}}^*}, \widehat{t'})$ as its final output.
  - If $\mathsf{tk}' = \bot$, it computes $S_{\mathrm{t}} \leftarrow \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \bot)$, uniformly samples $\widehat{t'} \leftarrow S_{\mathrm{t}}$, and returns $(\widehat{c}, \widehat{k}_{\mathsf{J}}, \widehat{r_{\mathsf{c}}^*}, \widehat{t'})$ as its final output.

That is the construction of $\mathcal{B}'$.

It is evident that $\mathcal{B}'$ perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$. Note that evt occurs if and only if $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}^*, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1$ (where $(pk_{\mathsf{r}}^*, m^*) \notin Q_{\mathrm{sig}}$), which implies that $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk_{\mathsf{J}}, \widehat{t'}, \widehat{c}) = \widehat{k}_{\mathsf{J}}$ (where $\widehat{t'} = \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, \widehat{c}_{\mathrm{t}}, \mathsf{tk}^*)$ when $\mathsf{tk}^* \neq \bot$, or $\widehat{t'} \in \mathsf{USPCE.Dec}(\mathsf{pp}_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}, c'_{\mathrm{t}}, \bot)$ when $\mathsf{tk}^* = \bot$).

Considering that $|S_{\mathrm{t}}| \leq \mathsf{poly}(\lambda)$ for some polynomial $\mathsf{poly}(\lambda)$, we obtain

$$\mathbf{Adv}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma}, \mathcal{B}'}(\lambda) \geq \frac{1}{\mathsf{poly}(\lambda)} \Pr[\mathbf{G}_2 \Rightarrow 1]. \tag{12}$$

Combining equations (9)-(12), we obtain that

$$\mathbf{Adv}^{\text{ex-univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma}, \mathcal{B}'}(\lambda) \geq \frac{1}{\mathsf{poly}(\lambda)} \Pr[\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r_{\mathsf{c}}^*}, \widehat{r_{\mathsf{k}}^*}, \widehat{\mathsf{USPCE}})}] - \mathsf{negl}(\lambda),$$

which is also non-negligible, contradicting the extended universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$. □

E.3 Proof of sender binding

Proof. For any PPT adversary $\mathcal{A}$ attacking the sender-binding property of MAMF, we denote $\mathcal{A}$'s input as $(\mathsf{pp}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$, and $\mathcal{A}$'s final output as $(pk_{\mathsf{s}}^*, m^*, \sigma^*, \mathsf{tk}^*)$. Then, we parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*)$. Since $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ is a NIZK proof obtained applying the Fiat-Shamir transform to Sigma protocols, we can further parse $\widehat{\pi} = (\widehat{\mathsf{cm}}, \widehat{\mathsf{z}})$. Note that $\mathcal{A}$ can query the random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$.

Without loss of generality, we assume that (i) $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output $(pk_{\mathsf{s}}^*, m^*, \sigma^*)$; (ii) for each of $\mathcal{A}$'s verification query $(pk'_{\mathsf{s}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}))$ (parsing $\pi' = (\mathsf{cm}', \mathsf{z}')$ and letting $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m'))$, $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \mathsf{cm}', y')$ before submitting $(pk'_{\mathsf{s}}, m', \sigma')$ to $\mathcal{O}^{\mathsf{Verify}}$; and (iii) for each of $\mathcal{A}$'s judge query $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}), \mathsf{tk}')$ (parsing $\pi' = (\mathsf{cm}', \mathsf{z}')$ and letting $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}}, m'))$, $\mathcal{A}$ has queried the random oracle on $(pk'_{\mathsf{r}}, \mathsf{cm}', y')$ before submitting $(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{O}^{\mathsf{Judge}}$. This assumption holds without loss of generality, because if $\mathcal{A}$ does not make these queries, we can easily construct another adversary, based on $\mathcal{A}$, that makes these types of random-oracle queries.

Hence, in this case, the challenger in game $\mathbf{G}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ queries the random oracle on it.

Let evt denote the event that $(\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}^*, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1) \wedge (\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}^*, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 0)$, where tk^* satisfies $\mathsf{WellForm}_{\mathsf{tk}}(\mathsf{pp}, pk_{\mathsf{J}}, \mathsf{tk}^*) = 1$ when $m^* \notin \mathsf{S}$, and $\mathsf{tk}^* = \bot$ when $m^* \in \mathsf{S}$.

We derive

$$\mathbf{Adv}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \Pr[\mathsf{evt}].$$

Thus, what remains is to prove that $\Pr[\mathsf{evt}]$ is negligible.

Assume that $\Pr[\mathsf{evt}]$ is non-negligible.

Note that when evt occurs, we have $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}^*, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$, which implies that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{\pi}, \widehat{y}) = 1$. Since $\mathsf{NIZK}^{\mathcal{R}}$ is a NIZK proof system obtained via the Fiat-Shamir transform from a Sigma protocol, according to a rewinding lemma [BS20, Lemma 19.2] and knowledge soundness of the Sigma protocol, a witness $\widehat{x}$ for $\widehat{y}$ (satisfying $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r_{\mathsf{c}}^*}, \widehat{r_{\mathsf{k}}^*}, \widehat{r_{\mathsf{USPCE}}})$) can be extracted with non-negligible probability. The reason is as follows.

Let q_{ro} denote the total number of random oracle queries (in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$) made by $\mathcal{A}$. Since $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output $(pk_{\mathsf{s}}^*, m^*, \sigma^*, \mathsf{tk}^*)$, for $j \in [q_{\mathrm{ro}}]$, let $\mathsf{evt}^{(j)}$ denote the event that evt occurs and $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ is $\mathcal{A}$'s j-th random oracle query. Obviously, $\Pr[\mathsf{evt}] = \sum_{j=1}^{q_{\mathrm{ro}}} \Pr[\mathsf{evt}^{(j)}]$. So the fact that $\Pr[\mathsf{evt}]$ is non-negligible implies that there must be some $j^* \in [q_{\mathrm{ro}}]$, such that $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible. On the other hand, when $\mathsf{evt}^{(j^*)}$ occurs, we can rewind back to the moment when $\mathcal{A}$ made its j^*-th random oracle query, and respond with a fresh and uniformly sampled value for this query (since the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ makes its j^*-th random oracle query). If $\mathsf{evt}^{(j^*)}$ occurs again, we can use the knowledge soundness of the Sigma protocol to extract a valid witness $\widehat{x}$ for $\widehat{y}$. Since $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible, the rewinding lemma [BS20, Lemma 19.2] guarantees that the witness can be extracted successfully with non-negligible probability.

Note that when $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}^*, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$, if the extracted witness $\widehat{x}$ for $\widehat{y}$ is $(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$, then $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}^*, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1$ for the aforementioned tk^*. In this case, evt does not occur.

Hence, when evt occurs, the extracted witness for $\widehat{y}$ must be $\widehat{x} = (\perp, \widehat{t}, \perp, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$. Let $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ denote the event that evt occurs and a witness $\widehat{x} = (\perp, \widehat{t}, \perp, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$ for $\widehat{y}$ is successfully extracted. Since $\Pr[\mathsf{evt}]$ is non-negligible, we derive that $\Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}]$ is also non-negligible.

Now, we consider a sequence of games.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$. Specifically, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$, $(pk_{\mathsf{J}}, sk_{\mathsf{J}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}})$, and maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modeled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Subsequently, the challenger sends $(\mathsf{pp}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, the challenger returns cl to $\mathcal{A}$; otherwise, it uniformly samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathsf{s}}, m', \sigma')}$: The challenger returns $\mathsf{Verify}(\mathsf{pp}, pk'_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m', \sigma')$ to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')}$: The challenger returns $\mathsf{Judge}(\mathsf{pp}, pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(pk^*_{\mathsf{s}}, m^*, \sigma^*, \mathsf{tk}^*)$, the challenger checks whether $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs. If so, the challenger outputs 1; otherwise, it outputs 0. In the following, we use $\mathbf{G}_i \Rightarrow 1$ to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ ($i \in \{0, 1, 2\}$).

Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$. Note that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\widehat{c} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_c(\mathsf{pp}_{\mathsf{KEM}}; \widehat{r^*_{\mathsf{c}}})$.

Since $\mathbf{G}_0 = \mathbf{G}^{\text{s-bind}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, we derive

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}]. \tag{13}$$

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when $\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, the challenger returns 0 if $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$. Note that the challenger can use algorithm $\mathsf{CheckCwel}$ to check whether $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ (with the help of $\widehat{r^*_{\mathsf{c}}}$) efficiently. The ciphertext unexplainability of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_0 \Rightarrow 1]| \leq \mathsf{negl}(\lambda). \tag{14}$$

Game $\mathbf{G}_2$: This game is the same as $\mathbf{G}_1$, except that when $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Verify}}$ on $(pk'_{\mathsf{s}}, m', \sigma')$, the challenger generates the response as follows:

(i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$.
(ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 0$, return 0 to $\mathcal{A}$.
(iii) Check whether $c' \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$ or not (with the help of some unbounded algorithm):
  - If $c' \notin \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$, return 0 to $\mathcal{A}$ directly.

- If $c' \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, find $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r') = c'$. Then, the challenger checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') = k'_{\mathsf{r}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.

We stress that $\mathbf{G}_2$ is an inefficient game.

Let bad denote the event that "$\mathcal{A}$ submits a verification query $(pk'_{\mathsf{s}}, m', \sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}))$ satisfying that (i) $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 1$ (where $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$), (ii) $c' \in \mathcal{C}^*_{pp} \backslash \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, and (iii) $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c') = k'_{\mathsf{r}}$". Note that from $\mathcal{A}$'s point of view, $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical except that bad occurs. The universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that the probability that bad occurs is negligible (note that when $c' \in \mathcal{C}^*_{pp} \setminus \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, an unbounded algorithm can trivially find r' satisfying $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r')$). Hence,

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad}] \leq \mathsf{negl}(\lambda). \tag{15}$$

Next, we show an unbounded adversary $\mathcal{B}$, which simulates $\mathbf{G}_2$ for $\mathcal{A}$, attacking the universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk})$, $\mathcal{B}$ runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$, and $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$. $\mathcal{B}$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_{\mathsf{J}} = (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$, $sk_{\mathsf{J}} = (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$, and $pk_{\mathsf{r}} := \tilde{pk}$. $\mathcal{B}$ maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, with these parameters, $\mathcal{B}$ simulates $\mathbf{G}_2$ for $\mathcal{A}$, answering $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, $\mathcal{B}$ returns cl; otherwise, $\mathcal{B}$ uniformly samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl.
- $\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathsf{s}}, m', \sigma')}$: $\mathcal{B}$ proceeds as follows:
  (i) Parse $\sigma' = (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}})$. Let $y' = (\mathsf{pp}, pk'_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathsf{t}}, m')$.
  (ii) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi', y') = 0$, return 0 to $\mathcal{A}$.
  (iii) Check whether $c' \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$ or not:
    - If $c' \notin \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, return 0 to $\mathcal{A}$ directly.
    - If $c' \in \mathcal{C}_{\mathsf{pp}}^{\text{well-f}}$, find $r' \in \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ (with the help of some unbounded algorithm) satisfying $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r') = c'$. Then, $\mathcal{B}$ checks whether $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') = k'_{\mathsf{r}}$ or not. If so, it returns 1 to $\mathcal{A}$; otherwise, it returns 0 to $\mathcal{A}$.
- $\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')}$: $\mathcal{B}$ returns $\mathsf{Judge}(\mathsf{pp}, pk'_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{A}$.

Receiving $\mathcal{A}$'s final output $(pk^*_{\mathsf{s}}, m^*, \sigma^*, \mathsf{tk}^*)$, since $\mathcal{B}$ cannot check whether $\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}$ occurs by itself (because it does not have sk_{r}), it proceeds as follows.

(1) Parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}}, m^*)$.

(2) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{y}, \widehat{\pi}) = 0$, then $\mathcal{B}$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output.
(3) If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{y}, \widehat{\pi}) = 1$, extract a witness $\widehat{x}$ for $\widehat{y}$ (via the rewinding technique) such that $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \perp, \perp, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\perp, \widehat{t}, \perp, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$:
   - If $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \perp, \perp, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output.
   - If $\widehat{x} = (\perp, \widehat{t}, \perp, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$, $\mathcal{B}$ firstly checks whether $\widehat{c} \in \mathcal{C}^{\text{well-f}}_{\mathsf{pp}}$. If so, $\mathcal{B}$ aborts the simulation and returns a random tuple $(c_{\mathsf{ran}}, k_{\mathsf{ran}}, r^*_{\mathsf{ran}})$ as its final output. Otherwise, $\mathcal{B}$ returns $(\widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{r^*_{\mathsf{c}}})$ as its final output.

That is the construction of $\mathcal{B}$.
It is evident that $\mathcal{B}$ perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$. So we obtain

$$\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}}(\lambda) \geq \Pr[\mathbf{G}_2 \Rightarrow 1]. \tag{16}$$

Combining equations (13)-(16), we obtain that

$$\mathbf{Adv}^{\text{univ}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}}(\lambda) \geq \Pr[\mathsf{evt}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{\mathsf{USPCE}})}] - \mathsf{negl}(\lambda),$$

which is also non-negligible, contradicting the universality of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$. □

E.4 Proof of universal deniability

Proof. We use a sequence of games to show that MAMF satisfies the universal deniability. Without loss of generality, assume that $\mathcal{A}$ makes q_{ch} queries to $\mathcal{O}^{\mathsf{F\text{-}F}}$ in game $\mathbf{G}^{\text{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 0$, except that the final output of the challenger in $\mathbf{G}_0$ is the adversary $\mathcal{A}$'s final output b'. Specifically, given the security parameter λ and a set $\mathsf{S} \subseteq \mathcal{U}$, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})$, $(pk_{\mathsf{J}}, sk_{\mathsf{J}})$, $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}})$. The challenger maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, the challenger sends $(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

– $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, the challenger returns cl to $\mathcal{A}$; $\underline{\text{otherwise}}$, it samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl to $\mathcal{A}$.
– $\underline{\mathcal{O}^{\mathsf{F\text{-}F}}(m')}$: The challenger generates $\sigma_0 \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, $\underline{\text{and returns}}$ σ_0 to $\mathcal{A}$.

Finally, receiving $\mathcal{A}$'s final output b', the challenger returns b' as its own final output.

In the following, we use $\mathbf{G}_i \Rightarrow 1$ (resp., $\mathbf{G}^{(j)}_{2.i} \Rightarrow 1$) to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ for $i \in \{0, 1, 3\}$ (resp., in game $\mathbf{G}^{(j)}_{2.i}$ for $i \in [q_{\text{ch}}]$ and $j \in \{0, 1, 2, 3\}$).

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}F}}$, the challenger calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$.

By the zero knowledge property of NIZK, we have that

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_{2.1}^{(0)}$: This game is totally the same as $\mathbf{G}_1$.

Game $\mathbf{G}_{2.i+1}^{(0)}$ ($i \in [q_{\mathrm{ch}} - 1]$): This game is totally the same as $\mathbf{G}_{2.i}^{(3)}$.

Game $\mathbf{G}_{2.i}^{(1)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(0)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}F}}$, k'_{r} and k'_{J} are generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}$ and $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}$, respectively.

Due to the correctness property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, when $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r')$, we have

$$\begin{aligned}\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c'),\\ \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c').\end{aligned}$$

Therefore, $\mathbf{G}_{2.i}^{(1)}$ is identical to $\mathbf{G}_{2.1}^{(0)}$ from $\mathcal{A}$'s point of view.

Game $\mathbf{G}_{2.i}^{(2)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(1)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}F}}$, c' is generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}$. That is to say, the challenger samples $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, and computes $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$.

Due to the indistinguishability property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, any PPT adversary cannot distinguish between $\mathbf{G}_{2.i}^{(2)}$ and $\mathbf{G}_{2.i}^{(1)}$ with non-negligible probability.

Game $\mathbf{G}_{2.i}^{(3)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(2)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}F}}$, $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(\mathsf{pp}; r^*_{\mathsf{k}})$, and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$.

Note that the uniformity of sampled keys of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that k'_{r} is uniformly distributed over $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$. Thus, according to the smoothness and the extended smoothness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, we obtain that $|\Pr[\mathbf{G}_{2.i}^{(3)} \Rightarrow 1] - \Pr[\mathbf{G}_{2.i}^{(2)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

We stress that in game $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, in the generation of the response to each of $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{F\text{-}F}}$, $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(\mathsf{pp}; r^*_{\mathsf{k}})$ and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$, where r^*_{c} and r^*_{k} are both uniformly and independently sampled from $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$.

Game $\mathbf{G}_3$: This game is the same as $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}F}}$, the challenger calls the NIZK generation algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ and the witness x' is set to be $(\perp, t', \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$. The case is similar to that from $\mathbf{G}_0$ to $\mathbf{G}_1$. Thus, we have $|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

In fact, $\mathbf{G}_3$ corresponds to the game $\mathbf{G}^{\mathrm{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 1$, except that the challenger in $\mathbf{G}_3$ returns $\mathcal{A}$'s final output b' as its own final output. In other words, when the adversary $\mathcal{A}$ issues $\mathcal{O}^{\mathsf{F\text{-}F}}(m')$ queries, the challenger calls Forge to generates the signatures.

Hence, we conclude that

$$\mathbf{Adv}^{\mathrm{UnivDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \frac{1}{2}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

□

E.5 Proof of receiver compromise deniability

Proof. We use a sequence of games to show that MAMF satisfies the receiver compromise deniability. Without loss of generality, assume that $\mathcal{A}$ makes q_{ch} queries to $\mathcal{O}^{\mathsf{F\text{-}RF}}$ in game $\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 0$, except that the final output of the challenger in $\mathbf{G}_0$ is the adversary $\mathcal{A}$'s final output b'. Specifically, given the security parameter λ and a set $\mathsf{S} \subseteq \mathcal{U}$, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}})$, $(pk_{\mathsf{J}}, sk_{\mathsf{J}})$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, the challenger sends $(\mathsf{pp}, sk_{\mathsf{s}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, the challenger returns cl; otherwise, the challenger samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl.
- $\underline{\mathcal{O}^{\mathsf{F\text{-}RF}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')}$: If $\mathsf{WellForm}_{\mathsf{u}}(\mathsf{pp}, pk'_{\mathsf{r}}, sk'_{\mathsf{r}}) = 0$, the challenger returns $\perp$; otherwise, the challenger generates $\sigma_0 \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, and returns σ_0 to $\mathcal{A}$.

Finally, receiving $\mathcal{A}$'s final output b', the challenger returns b' as its own final output.

In the following, we use $\mathbf{G}_i \Rightarrow 1$ (resp., $\mathbf{G}^{(j)}_{2.i} \Rightarrow 1$) to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ for $i \in \{0, 1, 3\}$ (resp., in game $\mathbf{G}^{(j)}_{2.i}$ for $i \in [q_{\mathrm{ch}}]$ and $j \in \{0, 1, 2, 3\}$).

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}RF}}$, the challenger calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$.

By the zero knowledge property of NIZK, we have that

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

Game $\mathbf{G}^{(0)}_{2.1}$: This game is totally the same as $\mathbf{G}_1$.

Game $\mathbf{G}^{(0)}_{2.i+1}$ ($i \in [q_{\mathrm{ch}} - 1]$): This game is totally the same as $\mathbf{G}^{(3)}_{2.i}$.

Game $\mathbf{G}_{2.i}^{(1)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(0)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}RF}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), k'_{r} and k'_{J} are generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}$ and $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}$, respectively. That is to say, the challenger computes $k'_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c')$, where $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$.

Due to the correctness property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, when $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r')$, we have

$$\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{r}}; r') = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c'),$$
$$\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c').$$

Therefore, $\mathbf{G}_{2.i}^{(1)}$ is identical to $\mathbf{G}_{2.1}^{(0)}$ from $\mathcal{A}$'s point of view.

Game $\mathbf{G}_{2.i}^{(2)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(1)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}RF}}$, c' is generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}$. That is to say, the challenger samples $r_{\mathsf{c}}^{*} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$, and computes $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}(\mathsf{pp}; r_{\mathsf{c}}^{*})$.

Due to the indistinguishability property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, any PPT adversary cannot distinguish between $\mathbf{G}_{2.i}^{(2)}$ and $\mathbf{G}_{2.i}^{(1)}$ with non-negligible probability.

Game $\mathbf{G}_{2.i}^{(3)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(2)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}RF}}$, k'_{J} is computed as follows: $r_{\mathsf{k}}^{*} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$, $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r_{\mathsf{k}}^{*})$.

Note that the uniformity of sampled keys of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that k'_{J} is uniformly distributed over $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$. Thus, according to the extended smoothness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, we obtain $|\Pr[\mathbf{G}_{2.i}^{(3)} \Rightarrow 1] - \Pr[\mathbf{G}_{2.i}^{(2)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

We stress that in game $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, in the generation of the response to each of $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{F\text{-}RF}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}(\mathsf{pp}; r_{\mathsf{c}}^{*})$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r_{\mathsf{k}}^{*})$, where r_{c}^{*} and r_{k}^{*} are both uniformly and independently sampled from $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$.

Game $\mathbf{G}_3$: This game is the same as $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}RF}}$, the challenger calls the NIZK generation algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ and the witness x' is set to be $(\perp, t', \perp, r_{\mathsf{c}}^{*}, r_{\mathsf{k}}^{*}, r_{\mathsf{USPCE}})$. The case is similar to that from $\mathbf{G}_0$ to $\mathbf{G}_1$. Thus, we have $|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

In fact, $\mathbf{G}_3$ corresponds to the game $\mathbf{G}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}^{\mathrm{ReComDen}}(\lambda)$ when $b = 1$, except that the challenger in $\mathbf{G}_3$ returns $\mathcal{A}$'s final output b' as its own final output. In other words, when the adversary $\mathcal{A}$ issues $\mathcal{O}^{\mathsf{F\text{-}RF}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ queries in $\mathbf{G}_3$, the challenger calls RForge to generates the signatures.

Hence, we conclude that

$$\mathbf{Adv}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}^{\mathrm{ReComDen}}(\lambda) = \frac{1}{2}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

□

E.6 Proof of judge compromise deniability

Proof. We use a sequence of games to show that MAMF satisfies the judge compromise deniability. Without loss of generality, assume that $\mathcal{A}$ makes q_{ch} queries to $\mathcal{O}^{\mathsf{F\text{-}JF}}$ in game $\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 0$, except that the final output of the challenger in $\mathbf{G}_0$ is the adversary $\mathcal{A}$'s final output b'. Specifically, given the security parameter λ and a set $\mathsf{S} \subseteq \mathcal{U}$, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}})$, $(pk_{\mathsf{J}}, sk_{\mathsf{J}})$, $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}})$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, the challenger sends $(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, sk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, the challenger returns cl; otherwise, the challenger samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl.
- $\underline{\mathcal{O}^{\mathsf{F\text{-}JF}}(m')}$: The challenger generates $\sigma_0 \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, and returns σ_0 to $\mathcal{A}$.

Finally, receiving $\mathcal{A}$'s final output b', the challenger returns b' as its own final output.

In the following, we use $\mathbf{G}_i \Rightarrow 1$ (resp., $\mathbf{G}^{(j)}_{2.i} \Rightarrow 1$) to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ for $i \in \{0, 1, 3\}$ (resp., in game $\mathbf{G}^{(j)}_{2.i}$ for $i \in [q_{\mathrm{ch}}]$ and $j \in \{0, 1, 2, 3\}$).

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}JF}}$, the challenger calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$.

By the zero knowledge property of NIZK, we have that

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

Game $\mathbf{G}^{(0)}_{2.1}$: This game is totally the same as $\mathbf{G}_1$.

Game $\mathbf{G}^{(0)}_{2.i+1}$ ($i \in [q_{\mathrm{ch}} - 1]$): This game is totally the same as $\mathbf{G}^{(3)}_{2.i}$.

Game $\mathbf{G}^{(1)}_{2.i}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}^{(0)}_{2.i}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}JF}}$, k'_{r} and k'_{J} are generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}$ and $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}$, respectively. That is to say, the challenger computes $k'_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c')$, where $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$.

Due to the correctness property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, when $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r')$, we have

$$\begin{aligned}\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk_{\mathsf{r}}; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, c'),\\ \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c').\end{aligned}$$

Therefore, $\mathbf{G}^{(1)}_{2.i}$ is identical to $\mathbf{G}^{(0)}_{2.1}$ from $\mathcal{A}$'s point of view.

Game $\mathbf{G}_{2.i}^{(2)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(1)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}JF}}$, c' is generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}$. That is to say, the challenger samples $r_{\mathsf{c}}^{*} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$, and computes $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}(\mathsf{pp}; r_{\mathsf{c}}^{*})$.

Due to the indistinguishability property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, any PPT adversary cannot distinguish between $\mathbf{G}_{2.i}^{(2)}$ and $\mathbf{G}_{2.i}^{(1)}$ with non-negligible probability.

Game $\mathbf{G}_{2.i}^{(3)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{2.i}^{(2)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{F\text{-}JF}}$, k'_{r} is computed as follows: $r_{\mathsf{k}}^{*} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(\mathsf{pp}; r_{\mathsf{k}}^{*})$.

Note that the uniformity of sampled keys of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that k'_{r} is uniformly distributed over $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$. Thus, according to the smoothness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, we have $|\Pr[\mathbf{G}_{2.i}^{(3)} \Rightarrow 1] - \Pr[\mathbf{G}_{2.i}^{(2)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

We stress that in game $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, in the generation of the response to each of $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{F\text{-}JF}}$, $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}^{*}(\mathsf{pp}; r_{\mathsf{c}}^{*})$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{SamEncK}(\mathsf{pp}; r_{\mathsf{k}}^{*})$ and $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c')$, where r_{c}^{*} and r_{k}^{*} are both uniformly and independently sampled from $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^{*}$.

Game $\mathbf{G}_3$: This game is the same as $\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)}$, except that when generating the NIZK proof in the $\mathcal{O}^{\mathsf{F\text{-}JF}}$, the challenger calls the NIZK generation algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ and the witness x' is set to be $(\perp, t', \perp, r_{\mathsf{c}}^{*}, r_{\mathsf{k}}^{*}, r_{\mathsf{USPCE}})$. The case is similar to that from $\mathbf{G}_0$ to $\mathbf{G}_1$. Thus, we have $|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_{2.q_{\mathrm{ch}}}^{(3)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

In fact, $\mathbf{G}_3$ corresponds to the game $\mathbf{G}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}^{\mathrm{JuComDen}}(\lambda)$ when $b = 1$, except that the challenger in $\mathbf{G}_3$ returns $\mathcal{A}$'s final output b' as its own final output. In other words, when the adversary $\mathcal{A}$ issues $\mathcal{O}^{\mathsf{F\text{-}JF}}(m')$ queries in $\mathbf{G}_3$, the challenger calls JForge to generates the signatures.

Hence, we conclude that

$$\mathbf{Adv}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}^{\mathrm{JuComDen}}(\lambda) = \frac{1}{2}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \mathsf{negl}(\lambda).$$

□

E.7 Proof of unframeability

Proof. For any PPT adversary $\mathcal{A}$ attacking the unframeability of MAMF, we denote $\mathcal{A}$'s input as $(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, sk_{\mathsf{J}})$, and $\mathcal{A}$'s final output as (m^*, σ^*). Then, we parse $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}})$. Let $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*)$. Let Q_{sig} denote the set of messages that $\mathcal{A}$ has submitted to $\mathcal{O}^{\mathsf{Frank}}$. Since $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{Prove}, \mathsf{Verify})$ is a NIZK proof obtained applying the Fiat-Shamir transform to Sigma protocols, we can further parse $\widehat{\pi} = (\widehat{\mathsf{cm}}, \widehat{\mathsf{z}})$. Note that $\mathcal{A}$ can query the random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$.

We present the following claim, the proof of which is almost the same as that of the claim in Appendix E.1. So we omit it here.

Claim. If $\mathcal{A}$ wins the game $\mathbf{G}^{\text{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ during the generation of the responses to $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Frank}}$-oracle queries in $\mathbf{G}^{\text{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$.

Without loss of generality, we assume that $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output (m^*, σ^*). This assumption holds without loss of generality, because if $\mathcal{A}$ does not make these queries, we can easily construct another adversary, based on $\mathcal{A}$, that makes these types of random-oracle queries.

Hence, in this case, if $\mathcal{A}$ wins the game $\mathbf{G}^{\text{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$, the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ queries the random oracle on it.

Let evt denote the event that $(\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1) \wedge (\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1)$, where $m^* \notin Q_{\text{sig}}$ and $\mathsf{tk}^* \leftarrow \mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*)$.

We derive

$$\mathbf{Adv}^{\text{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \Pr[\mathsf{evt}].$$

Thus, what remains is to prove that $\Pr[\mathsf{evt}]$ is negligible.

Assume that $\Pr[\mathsf{evt}]$ is non-negligible.

Note that when evt occurs, we have $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$, which implies that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \widehat{\pi}, \widehat{y}) = 1$. Since $\mathsf{NIZK}^{\mathcal{R}}$ is a NIZK proof system obtained via the Fiat-Shamir transform from a Sigma protocol, according to a rewinding lemma [BS20, Lemma 19.2] and knowledge soundness of the Sigma protocol, a witness $\widehat{x}$ for $\widehat{y}$ (satisfying $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ or $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$) can be extracted with non-negligible probability. The reason is as follows.

Let q_{ro} denote the total number of random oracle queries (in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$) made by $\mathcal{A}$. Since $\mathcal{A}$ has queried the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ before returning its final output (m^*, σ^*), for $j \in [q_{\text{ro}}]$, let $\mathsf{evt}^{(j)}$ denote the event that evt occurs and $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ is $\mathcal{A}$'s j-th random oracle query. Obviously, $\Pr[\mathsf{evt}] = \sum_{j=1}^{q_{\text{ro}}} \Pr[\mathsf{evt}^{(j)}]$. So the fact that $\Pr[\mathsf{evt}]$ is non-negligible implies that there must be some $j^* \in [q_{\text{ro}}]$, such that $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible. On the other hand, when $\mathsf{evt}^{(j^*)}$ occurs, we can rewind back to the moment when $\mathcal{A}$ made its j^*-th random oracle query, and respond with a fresh and uniformly sampled value for this query (since the challenger does not program the random oracle on $(pk_{\mathsf{r}}, \widehat{\mathsf{cm}}, \widehat{y})$ until $\mathcal{A}$ makes its j^*-th random oracle query). If $\mathsf{evt}^{(j^*)}$ occurs again, we can use the knowledge soundness of the Sigma protocol to extract a valid witness $\widehat{x}$ for $\widehat{y}$. Since $\Pr[\mathsf{evt}^{(j^*)}]$ is non-negligible, the rewinding lemma [BS20, Lemma 19.2] guarantees that the witness can be extracted successfully with non-negligible probability.

Hence, let $\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}$ (resp., $\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}$) denote the event that evt occurs and a witness $\widehat{x} = (\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})$ (resp., $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})$) for $\widehat{y}$ is successfully extracted. Since $\Pr[\mathsf{evt}]$ is non-negligible, we derive that at least one of $\Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}}, \widehat{t}, \widehat{r}, \bot, \bot, \widehat{r_{\mathsf{USPCE}}})}]$ and $\Pr[\mathsf{evt}_{(\bot, \widehat{t}, \bot, \widehat{r^*_{\mathsf{c}}}, \widehat{r^*_{\mathsf{k}}}, \widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

Case 1: If $\Pr[\mathbf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

We show a PPT adversary $\mathcal{B}$ attacking the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk}, \tilde{sk})$, $\mathcal{B}$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$. $\mathcal{B}$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$, $sk_{\mathsf{J}} := (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$, and $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) := (\tilde{pk}, \tilde{sk})$. Then, with these parameters, $\mathcal{B}$ simulates $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that $\mathcal{B}$ can answer $\mathcal{A}$'s oracle queries by itself. Receiving $\mathcal{A}$'s final output (m^*, σ^*), if $\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\mathcal{B}$ returns $\widehat{sk_{\mathsf{s}}}$; otherwise, $\mathcal{B}$ returns a random secret key.

That is the construction of $\mathcal{B}$. Now we analyze $\mathcal{B}$'s advantage.

Note that $\mathcal{B}$ wins if and only if $\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}$ occurs and $\widehat{sk_{\mathsf{s}}} \neq sk_{\mathsf{s}}$, i.e.,

$$\begin{aligned}
\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}}(\lambda) &= \Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk_{\mathsf{s}}} \neq sk_{\mathsf{s}})] \\
&= \Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})} \wedge (\widehat{sk_{\mathsf{s}}} = sk_{\mathsf{s}})] \\
&\geq \Pr[\mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}] - \Pr[\widehat{sk_{\mathsf{s}}} = sk_{\mathsf{s}} \mid \mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}].
\end{aligned}$$

Next, we turn to analyze $\Pr[\widehat{sk_{\mathsf{s}}} = sk_{\mathsf{s}} \mid \mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}]$. From $\mathcal{A}$'s point of view, the information on sk_{s} beyond pk_{s} is released only in the responses returned by $\mathcal{O}^{\mathsf{Frank}}$. $\mathcal{O}^{\mathsf{Frank}}$ will not provide any information on sk_{s} beyond pk_{s} except with negligible probability, because of the zero-knowledge property of $\mathsf{NIZK}^{\mathcal{R}}$ (note that during the execution of Frank, the secret key is only used as a component of the witness to generate the NIZK proof). Hence,

$$\Pr[\widehat{sk_{\mathsf{s}}} = sk_{\mathsf{s}} \mid \mathsf{evt}_{(\widehat{sk_{\mathsf{s}}},\widehat{t},\widehat{r},\bot,\bot,\widehat{r_{\mathsf{USPCE}}})}] \leq \mathsf{negl}(\lambda).$$

So $\mathbf{Adv}^{\text{sk-2pr}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}}(\lambda)$ is non-negligible, contradicting the SK-second-preimage resistance of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$.

Case 2: If $\Pr[\mathbf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible:

Recall that $\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ denote the event that evt occurs and a witness $\widehat{x} = (\bot, \widehat{t}, \bot, \widehat{r_{\mathsf{c}}^*}, \widehat{r_{\mathsf{k}}^*}, \widehat{r_{\mathsf{USPCE}}})$ for $\widehat{y} = (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathrm{t}}, m^*)$ is successfully extracted.

Let $\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ denote the event that $\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs and the extracted witness $\widehat{x}$ satisfies

$$(\widehat{c}, \widehat{r_{\mathsf{c}}^*}) \in \mathcal{R}_{\mathrm{c}}^* \ \wedge\ (\widehat{k}_{\mathsf{r}}, \widehat{r_{\mathsf{k}}^*}) \in \mathcal{R}_{\mathrm{k}}^* \ \wedge\ ((pk_{\mathsf{USPCE}}, m^*, \widehat{c}_{\mathrm{t}}), (\widehat{t}, \widehat{r_{\mathsf{USPCE}}})) \in \mathcal{R}_{\mathsf{ct}},$$

and $\mathsf{evt}^{(2)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ denote the event that $\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs and the extracted witness $\widehat{x}$ satisfies

$$(\widehat{c}, \widehat{r_{\mathsf{c}}^*}) \in \mathcal{R}_{\mathrm{c}}^* \wedge (\ (\widehat{k}_{\mathsf{J}}, (\widehat{t}, \widehat{r_{\mathsf{k}}^*})) \in \mathcal{R}_{\mathrm{k}}^{\mathsf{d}*} \wedge_{\mathrm{eq}} ((pk_{\mathsf{USPCE}}, m^*, \widehat{c}_{\mathrm{t}}), (\widehat{t}, \widehat{r_{\mathsf{USPCE}}})) \in \mathcal{R}_{\mathsf{ct}}\ ).$$

It is evident that

$$\Pr[\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}] \leq \Pr[\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}] + \Pr[\mathsf{evt}^{(2)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]. \quad (17)$$

We present the following two lemmas with postponed proofs.

Lemma 10. $\Pr[\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}] \leq \mathsf{negl}(\lambda)$.

Lemma 11. $\Pr[\mathsf{evt}^{(2)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}] \leq \mathsf{negl}(\lambda)$.

Combining these two lemmas and Eq. (17), we derive that $\Pr[\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is negligible, contradicting the assumption that $\Pr[\mathsf{evt}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

So what remains is to prove the above two lemmas.

Proof (of Lemma 10). Assume that $\Pr[\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

We show a PPT adversary $\mathcal{B}'$ attacking the key unexplainability of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk}, \tilde{sk})$, $\mathcal{B}'$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$. $\mathcal{B}'$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$, $sk_{\mathsf{J}} := (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$, and $(pk_{\mathsf{r}}, sk_{\mathsf{r}}) := (\tilde{pk}, \tilde{sk})$. Then, with these parameters, $\mathcal{B}'$ simulates $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that $\mathcal{B}'$ can answer $\mathcal{A}$'s oracle queries by itself. Receiving $\mathcal{A}$'s final output (m^*, σ^*), $\mathcal{B}'$ parses $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$, and checks whether $\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs or not. If $\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\mathcal{B}'$ returns $(\widehat{c}, \widehat{r_{\mathsf{c}}^*}, \widehat{k}_{\mathsf{r}}, \widehat{r_{\mathsf{k}}^*})$ as its final output; otherwise, it returns a random tuple $(c^{(\mathsf{ran})}, r_{\mathsf{c}}^{*(\mathsf{ran})}, k^{(\mathsf{ran})}, r_{\mathsf{k}}^{*(\mathsf{ran})})$ satisfying $((c^{(\mathsf{ran})}, r_{\mathsf{c}}^{*(\mathsf{ran})}) \in \mathcal{R}_{\mathsf{c}}^*) \wedge ((k^{(\mathsf{ran})}, r_{\mathsf{k}}^{*(\mathsf{ran})}) \in \mathcal{R}_{\mathsf{k}}^*)$ as its final output.

That is the construction of $\mathcal{B}'$. It is evident that $\mathcal{B}'$ perfectly simulates game $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$.

Note that when $\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}$ occurs, we obtain that evt occurs and $((\widehat{c}, \widehat{r_{\mathsf{c}}^*}) \in \mathcal{R}_{\mathsf{c}}^*) \wedge ((\widehat{k}_{\mathsf{r}}, \widehat{r_{\mathsf{k}}^*}) \in \mathcal{R}_{\mathsf{k}}^*)$. Recall that when evt occurs, $\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*, \sigma^*) = 1$, which implies $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk_{\mathsf{r}}, \widehat{c}) = \widehat{k}_{\mathsf{r}}$. Hence,

$$\mathbf{Adv}^{\text{K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}'}(\lambda) \geq \Pr[\mathsf{evt}^{(1)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}],$$

which is non-negligible. □

Proof (of Lemma 11). Assume that $\Pr[\mathsf{evt}^{(2)}_{(\bot,\widehat{t},\bot,\widehat{r_{\mathsf{c}}^*},\widehat{r_{\mathsf{k}}^*},\widehat{r_{\mathsf{USPCE}}})}]$ is non-negligible.

We show a PPT adversary $\mathcal{B}''$ attacking the extended key unexplainability of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ as follows.

Upon receiving $(\tilde{\mathsf{pp}}, \tilde{pk}, \tilde{sk})$, $\mathcal{B}''$ initializes a set $Q_{\mathrm{sig}} := \emptyset$, runs $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$ for some set $\mathsf{S} \subseteq \mathcal{U}$, runs $(pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.KG}(\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}})$, and runs $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$ and $(pk_{\mathsf{r}}, sk_{\mathsf{r}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\tilde{\mathsf{pp}})$. $\mathcal{B}''$ sets $\mathsf{pp} := \tilde{\mathsf{pp}}$, $pk'_{\mathsf{J}} := \tilde{pk}$, $sk'_{\mathsf{J}} := \tilde{sk}$, $pk_{\mathsf{J}} := (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$ and $sk_{\mathsf{J}} := (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$. Then, with these parameters, $\mathcal{B}''$ simulates $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$. Note that $\mathcal{B}''$ can answer $\mathcal{A}$'s oracle queries by itself. Receiving $\mathcal{A}$'s final output (m^*, σ^*), $\mathcal{B}''$ parses $\sigma^* = (\widehat{\pi}, \widehat{c}, \widehat{k}_{\mathsf{r}}, \widehat{k}_{\mathsf{J}}, \widehat{c}_{\mathsf{t}})$, and checks whether $\mathsf{evt}^{(2)}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs or not. If $\mathsf{evt}^{(2)}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, $\mathcal{B}''$ returns $(\widehat{c}, \widehat{r^*_{\mathsf{c}}}, \widehat{k}_{\mathsf{J}}, \widehat{t}, \widehat{r^*_{\mathsf{k}}})$ as its final output; otherwise, it returns a random tuple $(c^{(\mathsf{ran})}, r^{*(\mathsf{ran})}_{\mathsf{c}}, k^{(\mathsf{ran})}, t^{(\mathsf{ran})}, r^{*(\mathsf{ran})}_{\mathsf{k}})$ satisfying $((c^{(\mathsf{ran})}, r^{*(\mathsf{ran})}_{\mathsf{c}}) \in \mathcal{R}^*_{\mathsf{c}}) \wedge ((k^{(\mathsf{ran})}, (t^{(\mathsf{ran})}, r^{*(\mathsf{ran})}_{\mathsf{k}})) \in \mathcal{R}^{\mathsf{d}*}_{\mathsf{k}})$ as its final output.

That is the construction of $\mathcal{B}''$. It is evident that $\mathcal{B}''$ perfectly simulates game $\mathbf{G}^{\mathrm{Unframe}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ for $\mathcal{A}$.

Note that when $\mathsf{evt}^{(2)}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}$ occurs, we obtain that evt occurs and

$$(\widehat{c}, \widehat{r^*_{\mathsf{c}}}) \in \mathcal{R}^*_{\mathsf{c}} \wedge ((\widehat{k}_{\mathsf{J}}, (\widehat{t}, \widehat{r^*_{\mathsf{k}}})) \in \mathcal{R}^{\mathsf{d}*}_{\mathsf{k}} \wedge_{\mathrm{eq}} ((pk_{\mathsf{USPCE}}, m^*, \widehat{c}_{\mathsf{t}}), (\widehat{t}, \widehat{r_{\mathsf{USPCE}}})) \in \mathcal{R}_{\mathsf{ct}}).$$

Recall that when evt occurs, $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m^*, \sigma^*, \mathsf{tk}^*) = 1)$ (where $\mathsf{tk}^* \leftarrow \mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m^*)$),which implies $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, \widehat{t}, \widehat{c}) = \widehat{k}_{\mathsf{J}}$. Hence,

$$\mathbf{Adv}^{\text{ex-K-unexpl}}_{\mathsf{dHPS\text{-}KEM}^{\Sigma},\mathcal{B}''}(\lambda) \geq \Pr[\mathsf{evt}^{(2)}_{(\perp,\widehat{t},\perp,\widehat{r^*_{\mathsf{c}}},\widehat{r^*_{\mathsf{k}}},\widehat{r_{\mathsf{USPCE}}})}],$$

which is non-negligible. □

□

E.8 Proof of untraceability against judge

$\underline{\mathsf{SimFrank}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m)}$:
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r^*_{\mathsf{c}})$
$k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}_{\mathsf{KEM}}, sk_{\mathsf{r}}, c)$
$t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(pp, t; r^*_{\mathsf{k}})$
$r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

Fig. 11 Simulator $\mathsf{SimFrank}$ in the game $\mathbf{G}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$

Proof. We construct a simulator $\mathsf{SimFrank}$ in Fig. 11 and then we use a sequence of games to show that MAMF satisfies the untraceability against judge. Note that here we construct $\mathsf{SimFrank}$ by calling RForge.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 0$, except that the final output of the challenger in $\mathbf{G}_0$ is the adversary $\mathcal{A}$'s final output b'. Specifically, given the security parameter λ and a set $\mathsf{S} \subseteq \mathcal{U}$, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}})$, $(pk_{\mathsf{J}} = (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}), sk_{\mathsf{J}} = (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}}))$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$, and initializes a set $Q_{\mathrm{m}} := \emptyset$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, the challenger sends $(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, sk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\mathcal{O}^{\mathsf{RO}}(str)$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, the challenger returns cl; otherwise, the challenger samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl.
- $\mathcal{O}^{\mathsf{Ch}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$: If $(\mathsf{WellForm}_{\mathsf{u}}(\mathsf{pp}, pk'_{\mathsf{r}}, sk'_{\mathsf{r}}) = 0) \vee (m' \in \mathsf{S}) \vee (m' \in Q_{\mathrm{m}})$, the challenger returns $\perp$ to $\mathcal{A}$. Otherwise, it sets $Q_{\mathrm{m}} \leftarrow Q_{\mathrm{m}} \cup \{m'\}$, generates $\sigma_0 \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, and returns σ_0 to $\mathcal{A}$.
- $\mathcal{O}^{\mathsf{TKGen}}(m')$: If $m' \in Q_{\mathrm{m}}$, the challenger returns $\perp$; otherwise, it computes $\mathsf{tk}' \leftarrow \mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, sets $Q_{\mathrm{m}} \leftarrow Q_{\mathrm{m}} \cup \{m'\}$ and returns tk' to $\mathcal{A}$.

Finally, receiving $\mathcal{A}$'s final output b', the challenger returns b' as its own final output.

In the following, we use $\mathbf{G}_i \Rightarrow 1$ to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ ($i \in \{0, 1, \cdots, 7\}$).

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except for the generation of the NIZK proof in the response of the $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries. Specifically, in this game, for each of $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries, the challenger calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$ (during running the algorithm Frank) to generate the NIZK proof, instead of generating the proof with $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ as in $\mathbf{G}_0$.

By the zero knowledge property of NIZK, we have $|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Game $\mathbf{G}_2$: This game is the same as $\mathbf{G}_1$ except that in the response of the adversary's $\mathcal{O}^{\mathsf{Ch}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ queries, c'_{t} is generated as follows: sampling another $\tilde{t}' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$ and then computing $c'_{\mathrm{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', \tilde{t}'; r_{\mathsf{USPCE}})$, where $r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$ as in $\mathbf{G}_1$. In other words, t' and $\tilde{t}'$ are independently sampled in $\mathbf{G}_2$, where t' is used to compute $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathsf{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk'_{\mathsf{J}}, t'; r')$, and $\tilde{t}'$ is used to compute $c'_{\mathrm{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', \tilde{t}'; r_{\mathsf{USPCE}})$.

We present the following lemma.

Lemma 12. $|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Proof (of Lemma 12). Assume that $|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]|$ is non-negligible, where the adversary $\mathcal{A}$ makes at most q_{ch} $\mathcal{O}^{\mathsf{Ch}}$-oracle queries.

We show a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ attacking the confidentiality against user of USPCE as follows.

On receiving $(\mathsf{pp}_{\mathsf{USPCE}}, pk_{\mathsf{USPCE}}, sk_{\mathsf{USPCE}})$, $\mathcal{B}_1$ firstly uniformly samples $j^* \leftarrow [q_{\mathrm{ch}}]$, initializes a set $Q_{\mathrm{m}} := \emptyset$, runs $\mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KEMSetup}(\lambda)$, and runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\mathsf{pp}_{\mathsf{KEM}})$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\mathsf{pp}_{\mathsf{KEM}})$. $\mathcal{B}_1$ sets $\mathsf{pp} := \mathsf{pp}_{\mathsf{KEM}}$, $pk_{\mathsf{J}} = (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}})$ and $sk_{\mathsf{J}} = (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}})$. It also maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}_1$ sends $(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{J}}, sk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{RO}}$ and $\mathcal{O}^{\mathsf{TKGen}}$ as follows:

– $\underline{\mathcal{O}^{\mathsf{RO}}(str)}$: If there is some $(str, \mathsf{cl}) \in L_{\mathrm{ro}}$, $\mathcal{B}_1$ returns cl; otherwise, $\mathcal{B}_1$ samples $\underline{\mathsf{cl} \leftarrow \mathcal{CL}}$, adds (str, cl) to L_{ro}, and returns cl.
– $\underline{\mathcal{O}^{\mathsf{TKGen}}(m')}$: If $m' \in Q_{\mathrm{m}}$, $\mathcal{B}_1$ returns $\perp$; otherwise, it queries its own $\mathcal{O}^{\mathsf{TKGen}}$ oracle in game $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{B},\mathsf{S}}(\lambda)$ to obtain token tk', sets $Q_{\mathrm{m}} \leftarrow Q_{\mathrm{m}} \cup \{m'\}$ and returns tk' to $\mathcal{A}$.

For $\mathcal{A}$'s j-th query $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ to $\mathcal{O}^{\mathsf{ch}}$, when $j < j^*$, $\mathcal{B}_1$ answers the query as the challenger in game $\mathbf{G}_2$ does (i.e., during running the algorithm Frank, it samples $t', \tilde{t}' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$ and then computes $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk'_{\mathsf{J}}, t'; r)$ and $c'_{\mathrm{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', \tilde{t}'; r_{\mathsf{USPCE}})$, where r and r_{USPCE} are the corresponding randomness); when $j = j^*$, $\mathcal{B}$ answers the query as follows:

1. $\mathcal{B}_1$ generates the ciphertext and encapsulated keys as they are generated in Frank algorithm. In other words, $r \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$, $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}_{\mathsf{KEM}}; r)$, $k'_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk'_{\mathsf{r}}; r)$, $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk_{\mathrm{J}}, t'; r)$.
2. $\mathcal{B}_1$ uniformly samples $\tilde{t}' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, sets $m_0 = t'$, $m_1 = \tilde{t}'$ and $x^* = m'$, and sends (m_0, m_1, x^*) to the challenger of game $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{B},\mathsf{S}}(\lambda)$.
3. Receiving the ciphertext ct, $\mathcal{B}_2$ sets $c'_{\mathrm{t}} = \mathsf{ct}$. And then, $\mathcal{B}_2$ calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$ to generate the NIZK proof π', and returns $\sigma' := (\pi', c', k'_{\mathsf{r}}, k'_{\mathsf{J}}, c'_{\mathrm{t}})$ to $\mathcal{A}$.

Subsequently, $\mathcal{B}_2$ answers $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{RO}}$ and $\mathcal{O}^{\mathsf{TKGen}}$ as $\mathcal{B}_1$ does.

For $\mathcal{A}$'s j-th query $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ to $\mathcal{O}^{\mathsf{ch}}$ (note that $j > j^*$ now), $\mathcal{B}_2$ answers the query as the challenger in game $\mathbf{G}_1$ does (i.e., during running the algorithm Frank, it samples $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$ and computes $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}_{\mathsf{KEM}}, pk'_{\mathsf{J}}, t'; r)$ and $c'_{\mathrm{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', t'; r_{\mathsf{USPCE}})$, where r and r_{USPCE} are the corresponding randomness).

Finally, B_2 returns $\mathcal{A}$'s final output as its own final output.

That is the construction of the adversary $\mathcal{B}$.

A simple hybrid argument shows that

$$\mathbf{Adv}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{B},\mathsf{S}}(\lambda) = \frac{1}{q_{\mathrm{ch}}} |\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]|,$$

which is non-negligible. □

Game $\mathbf{G}_3$: This game is the same as $\mathbf{G}_2$, except that in the generation of the responses to $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Ch}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), k'_{r} and k'_{J} are generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}$ and $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}$, respectively. That is to say, the challenger computes $k'_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c')$, where $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$.

Due to the correctness property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, when $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r')$, we have

$$\begin{aligned}
\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{r}}; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c'),\\
\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') &= \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c').
\end{aligned}$$

Therefore, $\mathbf{G}_3$ is identical to $\mathbf{G}_2$ from $\mathcal{A}$'s point of view.

Game $\mathbf{G}_4$: This game is the same as $\mathbf{G}_3$, except that in the generation of the responses to $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Ch}}$, c' is generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}$. That is to say, the challenger chooses $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, and computes $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$, instead of sampling $r' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}$ and computing $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathsf{c}}(\mathsf{pp}; r')$.

Due to the indistinguishability property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, a simple hybrid argument shows that any PPT adversary cannot distinguish between $\mathbf{G}_3$ and $\mathbf{G}_4$ with non-negligible probability.

Game $\mathbf{G}_5$: This game is the same as $\mathbf{G}_4$, except that in the generation of the responses to $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Ch}}$, k'_{J} is computed as follows: $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r^*_{\mathsf{k}})$.

Note that the uniformity of sampled keys of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that k'_{J} is uniformly distributed over $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$. We stress that t' is uniformly and independently sampled from $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$. Thus, according to the special extended smoothness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, we obtain $|\Pr[\mathbf{G}_5 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Game $\mathbf{G}_6$: This game is the same as $\mathbf{G}_5$, except that in the generation of the responses to $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Ch}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), c'_{t} is generated as $c'_{\mathsf{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', t'; r_{\mathsf{USPCE}})$, where t' is the tag used to compute $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r^*_{\mathsf{k}})$. In other words, in $\mathbf{G}_5$, $c'_{\mathsf{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', \tilde{t}'; r_{\mathsf{USPCE}})$ where $\tilde{t}'$ is uniformly and independently sampled; in $\mathbf{G}_6$, $c'_{\mathsf{t}} = \mathsf{USPCE.Enc}(\mathsf{pp}_{\mathsf{USPCE}}, m', t'; r_{\mathsf{USPCE}})$ where t' is the tag used to compute $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r^*_{\mathsf{k}})$.

We present the following lemma, the proof of which is almost the same as that of Lemma 12, so we omit it here.

Lemma 13. $|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Game $\mathbf{G}_7$: This game is the same as $\mathbf{G}_6$, except for the generation of the NIZK proof in the response of the $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries. Specifically, in this game, for each of $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries, the challenger calls the NIZK generation algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ (with witness $x' = (\perp, t', \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$) to generate the NIZK proof.

By the zero knowledge property of NIZK, $|\Pr[\mathbf{G}_7 \Rightarrow 1] - \Pr[\mathbf{G}_6 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

It is evident that $\mathbf{G}_7$ corresponds to the game $\mathbf{G}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 1$, except that the challenger in $\mathbf{G}_7$ returns $\mathcal{A}$'s final output b' as its own final output. In other words, in $\mathbf{G}_7$, when $\mathcal{A}$ issues $\mathcal{O}^{\mathsf{Ch}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ queries, the challenger calls $\mathsf{SimFrank}$ in Fig. 11 to generates the signatures.

Therefore,

$$\mathbf{Adv}^{\text{Unt-J}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \frac{1}{2}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \mathsf{negl}(\lambda),$$

concluding the proof.

□

E.9 Proof of untraceability against agency

$\underline{\mathsf{SimFrank}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m)\text{:}}$
$(pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}) \leftarrow pk_{\mathsf{J}}$, $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $c \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathsf{c}}(\mathsf{pp}_{\mathsf{KEM}}; r^*_{\mathsf{c}})$
$k_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}_{\mathsf{KEM}}, sk_{\mathsf{r}}, c)$
$t \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$, $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(pp, t; r^*_{\mathsf{k}})$
$r_{\mathsf{USPCE}} \leftarrow \mathsf{USPCE}.\mathcal{RS}$, $c_{\mathsf{t}} \leftarrow \mathsf{USPCE.Enc}(pk_{\mathsf{USPCE}}, m, t; r_{\mathsf{USPCE}})$
$x \leftarrow (\perp, t, \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$, $y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}}, m)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathsf{t}})$

Fig. 12 Simulator SimFrank in the game $\mathbf{G}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$

Proof. We construct a simulator SimFrank in Fig. 12 and then we use a sequence of games to show that MAMF satisfies the untraceability against agency. Note that here we construct SimFrank by calling RForge.

Game $\mathbf{G}_0$: This is the original game $\mathbf{G}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 0$, except that the final output of the challenger in $\mathbf{G}_0$ is the adversary $\mathcal{A}$'s final output b'. Specifically, given the security parameter λ and a set $\mathsf{S} \in \mathcal{U}$, the challenger generates pp, $(pk_{\mathsf{Ag}}, sk_{\mathsf{Ag}})$, $(pk_{\mathsf{J}} = (pk_{\mathsf{USPCE}}, pk'_{\mathsf{J}}), sk_{\mathsf{J}} = (sk_{\mathsf{USPCE}}, sk'_{\mathsf{J}}))$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$, and initiates a set $Q_{\text{ch}} := \emptyset$. It maintains a local array L_{ro} to keep track of $\mathcal{A}$'s random oracle queries (here we use $\mathcal{CL}$ to denote the range of the hash function modelled as a random oracle in the NIZK scheme $\mathsf{NIZK}^{\mathcal{R}}$). Then, the challenger sends $(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracle queries as follows:

- $\underline{\mathcal{O}^{\mathsf{RO}}(str)\text{:}}$ If there is some $(str, \mathsf{cl}) \in L_{\text{ro}}$, the challenger returns cl; otherwise, the challenger samples $\mathsf{cl} \leftarrow \mathcal{CL}$, adds (str, cl) to L_{ro}, and returns cl.
- $\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')\text{:}}$ If $(pk'_{\mathsf{r}}, m') \notin Q_{\text{ch}}$, the challenger returns $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m', \sigma', \mathsf{tk}')$ to $\mathcal{A}$; otherwise, it returns $\perp$.
- $\underline{\mathcal{O}^{\mathsf{Ch}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')\text{:}}$ If $\mathsf{WellForm}_{\mathsf{u}}(\mathsf{pp}, pk'_{\mathsf{r}}, sk'_{\mathsf{r}}) = 0$, the challenger returns $\perp$; otherwise, it generates $\sigma_0 \leftarrow \mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk'_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m')$, and returns σ_0 to $\mathcal{A}$. It also sets $Q_{\text{ch}} \leftarrow Q_{\text{ch}} \cup \{(pk'_{\mathsf{r}}, m')\}$.

Finally, receiving $\mathcal{A}$'s final output b', the challenger returns b' as its own final output.

In the following, we use $\mathbf{G}_i \Rightarrow 1$ (resp., $\mathbf{G}^{(j)}_{3.i} \Rightarrow 1$) to denote that the challenger finally outputs 1 in game $\mathbf{G}_i$ for $i \in \{0, 1, 2, 4\}$ (resp., in game $\mathbf{G}^{(j)}_{3.i}$ for $i \in [q_{\text{ch}}]$ and $j \in \{0, 1, 2, 3\}$).

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that when $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Judge}}$ on $(pk'_{\mathsf{r}}, m', \sigma', \mathsf{tk}')$ satisfying $(pk'_{\mathsf{r}}, m') \notin Q_{\text{ch}}$, the challenger returns 0 as a response to $\mathcal{A}$ directly.

According to the receiver-binding property of MAMF, we obtain $|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_0 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Game $\mathbf{G}_2$: This game is the same as $\mathbf{G}_1$, except for the generation of the NIZK proof in the response of the $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries. Specifically, in this game, for each of $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries, the challenger calls the simulator $\mathcal{S}$ of the Fiat-Shamir NIZK proof system for $\mathcal{R}$ (during running the algorithm Frank) to generate the NIZK proof, instead of generating the proof with $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ as in $\mathbf{G}_0$.

By the zero knowledge property of NIZK, we have $|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

Game $\mathbf{G}_{3.1}^{(0)}$: This game is totally the same as $\mathbf{G}_2$.

Game $\mathbf{G}_{3.i+1}^{(0)}$ ($i \in [q_{\mathrm{ch}} - 1]$): This game is totally the same as $\mathbf{G}_{3.i}^{(3)}$.

Game $\mathbf{G}_{3.i}^{(1)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{3.i}^{(0)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{ch}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), k'_{r} and k'_{J} are generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}$ and $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}$, respectively. That is to say, the challenger computes $k'_{\mathsf{r}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c')$, where $t' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{T}$.

Due to the correctness property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, when $c' \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{c}}(\mathsf{pp}; r')$, we have

$$\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{r}}; r') = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c'),$$
$$\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dEncap}_{\mathrm{k}}(\mathsf{pp}, pk'_{\mathsf{J}}, t'; r') = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dDecap}(\mathsf{pp}, sk'_{\mathsf{J}}, t', c').$$

Therefore, $\mathbf{G}_{3.i}^{(1)}$ is identical to $\mathbf{G}_{3.i}^{(0)}$ from $\mathcal{A}$'s point of view. ,

Game $\mathbf{G}_{3.i}^{(2)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{3.i}^{(1)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{ch}}$, c' is generated by $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathrm{c}}$. That is to say, the challenger samples $r^*_{\mathsf{c}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, and computes $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathrm{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$.

Due to the indistinguishability property of the $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ scheme, any PPT adversary cannot distinguish between $\mathbf{G}_{3.i}^{(2)}$ and $\mathbf{G}_{3.i}^{(1)}$ with non-negligible probability.

Game $\mathbf{G}_{3.i}^{(3)}$ ($i \in [q_{\mathrm{ch}}]$): This game is identical to $\mathbf{G}_{3.i}^{(2)}$, except that in the generation of the response to $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{ch}}$, k'_{J} is computed as follows: $r^*_{\mathsf{k}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$, $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r^*_{\mathsf{k}})$.

Note that the uniformity of sampled keys of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$ guarantees that k'_{J} is uniformly distributed over $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{K}$. Thus, according to the extended smoothness of $\mathsf{dHPS\text{-}KEM}^{\Sigma}$, we obtain $|\Pr[\mathbf{G}_{3.i}^{(3)} \Rightarrow 1] - \Pr[\mathbf{G}_{3.i}^{(2)} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

We stress that in game $\mathbf{G}_{3.q_{\mathrm{ch}}}^{(3)}$, in the generation of the response to each of $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{ch}}$ (denoted as $(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$), $c' = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Encap}^*_{\mathrm{c}}(\mathsf{pp}; r^*_{\mathsf{c}})$, $k'_{\mathsf{r}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Decap}(\mathsf{pp}, sk'_{\mathsf{r}}, c')$ and $k'_{\mathsf{J}} = \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{dSamEncK}(\mathsf{pp}, t'; r^*_{\mathsf{k}})$, where r^*_{c} and r^*_{k} are both uniformly and independently sampled from $\mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathcal{RS}^*$.

Game $\mathbf{G}_4$: This game is the same as $\mathbf{G}_{3.q_{\mathrm{ch}}}^{(3)}$, except for the generation of the NIZK proof in the response of the $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries. Specifically, in this game, for each of $\mathcal{A}$'s $\mathcal{O}^{\mathsf{Ch}}$ queries, the challenger calls the NIZK generation algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ (with witness $x' = (\perp, t', \perp, r^*_{\mathsf{c}}, r^*_{\mathsf{k}}, r_{\mathsf{USPCE}})$) to generate the NIZK proof.

By the zero knowledge property of NIZK, $|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}^{(3)}_{3.q_{\mathrm{ch}}} \Rightarrow 1]| \leq \mathsf{negl}(\lambda)$.

It is evident that $\mathbf{G}_4$ corresponds to the game $\mathbf{G}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda)$ when $b = 1$, except that the challenger in $\mathbf{G}_4$ returns $\mathcal{A}$'s final output b' as its own final output. In other words, when the adversary $\mathcal{A}$ issues $\mathcal{O}^{\mathsf{Ch}}(pk'_{\mathsf{r}}, sk'_{\mathsf{r}}, m')$ queries in $\mathbf{G}_4$, the challenger calls $\mathsf{SimFrank}$ in Fig. 12 to generates the signatures.

Therefore, we have

$$\mathbf{Adv}^{\text{Unt-Ag}}_{\mathsf{MAMF},\mathsf{S},\mathcal{A}}(\lambda) = \frac{1}{2}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \mathsf{negl}(\lambda),$$

concluding the proof. □

E.10 Proof of confidentiality of sets

Proof. Supposing that there exists an adversary $\mathcal{A}$ breaking the confidentiality of sets of MAMF with non-negligible probability, we construct an adversary $\mathcal{B}$ to breaking the confidentiality of sets of USPCE as follows.

Upon receiving λ, $\mathcal{B}$ runs $\mathsf{pp} := \mathsf{pp}_{\mathsf{KEM}} \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{Setup}(\lambda)$, and forwards pp to the adversary $\mathcal{A}$.

After receiving the replies $(\mathsf{S}_0, \mathsf{S}_1)$ from $\mathcal{A}$, $\mathcal{B}$ sends $(\mathsf{S}_0, \mathsf{S}_1)$ to the challenger of the game of the confidentiality of sets of USPCE.

Receiving $(\mathsf{pp}_{\mathsf{USPCE}}, pk)$ from the challenger of the game of the confidentiality of sets of USPCE, $\mathcal{B}$ sets $pk_{\mathsf{Ag}} := \mathsf{pp}_{\mathsf{USPCE}}$, runs $(pk'_{\mathsf{J}}, sk'_{\mathsf{J}}) \leftarrow \mathsf{dHPS\text{-}KEM}^{\Sigma}.\mathsf{KG}(\mathsf{pp}_{\mathsf{KEM}})$, and sets $pk_{\mathsf{J}} := (pk, pk'_{\mathsf{J}})$. Then, $\mathcal{B}$ sends $(pk_{\mathsf{Ag}}, pk_{\mathsf{J}})$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ returns $\mathcal{A}$'s final output b' as its own final output.

That's the construction of $\mathcal{B}$.

It is evident that $\mathcal{B}$ perfectly simulates $\mathbf{G}^{\text{conf-set}}_{\mathsf{MAMF},\mathcal{A}}(\lambda)$ for $\mathcal{A}$, and

$$\mathbf{G}^{\text{conf-set}}_{\mathsf{USPCE},\mathcal{B}}(\lambda) \geq \mathbf{G}^{\text{conf-set}}_{\mathsf{MAMF},\mathcal{A}}(\lambda),$$

which is non-negligible. □

F Sigma protocols for plaintext knowledge

F.1 A simple case

In this section, we provide a Sigma protocol for plaintext knowledge. More details, we provide a sigma protocol for the following relation:

$$\mathcal{R}_{\mathrm{p}} = \{(c, (r, m)) : c = g^r \cdot m\}, \tag{18}$$

where g is a generator of an additional group $\mathbb{G}$ with prime order p, $r \in \mathbb{Z}_p^*$ is a randomness and $m \in \mathbb{G}$ is the plaintext. Here, we provide a sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{p}}}$ to prove that the prover knows the randomness r and the plaintext m. The protocol is shown in Fig. 13.

$\underline{P_1(c,(r,m))}$:
$w_1 \leftarrow \mathbb{Z}_p^*$, $w_2 \leftarrow \mathbb{G}$, $\mathsf{cm} := g^{w_1} \cdot w_2$
Send cm to the verifier.

$\underline{V_1(\mathsf{cm})}$: Send $\mathsf{cl} \leftarrow \mathbb{Z}_p^*$ to the prover.

$\underline{P_2(\mathsf{cm},\mathsf{cl},c,(m,r),\mathsf{aux}=(w_1,w_2))}$:
$\mathsf{z}_1 := w_1 - \mathsf{cl} \cdot r$, $\mathsf{z}_2 := w_2/m^{\mathsf{cl}}$
Send $\mathsf{z} = (\mathsf{z}_1, \mathsf{z}_2)$ to the verifier.

$\underline{V_2(c,\mathsf{cm},\mathsf{cl},\mathsf{z})}$: $(\mathsf{z}_1,\mathsf{z}_2) \leftarrow \mathsf{z}$
$\mathsf{cm}' := c^{\mathsf{cl}} \cdot g^{\mathsf{z}_1} \cdot \mathsf{z}_2$
If $\mathsf{cm} = \mathsf{cm}'$: Return 1
Else Return 0

Fig. 13 Sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{P}}}$ for relation $\mathcal{R}_{\mathrm{P}}$

Security analysis. Here, we analysis the correctness, knowledge soundness and special HVZK.
Correctness. We have that

$$\begin{aligned}\mathsf{cm}' &= c^{\mathsf{cl}} \cdot g^{\mathsf{z}_1} \cdot \mathsf{z}_2 \\ &= (g^r \cdot m)^{\mathsf{cl}} \cdot g^{w_1 - \mathsf{cl} \cdot r} \cdot (w_2/m^{\mathsf{cl}}) \\ &= g^{w_1} \cdot w_2 \\ &= \mathsf{cm}.\end{aligned}$$

Thus, we have $\mathsf{cm} = \mathsf{cm}'$, which implies correctness.
Knowledge soundness. Given two accepting transcripts $(\mathsf{cm},\mathsf{cl},\mathsf{z})$ and $(\mathsf{cm},\mathsf{cl}',\mathsf{z}')$, where $\mathsf{cl} \neq \mathsf{cl}'$, we compute

$$r = (\mathsf{z}_1 - \mathsf{z}_1') \cdot (\mathsf{cl}' - \mathsf{cl})^{-1} \mod p,$$

and

$$m = (\mathsf{z}_2/\mathsf{z}_2')^{(\mathsf{cl}'-\mathsf{cl})^{-1}} \mod p.$$

Since $\mathsf{cl} \neq \mathsf{cl}'$ and p is a prime, it is guaranteed that $(\mathsf{cl}' - \mathsf{cl})^{-1}$ exists. Thus, we can extract (r, m) successfully, which implies that $\Sigma^{\mathcal{R}_{\mathrm{P}}}$ supports knowledge soundness.
Special HVZK. The simulator Sim is shown in Fig. 14. It is easy to check that the transcript generated by Sim can be accepted by the honest verifier V_2. So we just prove that the distribution of the transcript generated by Sim is the same as that of the transcript between P and V.

For the transcript $(\mathsf{cm},\mathsf{cl},\mathsf{z})$ generated in $\Sigma^{\mathcal{R}_{\mathrm{P}}}$, we claim that cl and z are independent, with cl uniformly distributed over $\mathbb{Z}_p^*$, z_1 uniformly distributed over $\mathbb{Z}_p^*$ and z_2 uniformly distributed over $\mathbb{G}$. Note that cl is randomly picked by V_1. Since $\mathsf{z}_1 = w_1 - \mathsf{cl} \cdot r$ and w_1 is randomly sampled from $\mathbb{Z}_p^*$, it is easy to get

$\underline{\mathsf{Sim}(c, \mathsf{cl})}$:
$\mathsf{z}_1 \leftarrow \mathbb{Z}_p^*$, $\mathsf{z}_2 \leftarrow \mathbb{G}$, $\mathsf{cm} := c^{\mathsf{cl}} \cdot g^{\mathsf{z}_1} \cdot \mathsf{z}_2$
Return $(\mathsf{cm}, \mathsf{z} = (\mathsf{z}_1, \mathsf{z}_2))$

Fig. 14 Simulator Sim for Sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{P}}}$

that z_1 is uniformly distributed over $\mathbb{Z}_p^*$. In addition, the fact that $\mathsf{z}_2 = w_2/m^{\mathsf{cl}}$ and w_2 is randomly chosen from $\mathbb{G}$, implies that z_2 is uniformly distributed over $\mathbb{G}$. Therefore, our claim holds.

Then, given cl and z, cm is uniquely determined by $\mathsf{cm} = \mathsf{cm}' = c^{\mathsf{cl}} \cdot g^{\mathsf{z}_1} \cdot \mathsf{z}_2$ (the correctness guarantees that $\mathsf{cm} = \mathsf{cm}'$).

For the transcript $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ generated in the simulation, cl is randomly chosen over $\mathbb{Z}_p^*$, z_1 is randomly picked over $\mathbb{Z}_p^*$, and z_2 is randomly picked over $\mathbb{G}$. On the other hand, $\mathsf{cm} = c^{\mathsf{cl}} \cdot g^{\mathsf{z}_1} \cdot \mathsf{z}_2$.

Thus, the distribution of the transcript generated by Sim is the same as that of the transcript between P and V. So the Sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{P}}}$ is special HVZK.

In all, $\Sigma^{\mathcal{R}_{\mathrm{P}}}$ is correct and special HVZK, and supports knowledge soundness

F.2 A more general case

In this part, we provide a more general case, as shown in Eq. (19):

$$\mathcal{R}_{\mathrm{p}}^* = \{(y = (c_i, g_{i,j})_{i\in[l],j\in[k]}, x = ((r_j)_{j\in[k]}, m)) : c_i = (\prod_{j\in[k]} g_{i,j}^{r_j}) \cdot m\}, \tag{19}$$

where $g_{i,j}$ is a generator of an additional group $\mathbb{G}$ with prime order p, $r_{i,j} \in \mathbb{Z}_p^*$ is a randomness and $m \in \mathbb{G}$ is the plaintext.

Here, we provide the Sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{p}}^*}$ in Fig. 15 for relation $\mathcal{R}_{\mathrm{p}}^*$ in Eq. (19).

The security analysis of the Sigma protocol $\Sigma^{\mathcal{R}_{\mathrm{p}}^*}$ for relation $\mathcal{R}_{\mathrm{p}}^*$ in Eq. (19), is similar to the security analysis of the simple case in Appendix F.1. Therefore, we omit it here.

G Sigma protocols for "AND-EQUAL" operations

First of all, we introduce a new property for Sigma protocols and we call it *witness-only*. Informally, if the prover in the Sigma protocols can generate the commitment and response with input witness only and without using the statement, then we say that the Sigma protocols are witness-only.

Definition 35. (Witness-only). *Given a relation $\mathcal{R}$ and a Sigma protocol $\Sigma^{\mathcal{R}}$ for $\mathcal{R}$, we say the Sigma protocol $\Sigma^{\mathcal{R}}$ is* witness-only*, if for any statement-witness pair $(y, x) \in \mathcal{R}$, we can decompose the witness into k sub-witnesses $x = (x_1, \ldots, x_k)$, and if the prover $P = (P_1, P_2)$ in the Sigma protocol runs*

$\underline{P_1(y = (c_i, g_{i,j})_{i\in[l],j\in[k]}, x = ((r_j)_{j\in[k]}, m)):}$
For $j \in [k]$: $w_j \leftarrow \mathbb{Z}_p^*$
$w' \leftarrow \mathbb{G}$
For $i \in [l]$: $\mathsf{cm}_i := (\prod_{j\in[k]} g_{i,j}^{w_j}) \cdot w'$
Send $\mathsf{cm} = (\mathsf{cm}_i)_{i\in[l]}$ to the verifier.

$\underline{V_1(\mathsf{cm})}$: Send $\mathsf{cl} \leftarrow \mathbb{Z}_p^*$ to the prover.

$\underline{P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux} = ((w_j)_{j\in[k]}, w)):}$
For $j \in [k]$: $\mathsf{z}_j := w_j - \mathsf{cl} \cdot r_j$
$\mathsf{z}' := w'/m^{\mathsf{cl}}$
Send $\mathsf{z} = ((\mathsf{z}_j)_{j\in[k]}, \mathsf{z}')$ to the verifier.

$\underline{V_2(y, \mathsf{cm}, \mathsf{cl}, \mathsf{z}):}$
$((\mathsf{z}_j)_{j\in[k]}, \mathsf{z}') \leftarrow \mathsf{z}$
For $i \in [l]$: $\mathsf{cm}'_i := c_i^{\mathsf{cl}} \cdot (\prod_{j\in[k]} g_{i,j}^{\mathsf{z}_j}) \cdot \mathsf{z}'$
If $\forall i \in [l]$, $\mathsf{cm}_i = \mathsf{cm}'_i$: Return 1
Else Return 0

Fig. 15 Sigma protocol for relation $\mathcal{R}_{\mathrm{p}}$ in Eq. (19)

$P' = (P'_1, P'_2 = (P'_{2,1}, \ldots, P'_{2,k}))$ and $V = (V_1, V_2)$ runs $V' = (V_1, V'_2)$, as shown in Fig. 16, where $\mathcal{AUX}$ is the auxiliary space and also consists of k sub-spaces, $\mathcal{AUX} = \mathcal{AUX}_1 \times \cdots \times \mathcal{AUX}_k$ ($\mathcal{AUX}_j$ corresponds to x_j), and the responses z is uniformly distributed over $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_k$ (z_j is uniformly distributed over $\mathcal{Z}_j$ for each $j \in [k]$).

In our definition of witness-only, we refer to the property of challenge-independent extended honest-verifier zero-knowledge (CIEHVZK) proposed in [GGHAK22], which guarantees the existence of algorithm V'_2 along with the method of verification (i.e., recomputation of the commitment and then compare the equality) and it brings in some convenience in analysis in the HVZK.

We can examine that many Sigma protocols are witness-only. Here, we mainly focus on Sigma protocols for ψ-preimages.

Theorem 5. (Sigma protocol for ψ-preimages is witness-only). *Let $\mathfrak{G}_1$ and $\mathfrak{G}_2$ be groups with group operations $*_1$ and $*_2$ respectively, and let $\psi : \mathfrak{G}_1 \to \mathfrak{G}_2$ be a one-way group-homomorphism. Recall the simple Σ-protocol (denoted as Σ_ψ) of Cramer and Damgård [CD98] for the relation of preimages $\mathcal{R}_\psi$ ($(y, x) \in \mathcal{R}_\psi$ if and only if it holds that $y = \psi(x)$, where $y \in \mathfrak{G}_2$ and $x \in \mathfrak{G}_1$). The protocol Σ_ψ works as follows:*

1. *$P_1(y, x; r)$: The prover samples $r \leftarrow \mathfrak{G}_1$ and sends the image $\mathsf{cm} = \psi(r) \in \mathfrak{G}_2$ to the verifier.*
2. *$V_1(\mathsf{cm})$: On receiving cm from the prover, the verifier samples a challenge cl and sends it to the prover.*

$\underline{P_1(y, x)}$: $/\!/(y, x = (x_1, \ldots, x_k)) \in \mathcal{R}$
$\mathsf{aux} = (\mathsf{aux}_1, \ldots, \mathsf{aux}_k) \leftarrow \mathcal{AUX}$, $\mathsf{cm} \leftarrow P_1'(\mathsf{aux})$
Send cm to the verifier

$\underline{V_1(\mathsf{cm})}$: Send $\mathsf{cl} \leftarrow \mathcal{CL}$ to the prover

$\underline{P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux})}$:
For $j \in [k]$: $\mathsf{z}_j \leftarrow P_{2,j}'(\mathsf{cl}, \mathsf{aux}_j, x_j)$
Send $\mathsf{z} = (\mathsf{z}_j)_{j \in [k]}$ to the verifier

---

$\underline{V_2(y, \mathsf{cm}, \mathsf{cl}, \mathsf{z})}$:
$\mathsf{cm}' \leftarrow V_2'(y, \mathsf{cl}, \mathsf{z})$
If $\mathsf{cm} = \mathsf{cm}'$: Return 1
Else Return 0

Fig. 16 Sigma protocol $\Sigma^{\mathcal{R}}$ for relation $\mathcal{R}$

3. $P_2(\mathsf{cm}, \mathsf{cl}, y, x; r)$*: The prover interprets* cl *as an integer from a subset* $\mathcal{CL} \subseteq \mathcal{Z}$ *and replies with* $\mathsf{z} = x^{\mathsf{cl}} *_1 r$.
4. $V_2(y, \mathsf{cm}, \mathsf{cl}, \mathsf{z})$*: The verifier checks* $\psi(\mathsf{z}) \stackrel{?}{=} y^{\mathsf{cl}} *_2 \mathsf{cm}$.

Completeness follows since ψ *is a homomorphism:* $\psi(\mathsf{z}) = \psi(x^{\mathsf{cl}} *_1 r) = \psi(x)^{\mathsf{cl}} *_2 \psi(r) = y^{\mathsf{cl}} *_2 \mathsf{cm}$. *The knowledge soundness error is* $1/|\mathcal{CL}|$. *For any homomorphism* ψ, Σ_ψ *is witness-only.*

Proof. From the description in Theorem 5, we can know that in P_1 and P_2, the statement is not used to compute the commitment or the response. More exactly, we can define $\mathsf{cm} \leftarrow P_1'(\mathsf{aux} = r) = \psi(r)$, $\mathsf{z} \leftarrow P_2'(\mathsf{cl}, \mathsf{aux} = r, x) = x^{\mathsf{cl}} *_1 r$, $\mathsf{cm} \leftarrow V_2'(y, \mathsf{cl}, \mathsf{z}) = (y^{\mathsf{cl}})^{-1} *_2 \psi(\mathsf{z})$. Here, $\mathcal{AUX} = \mathcal{Z} = \mathfrak{G}_1$. Since $\mathsf{z} = x^{\mathsf{cl}} *_1 r$ and r is uniformly sampled over $\mathfrak{G}_1$, it implies that z is uniformly distributed over $\mathcal{Z} = \mathfrak{G}_1$.

Therefore, for any homomorphism ψ, Σ_ψ is witness-only. □

Remark 6. The following variants of ψ (with different choices of $\mathfrak{G}_1$, $\mathfrak{G}_1$, ψ) are captured in this generalization (along with other similar Sigma protocols):

- Guillou-Quisquater Sigma protocol [GQ88] (e-roots in an RSA group) for which $\mathfrak{G}_1 = \mathfrak{G}_2 = \mathbb{Z}_n^*$ for a semi-prime $n = pq$, $\mathcal{CL} = (0, e)$ and $\psi(x) := x^e$ for some prime $e \in \mathbb{N}$.
- Schnorr's Sigma protocol [Sch89] (knowledge of discrete logarithm): for which $\mathfrak{G}_1 = \mathbb{Z}_{|\mathbb{G}|}^*$, $\mathfrak{G}_2 = \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of prime order $|\mathbb{G}|$, $\mathcal{CL} = (0, |G|)$ and $\psi(x) := g^x$ for some $g \in \mathbb{G}$.
- Okamoto's Sigma protocol [Oka95] (knowledge of multiple discrete logarithms): similar to Schnorr's Sigma protocol, here we omit the detailed discussion.
- Chaum-Pedersen protocol [CP92] (equality of discrete logarithm): for which $\mathfrak{G}_1 = \mathbb{Z}_{|\mathbb{G}|}^*$, $\mathfrak{G}_2 = \mathbb{G} \times \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of prime order $|\mathbb{G}|$, $\mathcal{CL} = (0, |G|)$ and $\psi : \mathbb{Z}_{|\mathbb{G}|} \to \mathbb{G} \times \mathbb{G}$, $\psi(x) := (g_1^x, g_2^x)$ for some $g_1, g_2 \in \mathbb{G}$.

– Attema-Cramer [AC20] (opening of linear forms): for which $\mathfrak{G}_1 = \mathbb{Z}_{|\mathbb{G}|}^l \times \mathbb{Z}_{|\mathbb{G}|}$, $\mathfrak{G}_2 = (\mathbb{Z}_{|\mathbb{G}|}, \mathbb{G})$, $\mathcal{CL} = (0, |G|)$ and $\psi((\mathbf{x}, \gamma)) := (L(\mathbf{x}), \mathbf{g}^{\mathbf{x}} h^{\gamma})$ for some linear form $L(\mathbf{x}) = \langle \mathbf{x}, \mathbf{s} \rangle$, $\mathbf{s} \in \mathbb{Z}_{|\mathbb{G}|}^l$.
– Sigma protocol $\Sigma^{\mathcal{R}^*}$ in Fig. 15 in Appendix F.2: for which $\mathfrak{G}_1 = \mathbb{Z}_p^k \times \mathbb{G}$, $\mathfrak{G}_2 = \mathbb{G}^l$ where $\mathbb{G}$ is a cyclic group of prime order $|\mathbb{G}| = p$, $\mathcal{CL} = \mathbb{Z}_p^*$ and $\psi(x = ((r_j)_{j\in[k]}, m)) := (c_i = (\prod_{j\in[k]} g_{i,j}^{r_j}) \cdot m)_{i\in[l]}$, where $g_{i,j}$ is a generator of $\mathbb{G}$ with order p.

We can have the following corollary. Here, we omit the proof for simplicity, since the corollary is a trivially implied by definition of witness-only.

Corollary 1. *Given two relations $\mathcal{R}_1$ and $\mathcal{R}_2$, and given two witness-only Sigma protocols $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$ respectively, if the relation $\mathcal{R}_3$ consists $\mathcal{R}_1$ and $\mathcal{R}_2$ with an "AND" operation or an "OR" operation, then following [BS20], we can obtain a Sigma protocol $\Sigma^{\mathcal{R}_3}$ from $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$, and $\Sigma^{\mathcal{R}_3}$ is also witness-only.*

Then, we shown how to combine two relations with an "AND-EQUAL" operation and show how to combine two Sigma protocols with an "AND-EQUAL" operation. Before that, let's define the "AND-EQUAL" operations as follows.

Definition 36. (*Combining relations with "AND-EQUAL" operations).* *We define an "AND-EQUAL" operation as $\wedge_{\text{eq}}$ over the relations. Given two relations $\mathcal{R}_1$ and $\mathcal{R}_2$, if $\mathcal{R}_3 := (\mathcal{R}_1 \wedge_{\text{eq}} \mathcal{R}_2)$, then it means that*

$$\mathcal{R}_3 := \{((y_1, y_2), x) : (y_1, \boxed{x}) \in \mathcal{R}_1 \wedge (y_2, \boxed{x}) \in \mathcal{R}_2\}. \tag{20}$$

where the box in light gray $\boxed{x}$ denotes the same witness.

Further, we present a more relaxed definition for "AND-EQUAL" operations. In a nutshell, we allow that only part sub-witnesses are equal and we call it an "AND-EQUAL$_l$" operation.

Definition 37. (*Combining relations with "AND-EQUAL$_l$" operations).* *We define an "AND-EQUAL$_l$" operation as $\wedge_{\text{eq}}$ over the relations. Given two relations $\mathcal{R}_1$ and $\mathcal{R}_2$, if $\mathcal{R}_3 := (\mathcal{R}_1 \wedge_{\text{eq}} \mathcal{R}_2)$, then it means that*

$$\begin{aligned} \mathcal{R}_3 := \{((y_1, y_2), & x = (x'_1, \ldots, x'_l,\ x'_{l+1}, \ldots, x'_{k_1},\ x'_{k_1+1}, \ldots, x'_{k_1+k_2-l})) : \\ & (y_1, x_1 = (\boxed{x'_1, \ldots, x'_l},\ x'_{l+1}, \ldots, x'_{k_1})) \in \mathcal{R}_1 \\ & \wedge (y_2, x_2 = (\boxed{x'_1, \ldots, x'_l},\ x'_{k_1+1}, \ldots, x'_{k_1+k_2-l})) \in \mathcal{R}_2\}. \end{aligned} \tag{21}$$

where the box in light gray $\boxed{x'_1, \ldots, x'_l}$ denotes the equal part of sub-witnesses.

Now, we are ready to present how to construct a Sigma protocol for the relation combined with an "AND-EQUAL$_l$" operation. We have the following theorem.

Theorem 6. *Given $\Sigma^{\mathcal{R}_1}$ for the relation $\mathcal{R}_1$ with k_1 sub-witnesses and $\Sigma^{\mathcal{R}_2}$ for the relation $\mathcal{R}_2$ with k_2 sub-witnesses, then we can construct a sigma protocol $\Sigma^{\mathcal{R}_3}$ for $\mathcal{R}_3 := (\mathcal{R}_1 \wedge_{\text{eq}} \mathcal{R}_2)$ (as defined in Eq. (21), x_1 and x_2 share l same sub-witnesses), and $\Sigma^{\mathcal{R}_3}$ is also witness-only, if*

– $\Sigma^{\mathcal{R}_1}$ *and* $\Sigma^{\mathcal{R}_2}$ *are witness-only;*
– *For each* $j \in [l]$*, it holds* $\Sigma^{\mathcal{R}_1}.\mathcal{AUX}_j$ *and* $\Sigma^{\mathcal{R}_2}.\mathcal{AUX}_j$ *are the same;*
– *For each* $j \in [l]$*,* $\Sigma^{\mathcal{R}_1}.P'_{2,j}$ *and* $\Sigma^{\mathcal{R}_2}.P'_{2,j}$ *are the same;*
– *For each* $j \in [l]$*,* $\Sigma^{\mathcal{R}_1}.\mathcal{Z}_j$ *and* $\Sigma^{\mathcal{R}_2}.\mathcal{Z}_j$ *are the same;*

Proof. The construction is shown in Fig. 17.

$\underline{P_1(y = (y_1, y_2), x = (x_1, \ldots, x_{k_1+k_2-l}))}$:
$\mathsf{aux} \leftarrow \mathcal{AUX}$ // $\mathcal{AUX} = \Sigma^{\mathcal{R}_1}.\mathcal{AUX}_1 \times \cdots \times \Sigma^{\mathcal{R}_1}.\mathcal{AUX}_{k_1} \times \Sigma^{\mathcal{R}_2}.\mathcal{AUX}_{l+1} \times \cdots \times \Sigma^{\mathcal{R}_2}.\mathcal{AUX}_{k_2}$
$\mathsf{aux}_{\mathcal{R}_1} = (\mathsf{aux}_1, \ldots, \mathsf{aux}_{k_1})$, $\mathsf{aux}_{\mathcal{R}_2} = (\mathsf{aux}_1, \ldots, \mathsf{aux}_l, \mathsf{aux}_{k_1+1}, \ldots, \mathsf{aux}_{k_1+k_2-l})$
$\mathsf{cm}_{\mathcal{R}_1} \leftarrow \Sigma^{\mathcal{R}_1}.P'_1(\mathsf{aux}_{\mathcal{R}_1})$, $\mathsf{cm}_{\mathcal{R}_2} \leftarrow \Sigma^{\mathcal{R}_2}.P'_1(\mathsf{aux}_{\mathcal{R}_2})$
Send $\mathsf{cm} = (\mathsf{cm}_{\mathcal{R}_1}, \mathsf{cm}_{\mathcal{R}_2})$ to the verifier

$\underline{V_1(\mathsf{cm})}$: Send $\mathsf{cl} \leftarrow \mathcal{CL}$ to the prover

$\underline{P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux})}$:
For each $j \in [k_1]$: $\mathsf{z}_j \leftarrow \Sigma^{\mathcal{R}_1}.P'_{2,j}(\mathsf{cl}, \mathsf{aux}_j, x_j)$ // $\Sigma^{\mathcal{R}_1}.P'_{2,j} = \Sigma^{\mathcal{R}_2}.P'_{2,j}$, when $j \in [l]$
For each $i \in (k_1, k_1 + k_2 - l]$: $\mathsf{z}_i \leftarrow \Sigma^{\mathcal{R}_2}.P'_{2,i-k_1+l}(\mathsf{cl}, \mathsf{aux}_i, x_i)$
Send $\mathsf{z} = (\mathsf{z}_1, \ldots, \mathsf{z}_{k_1+k_2-l})$ to the verifier

---

$\underline{V_2(y, \mathsf{cm}, \mathsf{cl}, \mathsf{z})}$
$\mathsf{z}_{\mathcal{R}_1} = (\mathsf{z}_1, \ldots, \mathsf{z}_{k_1})$, $\mathsf{z}_{\mathcal{R}_2} = (\mathsf{z}_1, \ldots, \mathsf{z}_l, \mathsf{z}_{k_1+1}, \ldots, \mathsf{z}_{k_1+k_2-l})$
If $(\Sigma^{\mathcal{R}_1}.V_2(y_1, \mathsf{cm}_{\mathcal{R}_1}, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_1}) = 1) \wedge (\Sigma^{\mathcal{R}_2}.V_2(y_2, \mathsf{cm}_{\mathcal{R}_2}, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_2}) = 1)$:
  Return 1
Else Return 0

Fig. 17 Sigma protocol $\Sigma^{\mathcal{R}_3}$ for relation $\mathcal{R}_3$ from $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$

Security analysis. Here, we analysis the correctness, knowledge soundness and special HVZK.
Correctness. Correctness is trivially implied by the correctness of $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$.
Knowledge soundness. Note that given two accepting transcripts $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ and $(\mathsf{cm}, \mathsf{cl}', \mathsf{z}')$, where $\mathsf{cl} \neq \mathsf{cl}'$, the knowledge soundness of $\Sigma^{\mathcal{R}_1}$ guarantees that we can extract $(x_1, \ldots, x_{k_1})$. Similarly, the knowledge soundness of $\Sigma^{\mathcal{R}_2}$ guarantees that we can extract $(x_{k_1+1}, \ldots, x_{k_1+k_2-l})$. Thus, $\Sigma^{\mathcal{R}_3}$ supports knowledge soundness.
Special HVZK. The simulator Sim is shown in Fig. 18. It is easy to check that the transcript generated by Sim can be accepted by the honest verifier V_2. So we just prove that the distribution of the transcript generated by Sim is the same as that of the transcript between P and V.

For the transcript $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ generated in $\Sigma^{\mathcal{R}_3}$, we claim that cl and z are independent, with cl uniformly distributed over $\mathcal{CL}$ and z uniformly distributed over $\mathcal{Z}$. Note that cl is randomly picked over $\mathcal{CL}$ by V_1. The witness-only property of $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$ implies that z uniformly distributed over $\mathcal{Z}$. Therefore, our claim holds.

$\underline{\mathsf{Sim}(y=(y_1,y_2),\mathsf{cl})}$:
$\mathsf{z} \leftarrow \mathcal{Z}$ $/\!/\mathcal{Z} = \Sigma^{\mathcal{R}_1}.\mathcal{Z}_1 \times \cdots \times \Sigma^{\mathcal{R}_1}.\mathcal{Z}_{k_1} \times \Sigma^{\mathcal{R}_2}.\mathcal{Z}_{l+1} \times \cdots \times \Sigma^{\mathcal{R}_2}.\mathcal{Z}_{k_2}$
$\mathsf{z}_{\mathcal{R}_1} = (\mathsf{z}_1, \ldots, \mathsf{z}_{k_1})$, $\mathsf{z}_{\mathcal{R}_2} = (\mathsf{z}_1, \ldots, \mathsf{z}_l, \mathsf{z}_{k_1+1}, \ldots, \mathsf{z}_{k_1+k_2-l})$
$\mathsf{cm}_{\mathcal{R}_1} \leftarrow \Sigma^{\mathcal{R}_1}.V_2'(y_1, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_1})$, $\mathsf{cm}_{\mathcal{R}_2} \leftarrow \Sigma^{\mathcal{R}_2}.V_2'(y_2, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_2})$
Return $(\mathsf{cm} = (\mathsf{cm}_{\mathcal{R}_1}, \mathsf{cm}_{\mathcal{R}_1}), \mathsf{z})$

Fig. 18 Simulator Sim for Sigma protocol $\Sigma^{\mathcal{R}_3}$

Then, given cl and z, cm is uniquely determined by $\mathsf{cm}_{\mathcal{R}_1} = \mathsf{cm}'_{\mathcal{R}_1} = \Sigma^{\mathcal{R}_1}.V_2'(y_1, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_1})$ and $\mathsf{cm}_{\mathcal{R}_2} = \mathsf{cm}'_{\mathcal{R}_2} = \Sigma^{\mathcal{R}_2}.V_2'(y_2, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_2})$ (which is guaranteed by the correctness of $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$).

For the transcript $(\mathsf{cm}, \mathsf{cl}, \mathsf{z})$ generated in the simulation, cl is randomly chosen over $\mathcal{CL}$ and z is randomly picked over $\mathcal{Z}$. On the other hand, it holds that $\mathsf{cm}_{\mathcal{R}_1} \leftarrow \Sigma^{\mathcal{R}_1}.V_2'(y_1, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_1})$, and $\mathsf{cm}_{\mathcal{R}_2} \leftarrow \Sigma^{\mathcal{R}_2}.V_2'(y_2, \mathsf{cl}, \mathsf{z}_{\mathcal{R}_2})$.

Thus, the distribution of the transcript generated by Sim is the same as that of the transcript between P and V. So the Sigma protocol $\Sigma^{\mathcal{R}_3}$ is special HVZK.

It is trivial to know that $\Sigma^{\mathcal{R}_3}$ constructed in Fig. 18 is also witness-only, since $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$ are witness-only.

Therefore, a sigma protocol $\Sigma^{\mathcal{R}_3}$ for $\mathcal{R}_3 := (\mathcal{R}_1 \wedge_{\text{eq}} \mathcal{R}_2)$ constructed from $\Sigma^{\mathcal{R}_1}$ and $\Sigma^{\mathcal{R}_2}$ is correct and special HVZK and supports knowledge soundness, and $\Sigma^{\mathcal{R}_3}$ is also witness-only. □

H Improvements on the concrete construction of MAMF

As introduced in Sec. 6, we can construct an MAMF from USPCE and dual HPS-KEM$^{\Sigma}$. Plugging with a concrete USPCE scheme in Sec. 4 and a concrete dual HPS-KEM$^{\Sigma}$ in Sec. 5, we can obtain a concrete construction of the MAMF. In this section, we show some improvements on the concrete construction.

The detailed description is as follows.

The setup algorithm Setup and key generation algorithms (i..e, $\mathsf{KG}_{\mathsf{Ag}}$, KG_{J} and KG_{u}) are shown in Fig. 19 and are essentially the same as those in the general construction, except that here we adopt the concrete USPCE and HPS-KEM$^{\Sigma}$ as presented in Sec. 4 and Sec. 5. A mirror difference is that USPCE and dual HPS-KEM$^{\Sigma}$ can share the same group generation. Thus, pk_{Ag} does not contains the description about the group for the USPCE.

Note that in our setup algorithm, we initialize a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with order p and the dual HPS-KEM$^{\Sigma}$ is constructed over $\mathbb{G}_T$.

The main body of the improved MAMF scheme is shown in Fig. 20.

There are some changes in the franking algorithm Frank.

- $\underline{\mathsf{dEncap}_{\mathsf{k}} \Rightarrow \mathsf{Encap}_{\mathsf{k}} \text{ for } k_{\mathsf{J}}}$. It directly invokes $\mathsf{Encap}_{\mathsf{k}}$ to generate the k_{J} for the judge, instead of invoking $\mathsf{dEncap}_{\mathsf{k}}$ with input another tag t. Therefore, in fact, we do not require the second approach to generate encapsulated key in dual HPS-KEM$^{\Sigma}$ any more.

$\underline{\mathsf{Setup}(\lambda)}$:
$(e, \mathbb{G}, \mathbb{G}_T, g, p) \leftarrow \mathsf{GenG}(\lambda)$ // g is the generator of $\mathbb{G}$.
Choose two generators of $\mathbb{G}_T$ with order p: h_1 and h_2.
Return $\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, e, g, h_1, h_2, p)$

$\underline{\mathsf{KG}_{\mathsf{Ag}}(\mathsf{pp}, \mathsf{S})}$: // $(pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}}, sk_{\mathsf{Ag}}) := (\mathsf{pp}_{\mathsf{USPCE}}, \mathsf{ap}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}) \leftarrow \mathsf{USPCE.Setup}(\lambda, \mathsf{S})$
Choose hash functions $\widetilde{\mathsf{H}}, \overline{\mathsf{H}} : \mathcal{M} \to \mathbb{G}^*$.
$(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$ // $n = \mathsf{poly}(\lambda)$ and $|T_{\mathsf{init}}| = n' = \mathsf{poly}(n) = \mathsf{poly}(\lambda)$.
// pp_{CH} contains k random hash functions $\mathsf{H}_k : \mathcal{M} \to [n']$, T_{init} is the hash table, ST is the stash.
$(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, $\alpha' \leftarrow \mathbb{Z}_p^*$, $A' := g^{\alpha'}$, $s \leftarrow \mathbb{Z}_p^*$, $Y' := g^s$
Initialize two empty tables $\widetilde{T}, T'$ with length n'.
For each $i \in [n']$:
  If $T_{\mathsf{S}}[i] = \perp$: $\widetilde{T}[i] \leftarrow \mathbb{G}$, $T'[i] := (\widetilde{T}[i])^{\alpha'}$
  Else $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$, $T'[i] := (\widetilde{T}[i])^{\alpha'}$
Return $(pk_{\mathsf{Ag}} = (\widetilde{\mathsf{H}}, Y', A', \mathsf{pp}_{\mathsf{CH}}), \mathsf{ap}_{\mathsf{Ag}} = T', sk_{\mathsf{Ag}} = (\widetilde{T}, \mathsf{S}, s))$

$\underline{\mathsf{KG}_{\mathsf{J}}(\mathsf{pp}, pk_{\mathsf{Ag}}, \mathsf{ap}_{\mathsf{Ag}})}$:
$\alpha \leftarrow \mathbb{Z}_p^*$, $\beta \leftarrow \mathbb{Z}_p^*$, $X := g^\beta, Y := (Y')^\beta$ // USPCE.KG
For each $i \in [n']$: $T[i] := (T'[i])^\alpha$
$(s_1, s_2) \leftarrow (\mathbb{Z}_p^*)^2$, $pk'_{\mathsf{J}} := h_1^{s_1} h_2^{s_2}$ // dHPS-KEM$^{\Sigma}$.KG
Return $(pk_{\mathsf{J}} = (T, X, Y, pk'_{\mathsf{J}}), sk_{\mathsf{J}} = (\alpha, \beta, s_1, s_2))$

$\underline{\mathsf{KG}_{\mathsf{u}}(\mathsf{pp})}$: // dHPS-KEM$^{\Sigma}$.KG
$(s_1, s_2) \leftarrow (\mathbb{Z}_p^*)^2$, $pk := h_1^{s_1} h_2^{s_2}$, Return $(pk, sk = (s_1, s_2))$

Fig. 19 Algorithm descriptions of Setup, $\mathsf{KG}_{\mathsf{Ag}}$, KG_{J} and KG_{u}

- $\underline{\mathsf{USPCE.Enc} \text{ for } t \Rightarrow \mathsf{USPCE.Enc} \text{ for } k_{\mathsf{J}}}$. For the encryption part of USPCE, Frank here directly encrypt the encapsulation key of the judge, i.e., k_{J}.
- $\underline{\text{Changes in the relation } \mathcal{R}}$. In Frank algorithm, it also utilizes a NIZK proof algorithm $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}$ to create a NIZK proof. The relation $\mathcal{R}$ is defined in Eq. (22), which is instantiation of the relation shown in Sec. 6 and Fig. 1 with some modification. The detailed discussion is placed in the later section.
- $\underline{\text{Removing } k_{\mathsf{J}} \text{ in the signature } \sigma}$. The encapsulation key of the judge k_{J} is not included in the signature σ any more.

The verification algorithm Verify follows the step in the general construction. So as the algorithm TKGen. Thus, we omit the description of them.

The moderation algorithm Judge follows the main framework of the verification step in general construction. The difference is that:

- $\underline{\mathsf{dDecap} \Rightarrow \mathsf{Decap}}$. Here, it calls the decapsulation algorithm Decap, instead of dDecap to obtain the decapsulated key.
- $\underline{\mathsf{USPCE.Dec} \text{ to obtain } t \Rightarrow \mathsf{USPCE.Dec} \text{ to obtain } k_{\mathsf{J}}}$. The decryption of USPCE is to obtain k_{J}, instead of the tag t. If the token is empty, then the moderation algorithm Judge in the improved construction calls the decryption of

$\underline{\mathsf{Frank}(\mathsf{pp}, sk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$r \leftarrow \mathbb{Z}_p^*$, $u_1 := h_1^r$, $u_2 := h_2^r$, $k_{\mathsf{r}} := pk_{\mathsf{r}}^r$, $k_{\mathsf{J}} := (pk'_{\mathsf{J}})^r$ //dHPS-KEM$^{\Sigma}$.Encap
For each $j \in [k]$: //USPCE.Enc
  $\gamma_j \leftarrow \mathbb{Z}_p^*$, $Q_j := e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j}$, $S_j := e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot k_{\mathsf{J}}$
If $e(X, Y') \neq e(g, Y)$: Return $\perp$
$r'' \leftarrow \mathbb{Z}_p^*$, $c := (g^{r''}, e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot k_{\mathsf{J}})$
$x \leftarrow (s_1, s_2, r, \perp, r'', \perp, \perp, (\gamma_j)_{j\in[k]})$ //NIZK$^{\mathcal{R}}$.Prove
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}})$

$\underline{\mathsf{Verify}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m, \sigma)}$:
$(\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}}) \leftarrow \sigma$
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$ //NIZK$^{\mathcal{R}}$.Verify
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi, y) = 0$: Return 0
$(s_1, s_2) \leftarrow sk_{\mathsf{r}}$
If $u_1^{s_1} u_2^{s_2} \neq k_{\mathsf{r}}$: Return 0 //dHPS-KEM$^{\Sigma}$.Decap
Return 1

$\underline{\mathsf{TKGen}(\mathsf{pp}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$: //USPCE.TKGen
$(\widetilde{T}, \mathsf{S}, s) \leftarrow sk_{\mathsf{Ag}}$
Return $\mathsf{tk} := (\overline{\mathsf{H}}(m))^s$

$\underline{\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m, \sigma, \mathsf{tk})}$:
$(\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}}) \leftarrow \sigma$
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$ //NIZK$^{\mathcal{R}}$.Verify
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi, y) = 0$: Return 0
$(\alpha, \beta, s_1, s_2) \leftarrow sk_{\mathsf{J}}$
$k'_{\mathsf{J}} = u_1^{s_1} u_2^{s_2}$ //dHPS-KEM$^{\Sigma}$.Decap
If $\mathsf{tk} \in \mathbb{G}$: //USPCE.Dec
  $(U, V) \leftarrow c$, $k''_{\mathsf{J}} := V / e(\mathsf{tk}^{\beta}, U)$
  If $k'_{\mathsf{J}} = k''_{\mathsf{J}}$: Return 1
Else
  For $j \in [k]$:
    $k''_{\mathsf{J}} := S_j \cdot Q_j^{-\alpha}$
    If $k'_{\mathsf{J}} = k''_{\mathsf{J}}$: Return 1
Return 0

Fig. 20 Algorithm descriptions of Frank, Verify, TKGen, and Judge

USPCE is to obtain a set of k_{J}'s, and see if one of them matches the decapsulated key output by Decap. If the token is not empty, then the decryption of USPCE is to obtain a k_{J}, and see if it matches the decapsulated key output by Decap.

Three forging algorithms are shown in Fig. 21. It similar to the Frank algorithm, except that

– The ciphertext of HPS-KEM$^{\Sigma}$ is generated via Encap^*.
– If the key (of the receiver of the judge) is corrupted, then it calls the Decap algorithm to generate the corresponding key.
– If the key (of the receiver of the judge) is not corrupted, then it samples a random key from the key space or calls $\mathsf{SamEncK}$ to generate a key, which is depending on the exact case.
– It also needs to generate a NIZK proof. Due to the construction of the relation $\mathcal{R}$, it is guaranteed that the forger can also generate a valid proof, but using a different witness compared with the sender.

$\underline{\mathsf{Forge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$r, r' \leftarrow \mathbb{Z}_p^*$, $u_1 := h_1^r$, $u_2 := h_1^{r'}$, $k_{\mathsf{r}} \leftarrow \mathbb{G}_T$, $t_1, t_2 \leftarrow \mathbb{Z}_p^*$, $k_{\mathsf{J}} \leftarrow h_1^{t_1} h_2^{t_2}$
For each $j \in [k]$: //USPCE.Enc
  $\gamma_j \leftarrow \mathbb{Z}_p^*$, $Q_j := e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j}$, $S_j := e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot k_{\mathsf{J}}$
If $e(X, Y') \neq e(g, Y)$: Return $\perp$
$r \leftarrow \mathbb{Z}_p^*$, $c := (g^r, e(\overline{\mathsf{H}}(m), Y)^r \cdot k_{\mathsf{J}})$
$x \leftarrow (\perp, \perp, r, r', r'', t_1, t_2, (\gamma_j)_{j\in[k]})$
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}})$

$\underline{\mathsf{RForge}(\mathsf{pp}, pk_{\mathsf{s}}, sk_{\mathsf{r}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m)}$:
$r, r' \leftarrow \mathbb{Z}_p^*$, $u_1 := h_1^r$, $u_2 := h_1^{r'}$, $k_{\mathsf{r}} \leftarrow u_1^{s_1} u_2^{s_2}$, $t_1, t_2 \leftarrow \mathbb{Z}_p^*$, $k_{\mathsf{J}} \leftarrow h_1^{t_1} h_2^{t_2}$, $(s_1, s_2) \leftarrow sk_{\mathsf{r}}$
For each $j \in [k]$: //USPCE.Enc
  $\gamma_j \leftarrow \mathbb{Z}_p^*$, $Q_j := e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j}$, $S_j := e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot k_{\mathsf{J}}$
If $e(X, Y') \neq e(g, Y)$: Return $\perp$
$r \leftarrow \mathbb{Z}_p^*$, $c := (g^r, e(\overline{\mathsf{H}}(m), Y)^r \cdot k_{\mathsf{J}})$
$x \leftarrow (\perp, \perp, r, r', r'', t_1, t_2, (\gamma_j)_{j\in[k]})$
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}})$

$\underline{\mathsf{JForge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m)}$:
$r, r' \leftarrow \mathbb{Z}_p^*$, $u_1 := h_1^r$, $u_2 := h_1^{r'}$, $t_1, t_2 \leftarrow \mathbb{Z}_p^*$, $k_{\mathsf{r}} \leftarrow h_1^{t_1} h_2^{t_2}$, $(\alpha, s_1, s_2) \leftarrow sk_{\mathsf{J}}$, $k_{\mathsf{J}} \leftarrow u_1^{s_1} u_2^{s_2}$
For each $j \in [k]$: //USPCE.Enc
  $\gamma_j \leftarrow \mathbb{Z}_p^*$, $Q_j := e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j}$, $S_j := e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot k_{\mathsf{J}}$
If $e(X, Y') \neq e(g, Y)$: Return $\perp$
$r \leftarrow \mathbb{Z}_p^*$, $c := (g^r, e(\overline{\mathsf{H}}(m), Y)^r \cdot k_{\mathsf{J}})$
$x \leftarrow (s_1, s_2, r, r', r'', t_1, t_2, (\gamma_j)_{j\in[k]})$
$y \leftarrow (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c)$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{Prove}(pk_{\mathsf{r}}, y, x)$
Return $\sigma \leftarrow (\pi, u_1, u_2, (Q_j, S_j)_{j\in[k]}, c, k_{\mathsf{r}})$

Fig. 21 Algorithm descriptions of Forge, RForge and JForge

Changes of relation. The relation $\mathcal{R}$ is turned into Eq. (22).

$$
\begin{aligned}
\mathcal{R} = \{(\ & (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c = (U, V)), \\
& (s_1, s_2, r, r', r'', t_1, t_2, (\gamma_j)_{j\in[k]}) \ ) : \\
& (\ pk_{\mathsf{s}} = h_1^{s_1} h_2^{s_2} \wedge u_1 = h_1^{r} \wedge u_2 = h_2^{r} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot (pk'_{\mathsf{J}})^{r}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot (pk'_{\mathsf{J}})^{r}) \ ) \\
\vee \ & (\ u_1 = h_1^{r} \wedge u_2 = h_1^{r'} \wedge k_{\mathsf{r}} = h_1^{t_1} h_2^{t_2} \wedge pk'_{\mathsf{J}} = h_1^{s_1} h_2^{s_2} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot u_1^{s_1} u_2^{s_2}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot u_1^{s_1} u_2^{s_2}) \ ) \\
\vee \ & (\ u_1 = h_1^{r} \wedge u_2 = h_1^{r'} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot h_1^{t_1} h_2^{t_2}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot h_1^{t_1} h_2^{t_2}) \ ) \quad \}
\end{aligned}
\tag{22}
$$

It is clear that we can prove $\mathcal{R}$ via Sigma protocols, including Okamoto's Sigma protocol [Oka95], the Chaum-Pedersen protocol [CP92], Schnorr's Sigma protocol [Sch89] and its extension.

Here, we provide some explanation about it.

The relation comprises three sub-relations combined via the "OR" operation.

- The first part of the expression of $\mathcal{R}$ is similar to that in the general construction. If the first part is true, then it guarantees that:
  - (i) the proof generator knowing the secret key of the senders (i.e., ensuring accountability);
  - (ii) the ciphertext is well-formed;
  - (iii) different from the relation in the general construction, the well-formedness of the encapsulated key for the judge (i.e., k_{J}) is integrated into the well-formedness of the USPCE ciphertext. Then (ii) and (iii) further assures the receiver that both c and k_{J} can be successfully verified by the judge. In essence, when the receiver reports to the judge, the judge will readily accept the report.
- The second part of the expression of $\mathcal{R}$, is tailored for JForge.
  - Firstly, the forgers can generate an ill-formed ciphertext, knowing the discrete logarithm of u_1 and u_2 with base h_1.
  - Then, it proves that the proof generator knows the discrete logarithms of the encapsulated key k_{r} with two bases h_1 and h_2. Then given an ill-formed ciphertext, it is with negligible probability that the result of decapsulation, i.e., running Decap, equals to k_{r}. Otherwise $\log_{h_1} h_2$ can be computed.
  - Thirdly, it proves that the proof generator knows the secret key of the judge, i.e., knowing s_1 and s_2.

- Finally, it proves that the UPSCE ciphertext is well-formed. Note that, here the message is $u_1^{s_1} u_2^{s_2}$ (in fact, it is the decapsulated key for the judge via Decap), which means that given the ill-formed ciphertext, the decapsulated key of the judge equals to decryption result of USPCE ciphertext.

In all, it guarantees that the forger who can access to the judge's secret key, can generate proof for such sub-relation. In addition, it implies that the judge would accept the corresponding signature. It also guarantees that receiver would reject the signature (which is required for the unframeability).

– The third part of the expression of $\mathcal{R}$, is tailored for Forge and RForge.
  - Firstly, the forgers can generate an ill-formed ciphertext, knowing the discrete logarithm of u_1 and u_2 with base h_1.
  - Then, it proves that the UPSCE ciphertext is well-formed. Note that, here the message is $h_1^{t_1} h_2^{t_2}$. Thus, it guarantees that when decrypting the USPCE ciphertext, the judge can obtain $h_1^{t_1} h_2^{t_2}$ as the encapsulated key k_{J}. However, given an ill-formed ciphertext, it is with negligible probability that the result of decapsulation for the judge, i.e., running Decap, equals to k_{J}. Otherwise $\log_{h_1} h_2$ can be computed.

  In all, it guarantees that the forger who can or cannot access to the receiver's secret key, can generate proof for such sub-relation. In addition, it implies that the judge would not accept the corresponding signature, which is required for the unframeability.

Therefore, the relation $\mathcal{R}$ ensures that forgers can consistently generate a valid NIZK proof, thereby preserving deniability.

A discussion on the security analysis From the above description, we can summary the changes as follows

– The generation for the encapsulated key of the judge k_{J} is invoked by the $\mathsf{Encap}_{\mathrm{k}}$, instead of $\mathsf{dEncap}_{\mathrm{k}}$. In fact, $\mathsf{Encap}_{\mathrm{k}}$ offers universality, key unexplainability ans smoothness, while $\mathsf{dEncap}_{\mathrm{k}}$ offers the extended version of these properties. When invoking $\mathsf{Encap}_{\mathrm{k}}$, then we can replaced the extended version of these properties with the original version.
– The USPCE is to encrypt k_{J} not t. The encryption does not affect the security of the system. Since without knowing t, we cannot check if the judge's encapsulated key is correct via dDecap. Now it turns that without knowing k_{J}, we cannot check if the decapsulated key via Decap is correct.
– There are some changes in the relation $\mathcal{R}$. However, it achieves all the requirements as the relation in the general construction does.

Thus, our security proof for the general construction is still valid for the improved scheme. Here, we omit the detailed security proof.

Signature size. In this part, we conduct a comparison of the signature size between the improved scheme and the specific scheme derived from the general construction using concrete USPCE and dual HPS-KEM$^{\Sigma}$.

As shown in Table 1, following the general composition of Sigma protocols [BS20], the signature in the the scheme in Sec. 6 (a concrete scheme derived

Table 1: A comparison of signature size

Scheme	Signature size
Scheme in Sec. 6 *	$(2k+8)\times\lvert\mathbb{G}_T\rvert+1\times\lvert\mathbb{G}\rvert+(3k+14)\times\lvert\mathbb{Z}_p^*\rvert$
Scheme in Appendix H **	$(2k+4)\times\lvert\mathbb{G}_T\rvert+1\times\lvert\mathbb{G}\rvert+(3k+16)\times\lvert\mathbb{Z}_p^*\rvert$
Scheme in Appendix H using $\Sigma^{\mathcal{R}'}$ †	$(2k+4)\times\lvert\mathbb{G}_T\rvert+1\times\lvert\mathbb{G}\rvert+(k+14)\times\lvert\mathbb{Z}_p^*\rvert$

* The scheme in Sec. 6 is derived from the general construction in Sec. 6, plugging with the concrete USPCE and dual HPS-KEM$^{\Sigma}$

** Scheme in Appendix H is the concrete scheme shown in Fig. 19 to Fig. 21.

† Scheme in Appendix H using $\Sigma^{\mathcal{R}'}$ is the concrete construction obtained by applying $\Sigma^{\mathcal{R}'}$ in Fig. 22 and Fig. 23 to scheme in Appendix H.

from the general construction using concrete USPCE and dual HPS-KEM$^{\Sigma}$) comprises $(2k+8)$ elements in $\mathbb{G}_T$, 1 element in $\mathbb{G}$, and $(3k+14)$ elements in $\mathbb{Z}_p^*$, where k is the parameter for cuckoo hashing. On the other hand, the signature in the scheme in Appendix H (shown in Fig. 19 to Fig. 21, also following the general composition of Sigma protocols [BS20]) consists of $(2k+4)$ elements in $\mathbb{G}_T$, 1 element in $\mathbb{G}$, and $(3k+16)$ elements in $\mathbb{Z}_p^*$. Consequently, the signature of the scheme in Appendix H has 4 fewer elements in $\mathbb{G}_T$ but 2 more elements in $\mathbb{Z}_p^*$.

Here, we provide a more efficient Sigma protocol $\Sigma^{\mathcal{R}'}$ for $\mathcal{R}'$ (an equivalent relation of $\mathcal{R}$), such that the proof size is much smaller. Then, the signature in the scheme in Appendix H using $\Sigma^{\mathcal{R}'}$ consists of $(2k+4)$ elements in $\mathbb{G}_T$, 1 element in $\mathbb{G}$, and $(k+14)$ elements in $\mathbb{Z}_p^*$. Compared with the scheme in Sec. 6, the signature of the scheme in Appendix H using $\Sigma^{\mathcal{R}'}$ has 4 fewer elements in $\mathbb{G}_T$ and $2k$ fewer elements in $\mathbb{Z}_p^*$.

In the following, we describe the details of $\Sigma^{\mathcal{R}'}$. Firstly, we re-write $\mathcal{R}$ in Eq. (22) to the relation $\mathcal{R}'$ in Eq. (23). It is trivial to know that they are equivalent. Then, the protocols are shown in Fig. 22 and Fig. 23.

$$
\begin{aligned}
\mathcal{R}' = \{(\ & (\mathsf{pp}, m, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, k_{\mathsf{r}}, u_1, u_2, (Q_j, S_j)_{j\in[k]}, T, c=(U,V)), \\
& (s_1, s_2, s_3, s_4, r, r^*, r', r'', t_1, t_2, t_3, t_4, (\gamma_j)_{j\in[k]}) \ ) : \\
& (\ pk_{\mathsf{s}} = h_1^{s_1} h_2^{s_2} \wedge u_1 = h_1^{r} \wedge u_2 = h_2^{r} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot (pk'_{\mathsf{J}})^{r}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot (pk'_{\mathsf{J}})^{r}) \ ) \\
& \vee \ (\ u_1 = h_1^{r^*} \wedge u_2 = h_1^{r'} \wedge k_{\mathsf{r}} = h_1^{t_1} h_2^{t_2} \wedge pk'_{\mathsf{J}} = h_1^{s_3} h_2^{s_4} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot u_1^{s_3} u_2^{s_4}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot u_1^{s_3} u_2^{s_4}) \ ) \\
& \vee \ (\ u_1 = h_1^{r^*} \wedge u_2 = h_1^{r'} \\
& \quad \wedge_{j\in[k]} \ (Q_j = e(A', \widetilde{\mathsf{H}}(m))^{\gamma_j} \wedge S_j = e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot h_1^{t_3} h_2^{t_4}) \\
& \quad \wedge \ (U = g^{r''} \wedge V = e(\overline{\mathsf{H}}(m), Y)^{r''} \cdot h_1^{t_3} h_2^{t_4}) \ ) \quad \}
\end{aligned}
\tag{23}
$$

$\underline{P_1(y, x = (s_1, s_2, \bot, \bot, r, \bot, \bot, r'', \bot, \bot, \bot, \bot, (\gamma_j)_{j\in[k]}))}$:
If $x = (s_1, s_2, \bot, \bot, r, \bot, \bot, r'', \bot, \bot, \bot, \bot, (\gamma_j)_{j\in[k]})$:
$w' := (w_{s_1}, w_{s_2}, w_r, w_{r''}, (w_{\gamma_j})_{j\in[k]}) \leftarrow (\mathbb{Z}_p^*)^{k+4}$, $\mathsf{z}' := (\mathsf{z}_{s_3}, \mathsf{z}_{s_4}, \mathsf{z}_{r^*}, \mathsf{z}_{r'}, \mathsf{z}_{t_1}, \mathsf{z}_{t_2}, \mathsf{z}_{t_3}, \mathsf{z}_{t_4}) \leftarrow (\mathbb{Z}_p^*)^8$
$(\mathsf{cl}_2, \mathsf{cl}_3) \leftarrow (\mathbb{Z}_p^*)^2$, $\mathsf{cm}_{pk_\mathsf{s}} = h_1^{w_{s_1}} h_2^{w_{s_2}}$, $\mathsf{cm}_{u_1,1} = h_1^{w_r}$, $\mathsf{cm}_{u_2,1} = h_2^{w_r}$
For $j \in [k]$:
$\mathsf{cm}_{Q_j} = e(A', \widetilde{\mathsf{H}}(m))^{w_{\gamma_j}}$
$\mathsf{cm}_{S_j} = e(g, T[\mathsf{H}_j(m)])^{w_{\gamma_j}} \cdot (pk'_\mathsf{J})^{w_r + r(\mathsf{cl}_2+\mathsf{cl}_3)} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}_U = g^{w_{r''}}$, $\mathsf{cm}_V = e(\overline{\mathsf{H}}(m), Y)^{w_{r''}} \cdot (pk'_\mathsf{J})^{w_r + r(\mathsf{cl}_2+\mathsf{cl}_3)} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}_{u_1,23} = u_1^{\mathsf{cl}_2+\mathsf{cl}_3} h_1^{\mathsf{z}_{r^*}}$, $\mathsf{cm}_{u_2,23} = u_2^{\mathsf{cl}_2+\mathsf{cl}_3} h_2^{\mathsf{z}_{r'}}$, $\mathsf{cm}_{k_\mathsf{r}} = k_\mathsf{r}^{\mathsf{cl}_2} h_1^{\mathsf{z}_{t_1}} h_2^{\mathsf{z}_{t_2}}$, $\mathsf{cm}_{pk'_\mathsf{J}} = (pk'_\mathsf{J})^{\mathsf{cl}_2} h_1^{\mathsf{z}_{s_3}} h_2^{\mathsf{z}_{s_4}}$
$\mathsf{cm} \leftarrow (\mathsf{cm}_{pk_\mathsf{s}}, \mathsf{cm}_{u_1,1}, \mathsf{cm}_{u_2,1}, (\mathsf{cm}_{Q_j}, \mathsf{cm}_{S_j})_{j\in[k]}, \mathsf{cm}_U, \mathsf{cm}_V, \mathsf{cm}_{u_1,23}, \mathsf{cm}_{u_2,23}, \mathsf{cm}_{k_\mathsf{r}}, \mathsf{cm}_{pk'_\mathsf{J}})$
If $x = (\bot, \bot, s_3, s_4, \bot, r^*, r', r'', t_1, t_2, \bot, \bot, (\gamma_j)_{j\in[k]})$:
$w' := (w_{s_3}, w_{s_4}, w_{r^*}, w_{r'}, w_{r''}, w_{t_1}, w_{t_2}, (w_{\gamma_j})_{j\in[k]}) \leftarrow (\mathbb{Z}_p^*)^{k+7}$, $\mathsf{z}' := (\mathsf{z}_{s_1}, \mathsf{z}_{s_2}, \mathsf{z}_r, \mathsf{z}_{t_3}, \mathsf{z}_{t_4}) \leftarrow (\mathbb{Z}_p^*)^5$
$(\mathsf{cl}_1, \mathsf{cl}_3) \leftarrow (\mathbb{Z}_p^*)^2$, $\mathsf{cm}'_{pk_\mathsf{s}} = pk_\mathsf{s}^{\mathsf{cl}_1} h_1^{\mathsf{z}_{s_1}} h_2^{\mathsf{z}_{s_2}}$, $\mathsf{cm}'_{u_1,1} = u_1^{\mathsf{cl}_1} h_1^{\mathsf{z}_r}$, $\mathsf{cm}'_{u_2,1} = u_2^{\mathsf{cl}_1} h_2^{\mathsf{z}_r}$
For $j \in [k]$:
$\mathsf{cm}_{Q_j} = e(A', \widetilde{\mathsf{H}}(m))^{w_{\gamma_j}}$
$\mathsf{cm}_{S_j} = e(g, T[\mathsf{H}_j(m)])^{w_{\gamma_j}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot (u_1^{w_{s_3} + s_3(\mathsf{cl}_1+\mathsf{cl}_3)} u_2^{w_{s_4} + s_4(\mathsf{cl}_1+\mathsf{cl}_3)}) \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}_U = g^{w_{r''}}$, $\mathsf{cm}_V = e(\overline{\mathsf{H}}(m), Y)^{w_{r''}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot (u_1^{w_{s_3} + s_3(\mathsf{cl}_1+\mathsf{cl}_3)} u_2^{w_{s_4} + s_4(\mathsf{cl}_1+\mathsf{cl}_3)}) \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}_{u_1,23} = h_1^{w_{r^*}}$, $\mathsf{cm}_{u_2,23} = h_2^{w_{r'}}$, $\mathsf{cm}_{k_\mathsf{r}} = h_1^{w_{t_1}} h_2^{w_{t_2}}$, $\mathsf{cm}_{pk'_\mathsf{J}} = h_1^{w_{s_3}} h_2^{w_{s_4}}$
$\mathsf{cm} \leftarrow (\mathsf{cm}_{pk_\mathsf{s}}, \mathsf{cm}_{u_1,1}, \mathsf{cm}_{u_2,1}, (\mathsf{cm}_{Q_j}, \mathsf{cm}_{S_j})_{j\in[k]}, \mathsf{cm}_U, \mathsf{cm}_V, \mathsf{cm}_{u_1,23}, \mathsf{cm}_{u_2,23}, \mathsf{cm}_{k_\mathsf{r}}, \mathsf{cm}_{pk'_\mathsf{J}})$
If $x = (\bot, \bot, \bot, \bot, \bot, r^*, r', r'', \bot, \bot, t_3, t_4, (\gamma_j)_{j\in[k]})$:
$w' := (w_{r^*}, w_{r'}, w_{r''}, w_{t_3}, w_{t_4}, (w_{\gamma_j})_{j\in[k]}) \leftarrow (\mathbb{Z}_p^*)^{k+5}$, $\mathsf{z}' := (\mathsf{z}_{s_1}, \mathsf{z}_{s_2}, \mathsf{z}_{s_3}, \mathsf{z}_{s_4}, \mathsf{z}_r, \mathsf{z}_{t_1}, \mathsf{z}_{t_2}) \leftarrow (\mathbb{Z}_p^*)^7$
$(\mathsf{cl}_1, \mathsf{cl}_2) \leftarrow (\mathbb{Z}_p^*)^2$, $\mathsf{cm}'_{pk_\mathsf{s}} = pk_\mathsf{s}^{\mathsf{cl}_1} h_1^{\mathsf{z}_{s_1}} h_2^{\mathsf{z}_{s_2}}$, $\mathsf{cm}'_{u_1,1} = u_1^{\mathsf{cl}_1} h_1^{\mathsf{z}_r}$, $\mathsf{cm}'_{u_2,1} = u_2^{\mathsf{cl}_1} h_2^{\mathsf{z}_r}$
For $j \in [k]$:
$\mathsf{cm}_{Q_j} = e(A', \widetilde{\mathsf{H}}(m))^{w_{\gamma_j}}$
$\mathsf{cm}_{S_j} = e(g, T[\mathsf{H}_j(m)])^{w_{\gamma_j}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot (h_1^{w_{t_3} + t_3(\mathsf{cl}_1+\mathsf{cl}_2)} h_2^{w_{t_4} + t_4(\mathsf{cl}_1+\mathsf{cl}_2)})$
$\mathsf{cm}_U = g^{w_{r''}}$, $\mathsf{cm}_V = e(\overline{\mathsf{H}}(m), Y)^{w_{r''}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot (h_1^{w_{t_3} + t_3(\mathsf{cl}_1+\mathsf{cl}_2)} h_2^{w_{t_4} + t_4(\mathsf{cl}_1+\mathsf{cl}_2)})$
$\mathsf{cm}_{u_1,23} = h_1^{w_{r^*}}$, $\mathsf{cm}_{u_2,23} = h_2^{w_{r'}}$, $\mathsf{cm}_{k_\mathsf{r}} = k_\mathsf{r}^{\mathsf{cl}_2} h_1^{\mathsf{z}_{t_1}} h_2^{\mathsf{z}_{t_2}}$, $\mathsf{cm}_{pk'_\mathsf{J}} = (pk'_\mathsf{J})^{\mathsf{cl}_2} h_1^{\mathsf{z}_{s_3}} h_2^{\mathsf{z}_{s_4}}$
$\mathsf{cm} \leftarrow (\mathsf{cm}_{pk_\mathsf{s}}, \mathsf{cm}_{u_1,1}, \mathsf{cm}_{u_2,1}, (\mathsf{cm}_{Q_j}, \mathsf{cm}_{S_j})_{j\in[k]}, \mathsf{cm}_U, \mathsf{cm}_V, \mathsf{cm}_{u_1,23}, \mathsf{cm}_{u_2,23}, \mathsf{cm}_{k_\mathsf{r}}, \mathsf{cm}_{pk'_\mathsf{J}})$
Send cm to the verifier.

$\underline{V_1(\mathsf{cm})}$: Send $\mathsf{cl} \leftarrow \mathbb{Z}_p^*$ to the prover.

Fig. 22 P_1 and V_1 of Sigma protocol $\Sigma^{\mathcal{R}'}$ for relation $\mathcal{R}'$

$\underline{\text{Security analysis of } \Sigma^{\mathcal{R}'}}$. The verification of the verifier, i.e., V_2 would output 1, when all commitments for all statements equal to those sent by the prover P_1. Here, we pick cm_{S_j} for example. When $x = (s_1, s_2, \bot, \bot, r, \bot, \bot, r'', \bot, \bot, \bot, \bot, (\gamma_j)_{j\in[k]})$,

$$\begin{aligned}
\mathsf{cm}'_{S_j} &= S_j^{\mathsf{cl}} \cdot e(g, T[\mathsf{H}_j(m)])^{\mathsf{z}_{\gamma_j}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}} \\
&= (e(g, T[\mathsf{H}_j(m)])^{\gamma_j} \cdot (pk'_\mathsf{J})^r)^{\mathsf{cl}} \cdot e(g, T[\mathsf{H}_j(m)])^{\mathsf{z}_{\gamma_j}} \\
&\quad \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}} \\
&= e(g, T[\mathsf{H}_j(m)])^{\gamma_j \cdot \mathsf{cl} + \mathsf{z}_{\gamma_j}} \cdot (pk'_\mathsf{J})^{r \cdot \mathsf{cl} + \mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}
\end{aligned}$$

$P_2(\mathsf{cm}, \mathsf{cl}, y, x, \mathsf{aux})$:
If $x = (s_1, s_2, \bot, \bot, r, \bot, \bot, r'', \bot, \bot, \bot, \bot, (\gamma_j)_{j\in[k]})$:
  $(w', \mathsf{cl}_2, \mathsf{cl}_3, \mathsf{z}') \leftarrow \mathsf{aux}$
  $\mathsf{cl}_1 = \mathsf{cl} - \mathsf{cl}_1 - \mathsf{cl}_2$, $\mathsf{z}_{s_1} = w_{s_1} - \mathsf{cl}_1 \cdot s_1$, $\mathsf{z}_{s_2} = w_{s_2} - \mathsf{cl}_1 \cdot s_2$, $\mathsf{z}_r = w_r - \mathsf{cl}_1 \cdot r$, $\mathsf{z}_{r''} = w_{r''} - \mathsf{cl} \cdot r''$
  For $j \in [k]$: $\mathsf{z}_{\gamma_j} := w_{\gamma_j} - \mathsf{cl} \cdot \gamma_j$
  $\mathsf{z} \leftarrow (\mathsf{cl}_1, \mathsf{cl}_2, \mathsf{z}_{s_1}, \mathsf{z}_{s_2}, \mathsf{z}_r, \mathsf{z}_{r''}, (\mathsf{z}_{\gamma_j})_{j\in[k]}, \mathsf{z}')$
If $x = (\bot, \bot, s_3, s_4, \bot, r^*, r', r'', t_1, t_2, \bot, \bot, (\gamma_j)_{j\in[k]})$:
  $(w', \mathsf{cl}_1, \mathsf{cl}_3, \mathsf{z}') \leftarrow \mathsf{aux}$
  $\mathsf{cl}_2 = \mathsf{cl} - \mathsf{cl}_1 - \mathsf{cl}_3$, $\mathsf{z}_{s_3} = w_{s_3} - \mathsf{cl}_2 \cdot s_3$, $\mathsf{z}_{s_4} = w_{s_4} - \mathsf{cl}_2 \cdot s_4$, $\mathsf{z}_{r^*} = w_{r^*} - (\mathsf{cl}_2 + \mathsf{cl}_3) \cdot r^*$
  $\mathsf{z}_{r'} = w_{r'} - (\mathsf{cl}_2 + \mathsf{cl}_3) \cdot r'$, $\mathsf{z}_{r''} = w_{r''} - \mathsf{cl} \cdot r''$, $\mathsf{z}_{t_1} = w_{t_1} - \mathsf{cl}_2 \cdot t_1$, $\mathsf{z}_{t_2} = w_{t_2} - \mathsf{cl}_2 \cdot t_2$
  For $j \in [k]$: $\mathsf{z}_{\gamma_j} := w_{\gamma_j} - \mathsf{cl} \cdot \gamma_j$
  $\mathsf{z} \leftarrow (\mathsf{cl}_1, \mathsf{cl}_2, \mathsf{z}_{s_3}, \mathsf{z}_{s_4}, \mathsf{z}_{r^*}, \mathsf{z}_{r'}, \mathsf{z}_{r''}, \mathsf{z}_{t_1}, \mathsf{z}_{t_2}, (\mathsf{z}_{\gamma_j})_{j\in[k]}, \mathsf{z}')$
If $x = (\bot, \bot, \bot, \bot, \bot, r^*, r', r'', \bot, \bot, t_3, t_4, (\gamma_j)_{j\in[k]})$:
  $(w', \mathsf{cl}_1, \mathsf{cl}_2, \mathsf{z}') \leftarrow \mathsf{aux}$
  $\mathsf{cl}_3 = \mathsf{cl} - \mathsf{cl}_1 - \mathsf{cl}_2$, $\mathsf{z}_{r^*} = w_{r^*} - (\mathsf{cl}_2 + \mathsf{cl}_3) \cdot r^*$
  $\mathsf{z}_{r'} = w_{r'} - (\mathsf{cl}_2 + \mathsf{cl}_3) \cdot r'$, $\mathsf{z}_{r''} = w_{r''} - \mathsf{cl} \cdot r''$, $\mathsf{z}_{t_3} = w_{t_3} - \mathsf{cl}_3 \cdot t_3$, $\mathsf{z}_{t_4} = w_{t_4} - \mathsf{cl}_3 \cdot t_4$
  For $j \in [k]$: $\mathsf{z}_{\gamma_j} := w_{\gamma_j} - \mathsf{cl} \cdot \gamma_j$
  $\mathsf{z} \leftarrow (\mathsf{cl}_1, \mathsf{cl}_2, \mathsf{z}_{r^*}, \mathsf{z}_{r'}, \mathsf{z}_{r''}, \mathsf{z}_{t_3}, \mathsf{z}_{t_4}, (\mathsf{z}_{\gamma_j})_{j\in[k]}, \mathsf{z}')$
Send z to the verifier.

$V_2(y, \mathsf{cm}, \mathsf{cl}, \mathsf{z})$:
$(\mathsf{cl}_1, \mathsf{cl}_2, \mathsf{z}_{s_1}, \mathsf{z}_{s_2}, \mathsf{z}_r, \mathsf{z}_{r''}, (\mathsf{z}_{\gamma_j})_{j\in[k]}, \mathsf{z}') \leftarrow \mathsf{z}$, $\mathsf{cl}_3 = \mathsf{cl} - \mathsf{cl}_1 - \mathsf{cl}_2$
$\mathsf{cm}'_{pk_\mathsf{s}} = pk_\mathsf{s}^{\mathsf{cl}_1} h_1^{\mathsf{z}_{s_1}} h_2^{\mathsf{z}_{s_2}}$, $\mathsf{cm}'_{u_1,1} = u_1^{\mathsf{cl}_1} h_1^{\mathsf{z}_r}$, $\mathsf{cm}'_{u_2,1} = u_2^{\mathsf{cl}_1} h_2^{\mathsf{z}_r}$
For $j \in [k]$: $\mathsf{cm}'_{Q_j} = Q_j^{\mathsf{cl}} e(A', \widetilde{\mathsf{H}}(m))^{\mathsf{z}_{\gamma_j}}$, $\mathsf{cm}'_{S_j} = S_j^{\mathsf{cl}} \cdot e(g, T[\mathsf{H}_j(m)])^{\mathsf{z}_{\gamma_j}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}'_U = U^{\mathsf{cl}} g^{\mathsf{z}_{r''}}$, $\mathsf{cm}'_V = V^{\mathsf{cl}} \cdot e(\overline{\mathsf{H}}(m), Y)^{\mathsf{z}_{r''}} \cdot (pk'_\mathsf{J})^{\mathsf{z}_r} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}}$
$\mathsf{cm}'_{u_1,23} = u_1^{\mathsf{cl}_2+\mathsf{cl}_3} h_1^{\mathsf{z}_{r^*}}$, $\mathsf{cm}'_{u_2,23} = u_2^{\mathsf{cl}_2+\mathsf{cl}_3} h_2^{\mathsf{z}_{r'}}$, $\mathsf{cm}'_{k_\mathsf{r}} = k_\mathsf{r}^{\mathsf{cl}_2} h_1^{\mathsf{z}_{t_1}} h_2^{\mathsf{z}_{t_2}}$, $\mathsf{cm}'_{pk'_\mathsf{J}} = (pk'_\mathsf{J})^{\mathsf{cl}_2} h_1^{\mathsf{z}_{s_3}} h_2^{\mathsf{z}_{s_4}}$
$\mathsf{cm}' \leftarrow (\mathsf{cm}_{pk_\mathsf{s}}, \mathsf{cm}_{u_1,1}, \mathsf{cm}_{u_2,1}, (\mathsf{cm}_{Q_j}, \mathsf{cm}_{S_j})_{j\in[k]}, \mathsf{cm}_U, \mathsf{cm}_V, \mathsf{cm}_{u_1,23}, \mathsf{cm}_{u_2,23}, \mathsf{cm}_{k_\mathsf{r}}, \mathsf{cm}_{pk'_\mathsf{J}})$
Return $(\mathsf{cm} = \mathsf{cm}')$

Fig. 23 P_2 and V_2 of Sigma protocol $\Sigma^{\mathcal{R}'}$ for relation $\mathcal{R}'$

$$
\begin{aligned}
&= e(g, T[\mathsf{H}_j(m)])^{w_{\gamma_j}} \cdot (pk'_\mathsf{J})^{r\cdot\mathsf{cl}_1 + \mathsf{z}_r + r(\mathsf{cl}_2+\mathsf{cl}_3)} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}} \ /\!\!/\ \mathsf{z}_{\gamma_j} = w_{\gamma_j} - \mathsf{cl} \cdot \gamma_j \\
&= e(g, T[\mathsf{H}_j(m)])^{w_{\gamma_j}} \cdot (pk'_\mathsf{J})^{w_r + r(\mathsf{cl}_2+\mathsf{cl}_3)} \cdot u_1^{\mathsf{z}_{s_3}} u_2^{\mathsf{z}_{s_4}} \cdot h_1^{\mathsf{z}_{t_3}} h_2^{\mathsf{z}_{t_4}} \ /\!\!/\ \mathsf{z}_r = w_r - \mathsf{cl} \cdot r \\
&= \mathsf{cm}_{S_j}
\end{aligned}
$$

Similar computation can be applied to other cases (i.e., when $x = (\bot, \bot, s_3, s_4, \bot, r^*, r', r'', t_1, t_2, \bot, \bot, (\gamma_j)_{j\in[k]})$ and when $x = (\bot, \bot, \bot, \bot, \bot, r^*, r', r'', \bot, \bot, t_3, t_4, (\gamma_j)_{j\in[k]})$) and other commitments for other statements.

Thus, we can conclude that the Sigma protocol $\Sigma^{\mathcal{R}'}$ is correct.

It can be trivially to check that the Sigma protocol $\Sigma^{\mathcal{R}'}$ satisfies knowledge soundness and special HVZK, since it is similar to many classic Sigma protocols, e.g., Schnorr's Sigma protocol [Sch89]. Here, we omit the detailed analysis.

I Discussions on one-time token generation

In this paper, our primary focus is dedicated to the moderation of illegal messages. It is crucial to acknowledge that certain messages, such as those involving harassment or containing phishing links, may not universally fall under the category of illegal messages for all users. Consequently, these messages might not be encompassed within the predefined set designated by the agency. This presents a challenge for the moderator as determining the sender's identity becomes problematic in such scenarios.

To address this challenge, one potential approach is to empower the agency to generate a one-time token tailored for a specific MAMF signature and message, especially for those not included in the predefined set. This capability allows the moderator to conduct content moderation for that particular signature and message. In the following, we present some ideas, leveraging the inherent flexibility of our USPCE.

I.1 A new USPCE

Changes in definitions. It is clear that one more algorithm for one-time token generation in USPCE is required.

- $\mathsf{tk} \leftarrow \mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}, msk, \mathsf{ct}, x)$: The token algorithm takes as input the public parameter pp, the master secret key msk, a ciphertext ct and an element x, and outputs a token tk for ct and x.

The correctness of USPCE would be re-defined as follows.

Definition 38. (Correctness). *A universal set pre-constrained encryption scheme* $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{TKGen}_{\mathrm{one}}, \mathsf{TKGen}, \mathsf{Dec})$ *is* correct*, if for any* $\lambda \in \mathbb{N}$*, for any set* $\mathsf{S} \subset \mathcal{U}$*, and for all* $m \in \mathcal{M}$*, it holds that,*

- *when* $x \in \mathsf{S}$*:*

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S}) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)\end{array} : m \in S_{\mathrm{m}} = \mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \perp)\right] = 1 - \mathsf{negl}(\lambda);$$

- *when* $x \notin \mathsf{S}$*:*

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S}) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m) \\ \mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, msk, x)\end{array} : m = \mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \mathsf{tk})\right] = 1 - \mathsf{negl}(\lambda).$$

and

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S}) \\ (pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m) \\ \mathsf{tk} \leftarrow \mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}, msk, \mathsf{ct}, x)\end{array} : m = \mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \mathsf{tk})\right] = 1 - \mathsf{negl}(\lambda),$$

For other security properties, there are changes only in confidentiality against user.

Definition 39. (Confidentiality against users). *An USPCE scheme* USPCE *has* confidentiality against users*, if for any set* $\mathsf{S} \subseteq \mathcal{U}$ *and any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, its advantage*

$$\mathbf{Adv}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda) := |\Pr[\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ *is defined in Fig. 24.*

$\underline{\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)\text{:}}$
$b \leftarrow \{0,1\}$, $(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S})$, $Q_{\mathrm{x}} := \emptyset$
$U_{\mathrm{x}} := \emptyset$, $U_{\mathrm{ct,x}} := \emptyset$
$(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap})$, $(m_0, m_1, x^*, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1^{\mathcal{O}}(\mathsf{pp}, pk, sk)$
If $(x^* \notin \mathcal{U}) \vee (x^* \in \mathsf{S}) \vee (x^* \in Q_{\mathrm{x}})$: Return $\perp$
$U_{\mathrm{x}} \leftarrow U_{\mathrm{x}} \cup \{x^*\}$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x^*, m_b)$, $b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(\mathsf{ct}, st_{\mathcal{A}})$
$U_{\mathrm{ct,x}} \leftarrow U_{\mathrm{ct,x}} \cup \{(\mathsf{ct}, x^*)\}$
Return $(b' = b)$

$\underline{\mathcal{O}^{\mathsf{TKGen}}(x')\text{:}}$
If $x' \in U_{\mathrm{x}}$: Return $\perp$
$Q_{\mathrm{x}} \leftarrow Q_{\mathrm{x}} \cup \{x'\}$
Return $\mathsf{TKGen}(\mathsf{pp}, msk, x')$

$\underline{\mathcal{O}^{\mathsf{TKGen}_{\mathrm{one}}}(\mathsf{ct}', x')\text{:}}$
If $(\mathsf{ct}', x') \in U_{\mathrm{ct,x}}$: Return $\perp$
Return $\mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}, msk, \mathsf{ct}', x')$

Fig. 24 Games for defining confidentiality against users of USPCE

Changes in the concrete USPCE construction. In Fig. 25, we modify the USPCE to support one-time token generation.

The main idea is as follows. Similar to the ciphertext for some item x, we take the ciphertext (i.e., $(Q_j, S_j)_{j\in[k]}$ and c) as a new token x', then adopt a similar method to prepare the ciphertext c' for x'. After that, similar to the token generation for x, we can generate the token for x', i.e., for $(Q_j, S_j)_{j\in[k]}$, c etc. Note that, in order to check the integrity of $(Q_j, S_j)_{j\in[k]}$, c and c', we also prepare another ciphertext c'', which takes the hash value of $(Q_j, S_j)_{j\in[k]}$, c and c'' as the exponent. Then we can check if c'' is correct by two bilinear map computations.

The concrete relation $\mathcal{R}_{\mathsf{ct}}$ is also shown in Fig. 25. It is clear that we can prove the relation $\mathcal{R}_{\mathsf{ct}}$ by combining Schnorr's Sigma protocol [Sch89], the Chaum-Pedersen protocol [CP92] and the Sigma protocol in Appendix F, with the "AND" operation in [BS20] and "AND-EQUAL$_l$" operation proposed in Appendix G.

The correctness analysis is as follows.

For any $\mathsf{S} \subset \mathcal{U}$, any $(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S})$, any $(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap})$, and any $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)$,

- when $x \in \mathsf{S}$, the properties of cuckoo hashing guarantee that x is inserted in one of locations (e.g., $\mathsf{H}_1(x), \ldots, \mathsf{H}_k(x)$) in T_{S}. Assuming x is located at $\mathsf{H}_j(x)$ in the table, we obtain $\widetilde{\mathsf{H}}(x) = \widetilde{T}[\mathsf{H}_j(x)]$. Hence,

$$S_j \cdot Q_j^{-\alpha} = (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot (\ e(A', \widetilde{\mathsf{H}}(x))^{\gamma_j}\ )^{-\alpha}$$

$\underline{\mathsf{Setup}(\lambda, \mathsf{S})}$:
$(e, \mathbb{G}, \mathbb{G}_T, g, p) \leftarrow \mathsf{GenG}(\lambda)$ //g is the generator of $\mathbb{G}$ with order p.
Choose another two generators of $\mathbb{G}$ with order p: g_1 and h.
Choose hash functions $\widetilde{\mathsf{H}}$, $\overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, $\hat{\mathsf{H}} : \{0,1\}^* \to \mathbb{G}^*$, and $\check{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^*$.
$(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$ //$n = \mathsf{poly}(\lambda)$ and $|T_{\mathsf{init}}| = n' = \mathsf{poly}(n) = \mathsf{poly}(\lambda)$.
//$\mathsf{pp}_{\mathsf{CH}}$ contains k random hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$, T_{init} is the hash table, ST is the stash.
$(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, $\alpha' \leftarrow \mathbb{Z}_p^*$, $A' := g^{\alpha'}$, $s \leftarrow \mathbb{Z}_p^*$, $Y' := g^s$, $s_1 \leftarrow \mathbb{Z}_p^*$, $Y_1' := g^{s_1}$
Initialize two empty tables $\widetilde{T}, T'$ with length n'.
For each $i \in [n']$:
  If $T_{\mathsf{S}}[i] = \perp$: $\widetilde{T}[i] \leftarrow \mathbb{G}$, $T'[i] := (\widetilde{T}[i])^{\alpha'}$
  Else $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$, $T'[i] := (\widetilde{T}[i])^{\alpha'}$
Return $(\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, g_1, h, p, \widetilde{\mathsf{H}}, \overline{\mathsf{H}}, \hat{\mathsf{H}}, \check{\mathsf{H}}, Y', Y_1', A', \mathsf{pp}_{\mathsf{CH}}), \mathsf{ap} = T', msk = (\widetilde{T}, \mathsf{S}, s, s_1))$

$\underline{\mathsf{KG}(\mathsf{pp}, \mathsf{ap})}$: //$n' = |T|$
$\alpha \leftarrow \mathbb{Z}_p^*$, $\beta \leftarrow \mathbb{Z}_p^*$, $X := g^\beta, Y := (Y')^\beta$
$\beta_1 \leftarrow \mathbb{Z}_p^*$, $X_1 := g^{\beta_1}, Y_1 := (Y_1')^{\beta_1}$
For each $i \in [n']$: $T[i] := e(g, T'[i])^\alpha$
Return $(pk = (T, X, Y, X_1, Y_1), sk = (\alpha, \beta, \beta_1))$

$\underline{\mathsf{TKGen}(\mathsf{pp}, msk, x)}$:
$(\widetilde{T}, \mathsf{S}, s, s_1) \leftarrow msk$
Return $\mathsf{tk} := (\overline{\mathsf{H}}(x))^s$

$\underline{\mathsf{Enc}(\mathsf{pp}, pk, x, m)}$:
For each $j \in [k]$:
  $\gamma_j \leftarrow \mathbb{Z}_p^*$, $Q_j := e(A', \widetilde{\mathsf{H}}(x))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m$
If $e(X, Y') \neq e(g, Y)$: Return $\perp$
If $e(X_1, Y_1') \neq e(g, Y_1)$: Return $\perp$
$r \leftarrow \mathbb{Z}_p^*$, $c := (U = g^r, V = e(\overline{\mathsf{H}}(x), Y)^r \cdot m)$
$r' \leftarrow \mathbb{Z}_p^*$, $U' = g^{r'}$, $x' \leftarrow (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$
$V' := e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m$
$u \leftarrow \check{\mathsf{H}}(x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$, $c'' = (g_1^u \cdot h)^{r'}$
Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'', x)$

$\underline{\mathsf{Dec}(\mathsf{pp}, sk, \mathsf{ct}, \mathsf{tk})}$:
$((Q_j, S_j)_{j\in[k]}, c, c', c'', x) \leftarrow \mathsf{ct}$, $(\alpha, \beta, \beta_1) \leftarrow sk$
If $\mathsf{tk} \in \mathbb{G}$:
  $(U, V) \leftarrow c$, $(U', V') \leftarrow c'$
  $x' \leftarrow (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$
  If $e(\mathsf{tk}, g) = e(\overline{\mathsf{H}}(x), Y')$:
    Return $m := V / e(\mathsf{tk}^\beta, U)$
  If $e(\mathsf{tk}, g) = e(\hat{\mathsf{H}}(x'), Y_1')$:
    Return $m := V' / e(\mathsf{tk}^{\beta_1}, U')$
  Else Return $\perp$
Else
  For $j \in [k]$: $m_j := S_j \cdot Q_j^{-\alpha}$
  Return $\{m_1, \cdots, m_k\}$

$\underline{\mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}, msk, \mathsf{ct}, x)}$:
$((Q_j, S_j)_{j\in[k]}, c = (U, V), c' = (U', V'), c'', x) \leftarrow \mathsf{ct}$
$x' \leftarrow (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$
$u \leftarrow \check{\mathsf{H}}(x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c')$
If $e(g, c'') \neq e(U', g_1^u \cdot h)$: Return $\perp$
$(\widetilde{T}, \mathsf{S}, s, s_1) \leftarrow msk$
Return $\mathsf{tk} := (\hat{\mathsf{H}}(x'))^{s_1}$

$\mathcal{R}_{\mathsf{ct}} = \{((\mathsf{pp}, pk, x, (Q_j, S_j)_{j\in[k]}, c = (U, V), c' = (U', V'), c''), ((\gamma_j)_{j\in[k]}, r, r', m)) :$
$\wedge_{j\in[k]} (Q_j = e(A', \widetilde{\mathsf{H}}(x))^{\gamma_j} \wedge S_j = (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m) \wedge (U = g^r \wedge V = e(\overline{\mathsf{H}}(x), Y)^r \cdot m)$
$\wedge (U' = g^{r'} \wedge V' = e(\hat{\mathsf{H}}(x'), Y')^{r'} \cdot m) \wedge c'' = (g_1^u \cdot h)^{r'}\}$

Fig. 25 A concrete construction of USPCE with one-time token (Here, we assume that the $\mathcal{M} \subseteq \mathbb{G}_T$.)

$$= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g^{\alpha'}, \widetilde{\mathsf{H}}(x))^{-\alpha\gamma_j}$$

$$
\begin{aligned}
&= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, (\widetilde{\mathsf{H}}(x))^{\alpha'})^{-\alpha\gamma_j} \\
&= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, (\widetilde{T}[\mathsf{H}_j(x)])^{\alpha'})^{-\alpha\gamma_j} \\
&= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot e(g, T'[\mathsf{H}_j(x)])^{-\alpha\gamma_j} \\
&= (T[\mathsf{H}_j(x)])^{\gamma_j} \cdot m \cdot ((T[\mathsf{H}_j(x)])^{\gamma_j})^{-1} \\
&= m
\end{aligned}
$$

Thus, it holds that $m \in S_{\mathrm{m}} = \{m_1, \cdots, m_k\}$.

– when $x \notin \mathsf{S}$, for $\mathsf{tk} \leftarrow \mathsf{TKGen}(\mathsf{pp}, msk, x)$, we obtain

$$
\begin{aligned}
V/e(\mathsf{tk}^{\beta}, U) &= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e((\overline{\mathsf{H}}(x))^{s\beta}, g^r) \\
&= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), (g^s)^{\beta})^r \\
&= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), (Y')^{\beta})^r \\
&= e(\overline{\mathsf{H}}(x), Y)^r \cdot m/e(\overline{\mathsf{H}}(x), Y)^r \\
&= m
\end{aligned}
$$

Thus, we can obtain m.

– When $x \notin \mathsf{S}$, $(\mathsf{pp}, \mathsf{ap}, msk) \leftarrow \mathsf{Setup}(\lambda, \mathsf{S})$, $(pk, sk) \leftarrow \mathsf{KG}(\mathsf{pp}, \mathsf{ap})$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, pk, x, m)$, $\mathsf{tk} \leftarrow \mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}, msk, \mathsf{ct}, x)$. Then it holds that,

$$
\begin{aligned}
V'/e(\mathsf{tk}^{\beta_1}, U') &= e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m/e((\hat{\mathsf{H}}(x'))^{s_1 \cdot \beta_1}, g^{r'}) \\
&= e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m/e(\hat{\mathsf{H}}(x'), (g^{s_1})^{\beta_1})^{r'} \\
&= e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m/e(\hat{\mathsf{H}}(x'), (Y_1')^{\beta_1})^{r'} \\
&= e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m/e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \\
&= m
\end{aligned}
$$

Thus, we can obtain m.

Therefore, the USPCE in Fig. 25 is correct.

As shown in Fig. 4, confidentiality against authority and confidentiality of sets do not have token generation oracles. Thus, the security proof of the USPCE in Fig. 25 for confidentiality against authority and confidentiality of sets remains the same.

Here, we only provide the proof for confidentiality against user.

Theorem 7. USPCE *in Fig. 25 achieves confidentiality against users.*

Proof. We use a sequence of games to show that USPCE in Fig. 25 satisfies confidentiality against users.

Game $\mathbf{G}_0$: This is the game $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ defined in Fig. 24.

Game $\mathbf{G}_{\varrho}$ $(1 \leq \varrho \leq k)$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

– Pick $b \in \{0, 1\}$ randomly.

– For $1 \leq j \leq \varrho$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– For $\varrho < j \leq k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j = e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j = (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
– Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c = (U = g^r, V = e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
– Choose $r' \leftarrow \mathbb{Z}_p^*$ randomly, and compute $U' = g^{r'}$, $V' = e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m_b$, $c'' = (g_1^u \cdot h)^{r'}$, where $x' = (x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'')$.

Game $\mathbf{G}_{k+1}$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

– For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– Choose $U \leftarrow \mathbb{G}$, $V \leftarrow \mathbb{G}_T$ randomly, and set $c = (U, V)$.
– Choose $r' \leftarrow \mathbb{Z}_p^*$ randomly, and compute $U' = g^{r'}$, $V' = e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m_b$, $c'' = (g_1^u \cdot h)^{r'}$, where $x' = (x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'')$.

Game $\mathbf{G}_{k+2}$: This game is the same as $\mathbf{G}_0$ except that the challenge ciphertext ct is generated as follows.

– For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
– Choose $U \leftarrow \mathbb{G}$, $V \leftarrow \mathbb{G}_T$ randomly, and set $c = (U, V)$.
– Choose $r' \leftarrow \mathbb{Z}_p^*$, $V' \leftarrow \mathbb{G}_T$ randomly, and compute $U' = g^{r'}$, $c'' = (g_1^u \cdot h)^{r'}$, where $x' = (x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$.
– Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'')$.

We prove these games are indistinguishable in the following lemmas. It is clear that the adversary has no advantage in **Game $\mathbf{G}_{k+2}$**. Therefore, we conclude that the advantage of the adversary in $\mathbf{G}^{\text{conf-u}}_{\mathsf{USPCE},\mathcal{A},\mathsf{S}}(\lambda)$ defined in Fig. 24 is negligible.

Lemma 14. *If the hash function $\widetilde{\mathsf{H}}$ is a random oracle and the DBDH assumption holds, then for $1 \leq \varrho \leq k$, **Game $\mathbf{G}_{\varrho-1}$** and **Game $\mathbf{G}_{\varrho}$** are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game $\mathbf{G}_{\varrho-1}$** and **Game $\mathbf{G}_{\varrho}$** with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game $\mathbf{G}_{\varrho-1}$** or **Game $\mathbf{G}_{\varrho}$** as follows.

$\mathcal{B}$ first chooses hash functions $\overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, $\hat{\mathsf{H}} : \{0,1\}^* \to \mathbb{G}^*$, $\check{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. $\mathcal{B}$ also maintains a list $L_{\widetilde{\mathsf{H}}}$, where the list is empty initially. Then, $\mathcal{B}$ sets $A' = g^{z_1}$ and computes

$Y' = g^s,\ Y'_1 = g^{s_1}$, where $s, s_1 \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, $\mathcal{B}$ chooses $t_i \leftarrow \mathbb{Z}_p^*$ and computes $\widetilde{T}[i] = (g^{z_2})^{t_i}$. Note that, if $T_{\mathsf{S}}[i] \neq \perp$, $\mathcal{B}$ sets $\widetilde{\mathsf{H}}(T_{\mathsf{S}}[i]) = (g^{z_2})^{t_i}$ implicitly and adds the record $(T_{\mathsf{S}}[i], t_i, (g^{z_2})^{t_i})$ to the list $L_{\widetilde{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\widetilde{\mathsf{H}}$. Next, $\mathcal{B}$ choose $\alpha, \beta, \beta_1 \leftarrow \mathbb{Z}_p^*$ and computes $X = g^\beta, Y = (Y')^\beta,\ X_1 = g^{\beta_1}, Y_1 = (Y'_1)^{\beta_1}$. For each $i \in [n']$, it computes $T[i] = e(A', (g^{z_2})^{t_i})^\alpha$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, g_1, h, p, \overline{\mathsf{H}}, \hat{\mathsf{H}}, \check{\mathsf{H}}, Y', Y'_1, A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y, X_1, Y_1)$, $sk = (\alpha, \beta, \beta_1)$ to the adversary $\mathcal{A}$, where $g_1, h \leftarrow \mathbb{G}$.

$\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as follows:

- $\widetilde{\mathsf{H}}$ query on x: If there is some $(x, t_x \in \mathbb{Z}_p^*, h_x \in \mathbb{G}) \in L_{\widetilde{\mathsf{H}}}$, $\mathcal{B}$ returns h; otherwise, $\mathcal{B}$ samples $t_x \leftarrow \mathbb{Z}_p^*$, adds (x, t_x, g^{t_x}) to $L_{\widetilde{\mathsf{H}}}$, and returns g^{t_x}.
- TKGen query on x: $\mathcal{B}$ returns $(\overline{\mathsf{H}}(x))^s$.
- $\mathsf{TKGen}_{\text{one}}$ query on $(\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c = (U, V), c' = (U', V'), c''),\ x)$: $\mathcal{B}$ sets $x' = (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c')$. If $e(g, c'') \neq e(U', g_1^u \cdot h)$, return $\perp$; else return $\mathsf{tk} = (\hat{\mathsf{H}}(x'))^{s_1}$

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^* such that $x^* \notin \mathsf{S}$. $\mathcal{B}$ picks $b \in \{0, 1\}$ randomly and proceeds as follows.

- For $1 \leq j \leq \varrho - 1$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
- For $j = \varrho$, find some $(x^*, t_{x^*}, g^{t_{x^*}})$ in $L_{\widetilde{\mathsf{H}}}$, and set $\gamma_j = z_3$ implicitly. Then, let $i = \mathsf{H}_j(x^*)$ and compute $Q_j = e(A', g^{z_3})^{t_{x^*}} = e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j = Z^{\alpha t_i} \cdot m_b$. Note that, if $Z = e(g, g)^{z_1 z_2 z_3}$, then $S_j = (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then S_j also is a random element of $\mathbb{G}_T$.
- For $\varrho < j \leq k$, choose $\gamma_j \leftarrow \mathbb{Z}_p^*$ randomly, and compute $Q_j := e(A', \widetilde{\mathsf{H}}(x^*))^{\gamma_j}$, $S_j := (T[\mathsf{H}_j(x^*)])^{\gamma_j} \cdot m_b$.
- Choose $r \leftarrow \mathbb{Z}_p^*$ randomly, and compute $c := (g^r, e(\overline{\mathsf{H}}(x^*), Y)^r \cdot m_b)$.
- Choose $r' \leftarrow \mathbb{Z}_p^*$ randomly, and compute $U' = g^{r'}$, $V' = e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m_b$, $c'' = (g_1^u \cdot h)^{r'}$, where $x' = (x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$.
- Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'')$.

Observe that, if $Z = e(g, g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{\varrho-1}$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_\varrho$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g, g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$. Any non-negligible advantage of $\mathcal{A}$ is converted to a non-negligible advantage of $\mathcal{B}$.

Lemma 15. *If the hash function $\overline{H}$ is a random oracle and the DBDH assumption holds, then* ***Game*** $\boldsymbol{G}_k$ *and* ***Game*** $\boldsymbol{G}_{k+1}$ *are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_\varrho$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g, g)^{z_1 z_2 z_3}$ or Z is

a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_k$ or **Game** $\mathbf{G}_{k+1}$ as follows.

$\mathcal{B}$ first chooses hash functions $\widetilde{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, $\hat{\mathsf{H}} : \{0,1\}^* \to \mathbb{G}^*$, $\check{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_\lambda^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. Then, $\mathcal{B}$ computes $A' = g^{\alpha'}$ and sets $Y' = g^{z_1}, Y_1' = g^{s_1}$, where $\alpha', s_1 \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, if $T_{\mathsf{S}}[i] = \perp$, $\mathcal{B}$ chooses $\widetilde{T}[i] \leftarrow \mathbb{G}$; else $\mathcal{B}$ sets $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$. $\mathcal{B}$ also maintains a list $L_{\overline{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\overline{\mathsf{H}}$, where the list is empty initially. Next, $\mathcal{B}$ choose $\alpha, \beta, \beta_1 \leftarrow \mathbb{Z}_p^*$ and computes $X = g^\beta, Y = (Y')^\beta, X_1 = g^{\beta_1}, Y_1 = (Y_1')^{\beta_1}$. For each $i \in [n']$, it computes $T[i] = e(A', \widetilde{T}[i])^\alpha$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, g_1, h, p, \widetilde{\mathsf{H}}, \hat{\mathsf{H}}, \check{\mathsf{H}}, Y', Y_1', A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y, X_1, Y_1), sk = (\alpha, \beta, \beta_1)$ to the adversary $\mathcal{A}$, where $g_1, h \leftarrow \mathbb{G}$.

$\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as follows:

- $\overline{\mathsf{H}}$ query on x: If there is some $(x, t_x, h_x, coin) \in L_{\overline{\mathsf{H}}}$, $\mathcal{B}$ returns h_x; otherwise, $\mathcal{B}$ picks $coin \in \{0,1\}$ at random such that $\mathsf{Pr}[coin = 0] = \rho$. ($\rho$ will be determined later.) Then, randomly chooses $t_x \leftarrow \mathbb{Z}_p^*$. The record $(x, t_x, h_x = (g^{z_2})^{coin} \cdot g^{t_x}, coin)$ is added to $L_{\overline{\mathsf{H}}}$ and h_x is sent to $\mathcal{A}$.
- TKGen query on x: $\mathcal{B}$ searches $L_{\overline{\mathsf{H}}}$ for a record $(x, t_x, h_x, coin)$. If $coin = 1$, it aborts and terminates; otherwise $\mathcal{B}$ returns $(g^{z_1})^{t_x}$.
- $\mathsf{TKGen}_{\mathrm{one}}$ query on $(\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c = (U, V), c' = (U', V'), c''),\ x)$: $\mathcal{B}$ sets $x' = (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c')$. If $e(g, c'') \neq e(U', g_1^u \cdot h)$, return $\perp$; else return $\mathsf{tk} = (\hat{\mathsf{H}}(x'))^{s_1}$.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^* such that $x^* \notin \mathsf{S}$. $\mathcal{B}$ searches $L_{\overline{\mathsf{H}}}$ for a record $(x^*, t_{x^*}, h_{x^*}, coin)$. If $coin = 0$, it aborts and terminates; otherwise $\mathcal{B}$ picks $b \in \{0,1\}$ randomly and proceeds as follows.

- For each $j \in [k]$, choose $Q_j, S_j \leftarrow \mathbb{G}_T$ randomly.
- Compute $c = (U = g^{z_3}, V = Z \cdot e(g^{z_3}, g^{z_1})^{\beta t_{x^*}} \cdot m_b)$. Note that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $V = e(\overline{\mathsf{H}}(x^*), Y)^{z_3} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then V also is a random element of $\mathbb{G}_T$.
- Choose $r' \leftarrow \mathbb{Z}_p^*$ randomly, and compute $U' = g^{r'}$, $V' = e(\hat{\mathsf{H}}(x'), Y_1)^{r'} \cdot m_b$, $c'' = (g_1^u \cdot h)^{r'}$, where $x' = (x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x^* \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c' = (U', V'))$.
- Return $\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c, c', c'')$.

The probability that $\mathcal{B}$ does not abort during the simulation is given by $\rho^{q_{tken}}(1-\rho)$ which is maximized at $\rho = 1 - 1/(q_{tken}+1)$, where q_{tken} denotes the number of TKGen queries by the adversary $\mathcal{A}$. Now, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_k$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{k+1}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$.

Lemma 16. *If the hash function $\hat{H}$ is a random oracle, the hash function $\check{H}$ is collision-resistant and the DBDH assumption holds, then* ***Game*** $\boldsymbol{G}_{k+1}$ *and* ***Game*** $\boldsymbol{G}_{k+2}$ *are computationally indistinguishable.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that distinguishes **Game** $\mathbf{G}_{\varrho-1}$ and **Game** $\mathbf{G}_{\varrho}$ with non-negligible advantage. Then we build a PPT algorithm $\mathcal{B}$ breaking the DBDH assumption with non-negligible advantage. $\mathcal{B}$ is given $e, \mathbb{G}, \mathbb{G}_T, p, g, g^{z_1}, g^{z_2}, g^{z_3}, Z$ and going to tell whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element in $\mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to simulate **Game** $\mathbf{G}_k$ or **Game** $\mathbf{G}_{k+1}$ as follows.

$\mathcal{B}$ first chooses hash functions $\widetilde{\mathsf{H}}, \overline{\mathsf{H}} : \mathcal{U} \to \mathbb{G}^*$, $\check{\mathsf{H}} : \{0,1\}^* \to \mathbb{Z}_p^*$, and runs $(\mathsf{pp}_{\mathsf{CH}}, T_{\mathsf{init}}, ST) \leftarrow \mathsf{CH}_{\lambda}^{(\mathsf{rob})}.\mathsf{Setup}(\lambda, n)$, $(T_{\mathsf{S}}, ST) \leftarrow \mathsf{CH}_{\lambda}^{(\mathsf{rob})}.\mathsf{Insert}(pp_{\mathsf{CH}}, T_{\mathsf{init}}, ST, \mathsf{S})$, where $\mathsf{pp}_{\mathsf{CH}}$ contains k hash functions $(\mathsf{H}_j : \mathcal{U} \to [n'])_{j\in[k]}$. Then, $\mathcal{B}$ computes $A' = g^{\alpha'}$ and sets $Y' = g^s, Y'_1 = g^{z_1}$, where $\alpha', s \leftarrow \mathbb{Z}_p^*$. For each $i \in [n']$, if $T_{\mathsf{S}}[i] = \perp$, $\mathcal{B}$ chooses $\widetilde{T}[i] \leftarrow \mathbb{G}$; else $\mathcal{B}$ sets $\widetilde{T}[i] := \widetilde{\mathsf{H}}(T_{\mathsf{S}}[i])$. $\mathcal{B}$ also maintains a list $L_{\hat{\mathsf{H}}}$ for answering $\mathcal{A}$'s random oracle queries on $\hat{\mathsf{H}}$, where the list is empty initially. Next, $\mathcal{B}$ choose $\alpha, \beta, \beta_1, \delta, \delta' \leftarrow \mathbb{Z}_p^*$ and computes $X = g^{\beta}, Y = (Y')^{\beta}, X_1 = g^{\beta_1}, Y_1 = (Y'_1)^{\beta_1}, g_1 = g^{\delta}, h = g^{\delta'}$. For each $i \in [n']$, it computes $T[i] = e(A', \widetilde{T}[i])^{\alpha}$. $\mathcal{B}$ sends $\mathsf{pp} = (e, \mathbb{G}, \mathbb{G}_T, g, g_1, h, p, \widetilde{\mathsf{H}}, \hat{\mathsf{H}}, \check{\mathsf{H}}, Y', Y'_1, A', \mathsf{pp}_{\mathsf{CH}})$, $pk = (T, X, Y, X_1, Y_1), sk = (\alpha, \beta, \beta_1)$ to the adversary $\mathcal{A}$.

$\mathcal{B}$ answers $\mathcal{A}$'s oracle queries as follows:

- $\hat{\mathsf{H}}$ query on x: If there is some $(x, t_x \in \mathbb{Z}_p^*, h_x \in \mathbb{G}) \in L_{\hat{\mathsf{H}}}$, $\mathcal{B}$ returns h_x; otherwise, $\mathcal{B}$ randomly chooses $t_x \leftarrow \mathbb{Z}_p^*$. The record $(x, t_x, h_x = g^{t_x})$ is added to $L_{\hat{\mathsf{H}}}$ and h_x is sent to $\mathcal{A}$.
- TKGen query on x: $\mathcal{B}$ returns $(\overline{\mathsf{H}}(x))^s$.
- $\mathsf{TKGen}_{\mathrm{one}}$ query on $(\mathsf{ct} = ((Q_j, S_j)_{j\in[k]}, c = (U, V), c' = (U', V'), c''),\ x)$: $\mathcal{B}$ sets $x' = (x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel U')$, $u = \check{\mathsf{H}}(x \parallel (Q_j, S_j)_{j\in[k]} \parallel c \parallel c')$. If $e(g, c'') \neq e(U', g_1^u \cdot h)$, return $\perp$; else find some $(x', t_{x'}, h_{x'} = g^{t_{x'}})$ in $L_{\hat{\mathsf{H}}}$, and return $\mathsf{tk} = (g^{z_1})^{t_{x'}} = (\hat{\mathsf{H}}(x'))^{z_1}$.
  Note that, if $e(g, c'') = e(U', g_1^u \cdot h)$, with overwhelming probability, x' is not equal to x'^* (computed in the challenge phase) because $\mathsf{ct} \neq \mathsf{ct}^*$ and $\check{\mathsf{H}}$ is a collision-resistant hash function, where ct^* is the challenge ciphertext. Hence, there exists some $(x', t_{x'}, h_{x'} = g^{t_{x'}})$ in $L_{\hat{\mathsf{H}}}$.

At some point, $\mathcal{A}$ submits two messages m_0, m_1 and an item x^* such that $x^* \notin \mathsf{S}$. $\mathcal{B}$ picks $b \in \{0,1\}$ randomly and proceeds as follows.

- For each $j \in [k]$, choose $Q_j^*, S_j^* \leftarrow \mathbb{G}_T$ randomly.
- Choose $U^* \leftarrow \mathbb{G}$, $V^* \leftarrow \mathbb{G}_T$ randomly, and set $c^* = (U^*, V^*)$.
- Set $U'^* = g^{z_3}$, $x'^* = (x^* \parallel (Q_j^*, S_j^*)_{j\in[k]} \parallel c^* \parallel U'^*)$, randomly choose $t_{x'^*} \leftarrow \mathbb{Z}_p^*$, and the record $(x'^*, t_{x'^*}, h_{x'^*} = (g^{z_2})^{t_{x'^*}})$ is added to $L_{\hat{\mathsf{H}}}$.
- Compute $V'^* = T^{\beta_1 t_{x'^*}} \cdot m_b$, $c'^* = (g^{z_3})^{u^*\delta + \delta'} = (g_1^{u^*} \cdot h)^{z_3}$, where $u^* = \check{\mathsf{H}}(x^* \parallel (Q_j^*, S_j^*)_{j\in[k]} \parallel c^* \parallel c'^* = (U'^*, V'^*))$. Note that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $V'^* = e(\hat{\mathsf{H}}(x'^*), Y_1)^{z_3} \cdot m_b$; otherwise Z is a random element of $\mathbb{G}_T$, then V'^* also is a random element of $\mathbb{G}_T$.
- Return $\mathsf{ct}^* = ((Q_j^*, S_j^*)_{j\in[k]}, c^*, c'^*, c''^*)$.

Observe that, if $Z = e(g,g)^{z_1 z_2 z_3}$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{k+1}$; If Z is a random element of $\mathbb{G}_T$, then $\mathcal{B}$ has properly simulated **Game** $\mathbf{G}_{k+2}$.

Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish whether $Z = e(g,g)^{z_1 z_2 z_3}$ or Z is a random element of $\mathbb{G}_T$.

I.2 Changes in MAMF

Changes in definitions. Firstly, we present the algorithm description for one-time token generation in MAMF as follows.

- $\mathsf{tk} \leftarrow \mathsf{TKGen}_{\mathsf{one}}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m, \sigma)$: The token generation algorithm for a pair of message and signature is run by the agency. It takes the public parameter pp, a sender's public key pk_{s}, a receiver's public key pk_{r}, the agency's secret key sk_{Ag}, the judge's public key pk_{J}, a message m and a signature σ as input, and outputs a token tk.

Thus, the correctness of MAMF additional requires that

– if $m \notin \mathsf{S}$, then $\mathsf{tk} \leftarrow \mathsf{TKGen}_{\mathsf{one}}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m, \sigma)$, and $\mathsf{Judge}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, pk_{\mathsf{Ag}}, sk_{\mathsf{J}}, m, \sigma, \mathsf{tk}) = 1$.

Changes in MAMF construction. After that, we can construct the algorithm for the one-time token generation in MAMF as shown in Fig. 26.

$\underline{\mathsf{TKGen}_{\mathsf{one}}(\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{r}}, sk_{\mathsf{Ag}}, pk_{\mathsf{J}}, m, \sigma):}$
$(\pi, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}}) \leftarrow \sigma$
$y \leftarrow (\mathsf{pp}, pk_{\mathsf{s}}, pk_{\mathsf{Ag}}, pk_{\mathsf{J}}, c, k_{\mathsf{r}}, k_{\mathsf{J}}, c_{\mathrm{t}}, m)$
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{Verify}(pk_{\mathsf{r}}, \pi, y) = 0$: Return $\perp$
$\mathsf{pp}_{\mathsf{USPCE}} \leftarrow pk_{\mathsf{Ag}}$, $msk_{\mathsf{USPCE}} \leftarrow sk_{\mathsf{Ag}}$
Return $\mathsf{tk} \leftarrow \mathsf{USPCE}.\mathsf{TKGen}_{\mathrm{one}}(\mathsf{pp}_{\mathsf{USPCE}}, msk_{\mathsf{USPCE}}, c_{\mathrm{t}}, m)$

Fig. 26 Algorithm descriptions of $\mathsf{TKGen}_{\mathsf{one}}$ in MAMF

The correctness of MAMF is guaranteed by the correctness of the underlying USPCE. For other security properties of MAMF, we omit the rigorous security proof here. On the one hand, most of the proof for Theorem 3 can be applied here, except for the proof for the untraceability against judge, of which the game provides a one-time token generation oracle. On the other hand, similar to the tokens for illegal messages in the pre-defined set S, one-time token for messages not in S would not leak meaningful information either, which is guaranteed by the confidentiality against user. So the queries on one-time token oracle cannot help the adversary to gain additional advantages. Therefore, the MAMF with one-time token generation is also secure.