Asymmetric Group Message Franking: Definitions & Constructions

Junzuo Lai[1], Gongxian Zeng[2(✉)], Zhengan Huang[2(✉)], Siu Ming Yiu[3], Xin Mu[2], and Jian Weng[1(✉)]

[1] College of Information Science and Technology, Jinan University, Guangzhou, China
laijunzuo@gmail.com, cryptjweng@gmail.com
[2] Peng Cheng Laboratory, Shenzhen, China
gxzeng@cs.hku.hk, zhahuang.sjtu@gmail.com, mux@pcl.ac.cn
[3] The University of Hong Kong, Hong Kong, China
smyiu@cs.hku.hk

Abstract. As online group communication scenarios become more and more common these years, malicious or unpleasant messages are much easier to spread on the internet. Message franking is a crucial cryptographic mechanism designed for content moderation in online end-to-end messaging systems, allowing the receiver of a malicious message to report the message to the moderator. Unfortunately, the existing message franking schemes only consider 1-1 communication scenarios.

In this paper, we systematically explore message franking in group communication scenarios. We introduce the notion of asymmetric group message franking (AGMF), and formalize its security requirements. Then, we provide a framework of constructing AGMF from a new primitive, called HPS-KEM$^{\Sigma}$. We also give a construction of HPS-KEM$^{\Sigma}$ based on the DDH assumption. Plugging the concrete HPS-KEM$^{\Sigma}$ scheme into our AGMF framework, we obtain a DDH-based AGMF scheme, which supports message franking in group communication scenarios.

Keywords: Message franking; Hash proof system; Key encapsulation mechanism; Signature of knowledge

1 Introduction

In recent years, secure messaging applications have become extremely popular for conversations between individuals and groups. Billions of people communicate with each other via messaging applications like Facebook Messager, Twitter, Signal, Google Allo, etc. every day. However, these messaging applications are abused for the spread of malicious information such as harassment messages, phishing links, fake information and so on. Facebook Messager [20,21] introduced the concept of *message franking*, which was formally studied in [24] later. Generally, (symmetric or asymmetric) message franking [21,24,36] provides *accountability*, i.e., it allows the receiver to report the malicious messages to some moderator (e.g., the platform or some trusted third party), and meanwhile guarantees that no fake reports can be fabricated to frame an honest sender. Deniability, as also an explicit goal of Facebooks message franking based moderation system [21], is formalized for asymmetric message franking (AMF) in [36]. Informally, deniability ensures that when the receiver reports some malicious messages, only the moderator is able to validate the report. In other words, after a compromise, the sender can deny sending the messages technically, in order to avoid backlash or embarrassment (for more explanations, please refer to [36]). Now, message franking is a vital security feature for secure messaging applications, especially in government affairs, business and so on.

Compared with symmetric message franking, asymmetric message franking supports third-party moderation, decoupling the platform and the moderator, which enables cross-platform moderation of multiple messaging systems. As pointed out in [36], this property is necessary in decentralized or federated messaging systems like Matrix [2] or Mastodon [1], and is advantageous if the platform cannot adequately moderate messages, or if sub-communities want to enforce their own content policies.

However, the existing AMF [36] only considers the case of *1-1 communication*. As for another common scenarios, *group communications*, no works have ever related to this topic. Group communication plays an important role in teamwork or other multi-user scenarios, and many popular instance communication tools support it, such as WhatsApp and Signal. In addition, the IETF launched the message-layer security (MLS) working group, which aims to standardize an eponymous secure group messaging protocol. At the meanwhile, the academic researchers also paid lots of attention, such as [39,23,13,3,4].

Contributions. In this paper, we systematically explore message franking in group communication scenarios. The contributions are listed as follows.

- We introduce a new primitive called asymmetric group message franking (AGMF), and formalize its security notions.
- We present a variant of key encapsulation mechanism (KEM), called $\mathrm{HPS\text{-}KEM}^{\Sigma}$, and provide a construction based on the decisional Diffie-Hellman (DDH) assumption. The construction can be extended to be built based on the k-Linear assumption.
- We provide a framework of constructing AGMF from $\mathrm{HPS\text{-}KEM}^{\Sigma}$, and show that it achieves the required security properties. Actually, we also obtain a framework of constructing AMF from $\mathrm{HPS\text{-}KEM}^{\Sigma}$ (i.e., when the size of the receiver set is 1).

When plugging the concrete $\mathrm{HPS\text{-}KEM}^{\Sigma}$ scheme into our AGMF framework, we obtain an AGMF scheme based on the DDH assumption, which implies a DDH-based AMF scheme. Note that the only existing AMF scheme [36][†]

[†] Very recently, Issa et al. also consider a kind of AMF, called Hecate [28], but it is somewhat different from [36]. Firstly, [36] and this paper only focus on the intrinsic/fundamental security properties of A(G)MF, while Hecate [28] also considers

is constructed based on a somewhat exotic assumption, knowledge-of-exponent assumption (KEA) [5], or the Gap Diffie-Hellman (GDH) assumption [6].

AGMF primitive. In the context of AGMF, there are three kinds of parties involved: the sender, the multiple receivers, and the moderator (or called the judge). Syntactically, similar to AMF, an AGMF consists of nine algorithms: three algorithms for generating public parameters and key pairs, three algorithms (Frank, Verify, Judge) for creating and verifying genuine signatures, and the other three algorithms (Forge, RForge, JForge) for forging signatures. In a nutshell, the sender invokes the signing algorithm Frank to generate signatures for a receiver set. Any receiver calls Verify (with his/her secret key as input) to verify the received signatures. If some receiver reports some malicious message to the moderator, the moderator verifies the report with algorithm Judge. Algorithms Forge, RForge and JForge are not intended to be run by legitimate users. Their existence guarantee deniability in particular compromise scenarios.

We consider three kinds of security requirements for AGMF: accountability, deniability, and receiver anonymity.

- *Accountability.* Accountability is formalized with two special properties: sender binding and receiver binding. Sender binding guarantees that any sender should not be able to trick receivers into accepting unreportable messages, and receiver binding guarantees that any receivers cannot deceive the judge or other honest receivers (to frame the innocent sender).
- *Deniability.* Deniability is formalized with three special properties: universal deniability, receiver compromise deniability and judge compromise deniability. Universal deniability is formalized to guarantee deniability when no receiver secret key or judge secret key is compromised. Receiver compromise deniability is formalized to guarantee deniability when the secret keys of some receivers in the receiver set are compromised. Judge compromise deniability is formalized to guarantee deniability when the judge's secret key is compromised.
- *Receiver anonymity.* Receiver anonymity is formalized to guarantee that any one (except for the receivers in the receiver set), including the judge, cannot tell which receiver set a signature is generated for.

When formalizing the above security requirements, the existence of multiple receivers in group communication scenarios introduces new security risks, making the security models in AGMF different from that in AMF.

First of all, due to the existence of multiple receivers, it is natural to consider that the adversary in the security models of AGMF is able to corrupt some of the receivers. These corruptions bring in the following concerns.

others, e.g., forward/backward secrecy. Secondly, [36] only needs one round of communication and can generate the AMF signature on the fly, but Hecate [28] introduces an AMF with preprocessing model, resulting in one more preprocessing round with the moderator to get a "token" before generating the AMF signature. Hence, we follow the definition in [36], not considering Hecate [28] when talking about AMF.

– Compared with receiver binding for AMF, which requires that any *single* receiver cannot deceive the judge to frame the innocent sender, receiver binding for AGMF requires that *any corrupted receivers* cannot deceive the judge or *the other honest receivers* to frame the innocent sender.
– Recall that receiver compromise deniability for AMF requires that a party with the receiver's secret key is able to create a signature, such that for other parties with access to this secret key, it is indistinguishable from honestly-generated signatures. Comparatively, receiver compromise deniability for AGMF requires that *any corrupted users in the receiver set* are able to create a signature, such that for other parties with access to *these corrupted users' secret keys*, it is indistinguishable from honestly-generated signatures.

Secondly, we also formalize a new security notion called receiver anonymity, which is not considered in AMF. Receiver anonymity requires that any one (except for the receivers in the receiver set), including the judge, cannot tell which receiver set a signature is generated for. With receiver anonymity, the receivers in group communication scenarios can report the malicious messages to the moderator with less concerns. If the AGMF scheme does not support receiver anonymity, then the judge can know some information about the identities of receivers from the report. Then the reporter may be at the risk of vengeance, especially if the judge is possible to leak the receivers identity information to the sender. As a result, it would silence the reporters. Actually, anonymity has already been considered in many group scenarios, such as accountable anonymous group messaging system [14,38,35], and proactively accountable anonymous messaging [15].

More importantly, in all our proposed security models, the adversary is allowed to corrupt the receivers *adaptively*. In other words, how many and whose secret keys are compromised is unpredictable in practical scenarios, which is greatly different from that in AMF (i.e., only one receiver's secret key is compromised). Compared with non-adaptive corruptions (i.e., the adversary is required to announce all the corrupted users at the beginning before seeing all users' public keys), adaptive corruptions are more natural, and cryptographic schemes supporting adaptive corruptions are much more difficult to obtain, as mentioned in [25,26,29,3].

AGMF construction. Following, we highlight the technical details of our AGMF construction.

HPS-KEM$^\Sigma$. In order to provide a framework of constructing AGMF, we introduce a new primitive. This primitive is a variant of key encapsulation mechanism (KEM) satisfying that (i) it can be interpreted from the perspective of hash proof system (HPS) [17], and (ii) for some special relations (about the public/secret keys, the encapsulated keys and ciphertexts), there exist corresponding Sigma protocols [16]. We call this primitive *HPS-based KEM supporting Sigma protocols (HPS-KEM$^\Sigma$)*.

A HPS-KEM$^\Sigma$ scheme $\mathsf{HPS\text{-}KEM}^\Sigma$ mainly contains six algorithms: Setup, KG, $\mathsf{encap_c}$, $\mathsf{encap_k}$, decap and $\mathsf{encap^*_c}$. In a nutshell, Setup outputs a public parameter, and KG outputs a pair of public/secret user keys. Taking the public parameter as input, *without user's public key*, $\mathsf{encap_c}$ outputs a well-formed ciphertext, and $\mathsf{encap}_\mathsf{c}^*$ outputs a ciphertext which could be well-formed or ill-formed. The algorithm $\mathsf{encap_k}$, sharing the same randomness space with $\mathsf{encap_c}$, takes the public parameter and user's public key as input, and outputs an encapsulated key. With user's secret key, the algorithm decap is invoked to decapsulate the ciphertexts to get the encapsulated keys. The correctness demands that given a ciphertext output by $\mathsf{encap_c}$ with randomness r, decap will recover an encapsulated key, which is equivalent to that generated by $\mathsf{encap_k}$ with the same randomness r.

It is required that there are Sigma protocols to prove that some results are exactly output by KG, $\mathsf{encap_c}$, $\mathsf{encap_k}$ or $\mathsf{encap}_\mathsf{c}^*$. We also require the following properties informally.

1. Universality: when 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 a witness w (indicating that c is generated via $\mathsf{encap}_\mathsf{c}^*$), such that with the ciphertext c as input, decap outputs a key equal to k.
2. Unexplainability: it is difficult to generate a ciphertext c and a witness w (indicating that c is generated via $\mathsf{encap}_\mathsf{c}^*$), such that c is well-formed.
3. Indistinguishability: the ciphertext output by $\mathsf{encap}_\mathsf{c}^*$ is indistinguishable from the well-formed ciphertext output by $\mathsf{encap_c}$.
4. SK-second-preimage resistance: when given a pair of public and secret keys, it is difficult to generate another valid secret key for this public key.
5. Smoothness: when given a public key, the algorithm decap, fed with a ciphertext generated via $\mathsf{encap}_\mathsf{c}^*$ and a user's 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.

AGMF from HPS-KEM$^\Sigma$. Taking HPS-KEM$^\Sigma$ as a building block, we construct AGMF as follows.

The public/secret key pairs of all users (including the judge) are generated by invoking the key generation algorithm KG of the HPS-KEM$^\Sigma$.

Given a pair of sender's public/secret keys $(pk_\mathrm{s}, sk_\mathrm{s})$, a receiver set $S = \{pk_{\mathrm{r}_i}\}_{i\in[|S|]}$, the judge's public key pk_J and a message m, the sender calls Frank to generate the signature as follows:

(1) Compute $c \leftarrow \mathsf{encap_c}(pp; r)$, $k_\mathrm{J} \leftarrow \mathsf{encap_k}(pp, pk_\mathrm{J}; r)$ and $(k_{\mathrm{r}_i} \leftarrow \mathsf{encap_k}(pp, pk_{\mathrm{r}_i}; r))_{pk_{\mathrm{r}_i}\in S}$ with the same randomness r.

(2) Consider the following relation

$$\begin{aligned}\mathcal{R} = \{ \ (\ (sk_\mathrm{s}, r, r^*), (pp, pk_\mathrm{s}, pk_\mathrm{J}, c, k_\mathrm{J}) \ ) :& \\ (\ (sk_\mathrm{s}, pk_\mathrm{s}) \in \mathcal{R}_\mathrm{s} \wedge (r, (c, k_\mathrm{J}, pk_\mathrm{J})) \in \mathcal{R}_\mathrm{c,k} \ )& \\ \vee \ (\ (r^*, c) \in \mathcal{R}_\mathrm{c}^* \ ) \ \}&\end{aligned} \tag{1}$$

where $\mathcal{R}_\mathrm{s}$ is a relation proving that the sender's public/secret keys are valid, $\mathcal{R}_\mathrm{c,k}$ is a relation proving that (c, k_J) are generated via $\mathsf{encap_c}$ and $\mathsf{encap_k}$

with the same randomness r, and $\mathcal{R}_c^*$ is a relation proving that c is a ciphertext output by encap_c^* with randomness r^*. As the HPS-KEM$^\Sigma$ guarantees that there are Sigma protocols for KG, encap_c, encap_k and encap_c^*, we can obtain a signature of knowledge (SoK) scheme for $\mathcal{R}$ by applying the Fiat-Shamir transform [22] and composition operations of Sigma protocols [7].

(3) Employ the SoK scheme for $\mathcal{R}$ to generate a signature proof π of statement (pp, pk_s, pk_J, c, k_J) for a message $\overline{m} = (m||\{k_{r_i}\}_{pk_{r_i}\in S})$ with a witness (sk_s, r).

(4) Return the signature $\sigma = (\pi, c, k_J, \{k_{r_i}\}_{pk_{r_i}\in S})$.

The verification algorithm Verify and the moderation algorithm Judge work similarly. When some receiver receives a message and a signature or the judge receives a report with a message and a signature, the first step of these algorithms is to verify if the proof π in the signature is valid. If not valid, Verify (resp., Judge) returns 0; otherwise, Verify (resp., Judge) returns 1 if and only if $\mathsf{decap}(pp, sk_r, c) \in \{k_{r_i}\}$ (resp., $\mathsf{decap}(pp, sk_J, c) = k_J$).

Now we turn to describe the forging algorithms Forge, RForge and JForge.

Given a sender's public keys pk_s, a receiver set $S = \{pk_{r_i}\}_{i\in[|S|]}$, the judge's public key pk_J and a message m, the universal forging algorithm Forge proceeds as follows:

(1) Compute $c \leftarrow \mathsf{encap}_c^*(pp; r^*)$ with randomness r^*, $k_J \leftarrow \mathcal{K}$, and $(k_{r_i} \leftarrow \mathcal{K})_{pk_{r_i}\in S}$.

(2) Employ the SoK scheme for $\mathcal{R}$ in Eq. (1) to generate a signature proof π of statement (pp, pk_s, pk_J, c, k_J) for a message $\overline{m} = (m||\{k_{r_i}\}_{pk_{r_i}\in S})$ with a witness r^*.

(3) Return the signature $\sigma = (\pi, c, k_J, \{k_{r_i}\}_{pk_{r_i}\in S})$.

Given a sender's public key pk_s, a receiver set $S = \{pk_{r_i}\}_{i\in[|S|]}$, the corrupted receivers' secret keys $\{sk_{r_i}\}_{pk_{r_i}\in S_{cor}}$ (where $S_{cor} \subset S$), the judge's public key pk_J and a message m, the receiver compromise forging algorithm RForge proceeds similarly to Forge, except that $(k_{r_i})_{pk_{r_i}\in S}$ are generated as follows: for each $pk_{r_i} \in S\backslash S_{cor}$, samples $k_{r_i} \leftarrow \mathcal{K}$; for each $pk_{r_i} \in S_{cor}$, computes $k_{r_i} \leftarrow \mathsf{decap}(pp, sk_{r_i}, c)$.

Given a sender's public keys pk_s, a receiver set $S = \{pk_{r_i}\}_{i\in[|S|]}$, the judge's secret key sk_J and a message m, the judge compromise forging algorithm JForge proceeds similarly to Forge, except that k_J is generated by $k_J \leftarrow \mathsf{decap}(pp, sk_J, c)$.

Security analysis. Now we briefly show that our AGMF framework provides accountability, deniability and receiver anonymity.

Informally, sender binding requires that any malicious sender cannot generate a signature such that an honest receiver accepts it but the judge rejects it. If there exists an adversary generating such a signature $\sigma = (\pi, c, k_J, \{k_{r_i}\}_{pk_{r_i}\in S})$, then we have: (i) π is a valid proof for the relation $\mathcal{R}$; (ii) $k' = \mathsf{decap}(pp, sk_r, c) \in \{k_{r_i}\}_{pk_{r_i}\in S}$; (iii) $\mathsf{decap}(pp, sk_J, c) \neq k_J$. Observe that to generate the valid proof π for $\mathcal{R}$, the adversary needs to know witness (sk_s, r) or r^*. According to (i) and (iii), it implies that the adversary generates π using the witness r^*, which suggests that c is generated via encap_c^*. The unexplainability of HPS-KEM$^\Sigma$ implies that c is not well-formed with overwhelming probability. So according to (ii), (c, k', r^*) leads to a successful attack on universality of HPS-KEM$^\Sigma$.

Receiver binding requires that any malicious receivers cannot generate a signature such that an honest receiver or the judge accepts it.

- Supposing that there exists an adversary generating a signature $\sigma = (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$ such that an honest receiver accepts it, we have: (i) π is a valid proof for the relation $\mathcal{R}$; (ii) $k' = \mathsf{decap}(pp, sk_{\mathrm{r}}, c) \in \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S}$. Observe that to generate the valid proof π for $\mathcal{R}$, the adversary needs to know witness (sk_{s}, r) or r^*.
  - If the adversary knows (sk_{s}, r), it implies that sk_{s} is a valid secret key of the sender. Since the adversary is not allowed to corrupt the sender, it is contradictory to SK-second-preimage resistance of HPS-KEM$^{\Sigma}$.
  - If the adversary knows r^*, it implies that c is generated via $\mathsf{encap}^*_{\mathrm{c}}$. The unexplainability of HPS-KEM$^{\Sigma}$ guarantees that c is not well-formed with overwhelming probability. So according to (ii), (c, k', r^*) leads to a successful attack on universality of HPS-KEM$^{\Sigma}$.
- Supposing that there exists an adversary generating a signature $\sigma = (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$ such that the judge accepts it, we have: (i) π is a valid proof for the relation $\mathcal{R}$; (ii) $\mathsf{decap}(pp, sk_{\mathrm{J}}, c) = k_{\mathrm{J}}$. With similar analysis, this leads to a successful attack on SK-second-preimage resistance or universality of HPS-KEM$^{\Sigma}$.

Next, we turn to analyze universal deniability, receiver compromise deniability, and judge compromise deniability of our AGMF framework. Due to the similarity of security analysis of these deniabilities, here we just show how universal deniability is achieved.

Universal deniability requires that the outputs of Frank and Forge are indistinguishable. For the generation of signature $\sigma = (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$, the differences between the two algorithms are as follows.

- $(c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$: Algorithm Frank computes $c \leftarrow \mathsf{encap}_{\mathrm{c}}(pp; r)$, $k_{\mathrm{J}} \leftarrow \mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{J}}; r)$ and $(k_{\mathrm{r}_i} \leftarrow \mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{r}_i}; r))_{pk_{\mathrm{r}_i} \in S}$ with the same randomness r, while algorithm Forge computes $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$ with randomness r^*, and samples $k_{\mathrm{J}} \leftarrow \mathcal{K}$ and $(k_{\mathrm{r}_i} \leftarrow \mathcal{K})_{pk_{\mathrm{r}_i} \in S}$.

  The indistinguishability of HPS-KEM$^{\Sigma}$ guarantees that c output by $\mathsf{encap}_{\mathrm{c}}$ is indistinguishable from that output by $\mathsf{encap}^*_{\mathrm{c}}$. When $c \leftarrow \mathsf{encap}_{\mathrm{c}}(pp; r)$, we have $\mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{r}_i}; r) = \mathsf{decap}(pp, sk_{\mathrm{r}_i}, c)$ and $\mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{J}}; r) = \mathsf{decap}(pp, sk_{\mathrm{J}}, c)$. On the other hand, when $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$, the smoothness of HPS-KEM$^{\Sigma}$ guarantees that the encapsulated keys $k_{\mathrm{r}_i} \leftarrow \mathsf{decap}(pp, sk_{\mathrm{r}_i}, c)$ and $k_{\mathrm{J}} \leftarrow \mathsf{decap}(pp, sk_{\mathrm{J}}, c)$ are indistinguishable from those random keys $k_{\mathrm{r}_i} \leftarrow \mathcal{K}$ and $k_{\mathrm{J}} \leftarrow \mathcal{K}$. Therefore, through hybrid arguments, we can show that $(c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$ output by Frank and Forge are indistinguishable.
- π: Frank generates a signature proof π for $\mathcal{R}$ with a witness (sk_{s}, r), while Forge generates π for $\mathcal{R}$ with a witness r^*. Because of zero knowledge property of the SoK scheme for $\mathcal{R}$, anyone cannot distinguish the proof output by Frank from that output by Forge.

Finally, we briefly explain why our AGMF framework achieves receiver anonymity. Informally, receiver anonymity requires that given two receiver sets S_0 and S_1

with the same size, a signature generated by Frank for S_0 is indistinguishable from that for S_1. According to the above security analysis of universal deniability, the signature output by Frank is indistinguishable from that output by Forge. Notice that, the signature generated by Forge does not contain any information about the receiver set. Thus, the signatures generated by Frank for S_0 and for S_1 are indistinguishable.

Construction of HPS-KEM$^{\Sigma}$. Inspired by the DDH-based HPS [17], we provide a construction of HPS-KEM$^{\Sigma}$, which can be extended to be built based on the k-Linear assumption. The main algorithms are constructed as follows.

KG outputs a pair of public/secret keys $(pk, sk) = (g_1^{x_1} g_2^{x_2}, (x_1, x_2))$, where g_1, g_2 are two generators of group $\mathbb{G}$ of prime order p, and x_1, x_2 are uniformly sampled from $\mathbb{Z}_p^*$.

To generate a well-formed ciphertext c, $\mathsf{encap_c}$ outputs $c = (u_1, u_2) = (g_1^r, g_2^r)$, where r is uniformly random sampled from $\mathbb{Z}_p^*$.

For generating a ciphertext, $\mathsf{encap_c^*}$ chooses randomness $r^* = (r, r') \in \mathbb{Z}_p^{*2}$ and outputs $c = (u_1, u_2) = (g_1^r, g_1^{r'})$.

Algorithm $\mathsf{encap_k}$ outputs an encapsulated key $k = pk^r$, where r is uniformly random sampled from $\mathbb{Z}_p^*$.

When inputting a ciphertext $c = (u_1, u_2)$ and a secret key $sk = (x_1, x_2)$, the algorithm decap outputs a key $k' = u_1^{x_1} u_2^{x_2}$.

Note that there are Sigma protocols for KG, $\mathsf{encap_c}$, $\mathsf{encap_k}$ or $\mathsf{encap_c^*}$: Okamoto's Sigma protocol [31] for KG, the Chaum-Pedersen protocol [11] for $\mathsf{encap_c}$ and $\mathsf{encap_k}$ with the same randomness, and Schnorr's Sigma protocol [33] for $\mathsf{encap_c^*}$.

Now we show our HPS-KEM$^{\Sigma}$ construction achieves the required properties.

With similar analysis in [17], we can easily obtain universality, indistinguishability and smoothness of our construction.

For unexplainability, suppose that there exists an adversary breaking the unexplainability of our scheme. In other words, the adversary generates $c = (u_1, u_2)$ and $w = (r, r')$, such that (i) $(u_1 = g_1^r) \wedge (u_2 = g_1^{r'})$, and (ii) c is well-formed. Note that c is well-formed implies that $u_2 = g_2^r$. So we can compute $\log_{g_1} g_2 = \frac{r'}{r}$, solving the DL problem.

For SK-second-preimage resistance (SK-2PR), suppose that there exists an adversary breaking the SK-2PR of our scheme. In other words, given a public/secret key pair $(pk, sk) = (g_1^{x_1} g_2^{x_2}, (x_1, x_2))$, the adversary outputs another secret key $sk' = (x_1', x_2')$ such that $pk = g_1^{x_1'} g_2^{x_2'}$. We can compute $\log_{g_1} g_2 = (x_1 - x_1')/(x_2' - x_2)$, solving the DL problem.

Discussion I: Lower bound. Following we present a lower bound of the size of AGMF signature.

Theorem 1. *Any AGMF with receiver binding and receiver compromise deniability must have signature size $\Omega(n)$, where n is the number of the members in S.*

Proof. Suppose that there exists a distinguisher $\mathcal{D}$ who knows all receivers' secret keys. Given a signature generated by RForge with a corrupted receiver set S_{cor},

$\mathcal{D}$ can distinguish whether someone is in S_{cor} or not, by verifying validity of the signature. Note that receiver binding and receiver deniability guarantee that only the receivers in S_{cor} would accept the signature. Thus, $\mathcal{D}$ can determine the set S_{cor} when given a signature generated by RForge. Therefore, the signature must contain enough bits to indicate S_{cor}. Since $S_{\text{cor}} \subseteq S$ and it can be an arbitrary subset, there are $2^{|S|} = 2^n$ kinds of different subsets. Thus, the bit length of signature is at least $\log_2 2^{|S|} = \log_2 2^n = n$. Considering that the signature output by Frank is indistinguishable from that output by RForge, the bit length of signature output by Frank is also $\Omega(n)$. $\square$

When plugging the concrete HPS-KEM$^{\Sigma}$ scheme into our AGMF framework, we obtain an AGMF scheme based on the DDH assumption. The bit length of the signature would be $9 \times |\mathbb{Z}_p^*| + (n+3) \times |\mathbb{G}|$, where n is the number of receivers and p is the order of group $\mathbb{G}$.

Theorem 1 indicates that the size of signature of AGMF is linear in n, and the coefficient of n in the size of signature of our AGMF scheme is $|\mathbb{G}|$, which is almost optimal. Note that a proof with similar idea is given by Damgård et al. in [18, Theorem 1], to show the lower bound of the size of multi-designated verifier signatures with any-subset simulation and strong unforgeablity.

Discussion II: AGMF when $n = 1$. Note that our method actually provides a framework of constructing AMF from HPS-KEM$^{\Sigma}$ (i.e., when the size of the receiver set is 1). The AMF scheme [36] is firstly constructed based on a somewhat exotic assumption, the KEA assumption [5]. As mentioned by Tyagi et al. [36], the KEA assumption poses a challenge for interpreting the concrete security analyses since the KEA extractor is not concretely instantiated. Then, they also show a variant scheme that can be proven secure using the GDH assumption [6], at the cost of signatures with *slightly* larger size. Specifically, the bit length of the AMF signature [36] based on the GDH assumption would be $9 \times |\mathbb{Z}_p^*| + 4 \times |\mathbb{G}|$.

When plugging the concrete HPS-KEM$^{\Sigma}$ scheme into the AMF framework, we obtain a DDH-based AMF scheme. Although the size of the signature of our AMF scheme would be $9 \times |\mathbb{Z}_p^*| + 4 \times |\mathbb{G}|$ as well, we stress that at the same security level, the binary representation of the group element in our scheme has smaller size than that in the GDH-based AMF scheme [36].

Discussion III: AGMF from AMF [36] directly. A trivial construction of AGMF is extended directly from the existing AMF [36], e.g., integrating AMF [36] with the "trivial" Signal group key mechanism (i.e., a set of individual links to each member of the group).

The extension has two shortcomings: i) the signature contains n NIZK proofs, and ii) it needs a non-standard assumption, which is inherited from AMF [36]. Our scheme does not have these shortcomings. We also consider another extension in the full version of this paper. The key point is that we extend the relation used in AMF [36] for one receiver to a relation for multiple receivers. However, this extension also has similar shortcomings mentioned above. Due to the space limitations, more details of the extension will be given in the full version of this paper.

Discussion IV: Integrating AGMF with group messaging protocols. For end-to-end encryption systems, there are kinds of requirements, including message franking, privacy, forward/backward security, etc.. Our paper focuses on asymmetric message franking in group communication scenarios, not caring about the other intrinsic security of group messaging protocols (e.g., privacy and authenticity in the form of post-compromise forward secrecy). Discussing a unified security model capturing other security properties is out of the scope of this paper and we remain it as a future work.

A potential method to integrate AGMF with group messaging protocols (e.g., [23,14,13,3]) is similar to AMF. In other words, we treat the output of AGMF as a signature and then encrypt the message and the signature following these protocols.

Related work. The technique of symmetric message franking (SMF) was firstly introduced by Facebook [20,21]. Grubbs et al. [24] initiated a formal study of SMF, formalizing a cryptographic primitive called compactly committing authenticated encryption with associated data (AEAD), and then showing that many in-use AEAD schemes can be used for SMF. Dodis et al. [19] pointed out that the Facebook SMF scheme is actually insecure, and proposed an efficient single-pass construction of compactly committing AEAD. Observing that in all previous SMF schemes, to make a report the receiver has to reveal the whole communication for a session, Leontiadis et al. [30] and Chen et al. [12] independently presented SMF constructions to tackle this problem. In CRYPTO 2019, Tyagi et al. [36] initiated a formal study of AMF, formalizing security notions of accountability and deniability for AMF, and showing an AMF construction via signature of knowledge [8].

Recently, some works [37,32,28] explore source-tracking, which allows the moderator to pinpoint the original source of a viral message rather than the immediate sender of the message (in the setting of message franking [24,36]). These works mainly focus on end-to-end encrypted messaging. It is an interesting direction to consider source-tracking in group settings.

Group messaging and its variants have been studied in many works [39,23,14,13,3,4], focusing on different properties or security requirements. To the best of our knowledge, currently there are no variants of group messaging which can provide the aforementioned accountability, deniability and anonymity simultaneously.

In 2020, Damgård et al. [18] proposed the notion of *off-the-record for any subset* in the constructions of multi-designated verifier signature (MDVS) for the group Off-the-Record messaging. The notion is somewhat similar to the receiver compromise deniability defined in this paper. As designated verifier signature does not have all desired properties in the setting of AMF [36], the MDVS construction [18] does not provide all required properties (e.g., accountability) in our AGMF scenarios either.

Roadmap. We recall some preliminaries in Section 2. Then in Section 3, we present the primitive of AGMF and formalize its security notions of accountability, deniability and receiver anonymity. Next, in Section 4, we introduce a primitive called HPS-KEM$^{\Sigma}$ and present a concrete construction. Taking HPS-KEM$^{\Sigma}$

as a building block, we provide a framework of constructing AGMF, and show that it achieves accountability, deniability and receiver anonymity in Section 5.

2 Preliminaries

Notations. 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, we write $\mathsf{Alg}(x; r)$ for the process of Alg on input x with inner randomness r, and use $y \leftarrow \mathsf{Alg}(x)$ to denote the process of running Alg on input x with uniformly sampled inner randomness r, and assigning y the result.

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 [22] as follows. For convenience, the recalled NIZK is a variant integrating the notion of signature of knowledge in [9,10,36] and the notion of NIZK in [7].

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

- $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{prove}(m, x, y)$: The prove algorithm takes $(m, x, y) \in \mathcal{M} \times \mathcal{X} \times \mathcal{Y}$ 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 $(x, y) \in \mathcal{R}$, we always have $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{verify}(m, \mathsf{NIZK}^{\mathcal{R}}.\mathsf{prove}(m, x, y), y) = 1$.
- **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. 1. Suppose that $\mathsf{NIZK}^{\mathcal{R}}$ makes use of a hash function Hash, and the hash function Hash with output length len in Fig. 1 is modeled as a random oracle (a local array H is employed).

$\mathbf{G}^{\text{real}}_{\mathsf{NIZK},\mathcal{A}}(\lambda)$:	$\mathbf{G}^{\text{ideal}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda)$:
$b \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda)$	$b \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda)$
Return b	Return b
$\mathcal{O}^{\mathsf{prove}}(m, x, y)$:	$\mathcal{O}^{\mathsf{prove}}(m, x, y)$:
If $(x, y) \notin \mathcal{R}$: Return $\perp$	If $(x, y) \notin \mathcal{R}$: Return $\perp$
$\pi \leftarrow \mathsf{prove}(m, x, y)$	$(st, \pi) \leftarrow \mathcal{S}_{\mathsf{prove}}(st, m, y)$
Return π	Return π
$\mathcal{O}^{\text{ro}}(str)$:	$\mathcal{O}^{\text{ro}}(str)$:
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. 1 Games for defining zero knowledge of $\mathsf{NIZK}^{\mathcal{R}}$

Sigma protocol. A Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 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 $(x, y) \in \mathcal{R}$, the input of the prover (resp., verifier) is (x, y) (resp., y). The prover first computes $(cm, aux) \leftarrow P_1(x, y)$ and sends the commitment cm to the verifier. The verifier (i.e., V_1) returns a challenge $cl \leftarrow \mathcal{CL}$. Then the prover replies with $z \leftarrow P_2(cm, cl, x, y, aux)$. Receiving z, the verifier (i.e., V_2) outputs $b \in \{0, 1\}$. The tuple (cm, cl, z) is called a *conversation*. We require that V does not make any random choices other than the selection of cl. For any fixed (cm, cl, z), if the final output of $V(y)$ is 1, (cm, cl, z) is called an *accepting conversation* for y. Correctness requires for all $(x, y) \in \mathcal{R}$, when $P(x, y)$ and $V(y)$ interact with each other, the final output of $V(y)$ is always 1.

The corresponding security notions are as follows.

Definition 1. (Knowledge soundness). *We say that a Sigma protocol (P, V) for $R \subseteq \mathcal{X} \times \mathcal{Y}$ provides* knowledge soundness*, if there is an efficient deterministic algorithm* Ext *such that on input $y \in \mathcal{Y}$ and two accepting conversations $(cm, cl, z), (cm, cl', z')$ where $cl \neq cl'$,* Ext *always outputs an $x \in \mathcal{X}$ satisfying $(x, y) \in \mathcal{R}$.*

Definition 2. (Special HVZK). *We say that a Sigma protocol (P, V) for $R \subseteq \mathcal{X} \times \mathcal{Y}$ with challenge space $\mathcal{CL}$ is* special honest verifier zero knowledge (special HVZK)*, if there is a PPT simulator $\mathcal{S}$ which takes $(y, cl) \in \mathcal{Y} \times \mathcal{CL}$ as input and satisfies the following properties:*

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

(ii) for all $(x, y) \in \mathcal{R}$, the tuple (cm, cl, z), generated via $cl \leftarrow \mathcal{CL}$ and $(cm, z) \leftarrow \mathcal{S}(y, cl)$, has the same distribution as that of a transcript of a conversation between $P(x, y)$ 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 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, where its conversations (cm, cl, z) belong to some space $\mathcal{CM} \times \mathcal{CL} \times \mathcal{Z}$. Let $\mathsf{Hash} : \mathcal{M} \times \mathcal{CM} \to \mathcal{CL}$ 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:

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

According to [22,7], $\mathsf{NIZK}_{\mathrm{FS}}$ 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 $\pi = (z, cl)$), following [7].

Due to the page limitations, the cryptographic assumptions (the discrete logarithm assumption and the decisional Diffie-Hellman assumption) used in our security proofs will be recalled in the full version of this paper.

3 Asymmetric group message franking

In this section, we introduce a primitive called *asymmetric group message franking (AGMF)* and formalize its security notions. Generally, AGMF is a cryptographic primitive providing accountability, deniability and receiver anonymity in group communication scenarios simultaneously.

3.1 AGMF algorithms

We will firstly present the detailed notations of AGMF, and then explain the syntax of the algorithms.

Formally, an asymmetric group message franking (AGMF) scheme $\mathsf{AGMF} = (\mathsf{Setup}, \mathsf{KG}_{\mathrm{J}}, \mathsf{KG}_{\mathrm{u}}, \mathsf{Frank}, \mathsf{Verify}, \mathsf{Judge}, \mathsf{Forge}, \mathsf{RForge}, \mathsf{JForge})$ is a tuple of algorithms associated with a public key space $\mathcal{PK}$, a secret key space $\mathcal{SK}$, a message space $\mathcal{M}$ and a signature space $\mathcal{SG}$. Without loss of generality, we assume that all pk inputs are in $\mathcal{PK}$, all sk inputs are in $\mathcal{SK}$, all m inputs are in $\mathcal{M}$, and all σ inputs are in $\mathcal{SG}$.

The detailed descriptions of the nine algorithms are as follows.

- $pp \leftarrow \mathsf{Setup}(\lambda)$: The setup algorithm takes the security parameter as input, and outputs a global public parameters pp.
- $(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$: The randomized key generation algorithm KG_{J} takes pp as input, and outputs a key pair $(pk_{\mathrm{J}}, sk_{\mathrm{J}})$ for the judge.
- $(pk, sk) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$: The randomized key generation algorithm KG_{u} takes pp as input, and outputs a key pair $(pk_{\mathrm{u}}, sk_{\mathrm{u}})$ for users. Below we usually use $(pk_{\mathrm{s}}, sk_{\mathrm{s}})$ (resp., $(pk_{\mathrm{r}}, sk_{\mathrm{r}})$) to denote sender (resp., receiver) public/secret key pair.

- $\sigma \leftarrow \mathsf{Frank}(pp, sk_{\mathrm{s}}, S, pk_{\mathrm{J}}, m)$: The franking algorithm takes the public parameter pp, a sender's secret key sk_{s}, a polynomial-size receiver's public key set $S = \{pk_{\mathrm{r}_i}\}_{i\in[|S|]} \subset \mathcal{PK}$, the judge's public key pk_{J} and a message m as input, and outputs a signature σ.
- $b \leftarrow \mathsf{Verify}(pp, pk_{\mathrm{s}}, sk_{\mathrm{r}}, pk_{\mathrm{J}}, m, \sigma)$: The *deterministic* receiver verification algorithm takes $(pp, pk_{\mathrm{s}}, sk_{\mathrm{r}}, pk_{\mathrm{J}})$, a message m and a signature σ as input, and outputs a bit b, which indicates that the signature is valid or not.
- $b \leftarrow \mathsf{Judge}(pp, pk_{\mathrm{s}}, sk_{\mathrm{J}}, m, \sigma)$: The *deterministic* judge authentication algorithm takes $(pp, pk_{\mathrm{s}}, sk_{\mathrm{J}})$, a message m and a signature σ as input, and returns $b \in \{0, 1\}$.
- $\sigma \leftarrow \mathsf{Forge}(pp, pk_{\mathrm{s}}, S, pk_{\mathrm{J}}, m)$: The universal forging algorithm, on input $(pp, pk_{\mathrm{s}}, S, pk_{\mathrm{J}})$ and a message m, returns a "forged" signature σ, where $S \subset \mathcal{PK}$.
- $\sigma \leftarrow \mathsf{RForge}(pp, pk_{\mathrm{s}}, (pk_{\mathrm{r}_i}, sk_{\mathrm{r}_i})_{pk_{\mathrm{r}_i}\in S_{\mathrm{cor}}}, S, pk_{\mathrm{J}}, m)$: The receiver compromise forging algorithm takes $(pp, pk_{\mathrm{s}}, (pk_{\mathrm{r}_i}, sk_{\mathrm{r}_i})_{pk_{\mathrm{r}_i}\in S_{\mathrm{cor}}}, S, pk_{\mathrm{J}})$ and a message m as input, and returns a "forged" signature σ, where $S_{\mathrm{cor}} \subset S \subset \mathcal{PK}$.
- $\sigma \leftarrow \mathsf{JForge}(pp, pk_{\mathrm{s}}, S, sk_{\mathrm{J}}, m)$: The judge compromise forging algorithm takes $(pp, pk_{\mathrm{s}}, S, sk_{\mathrm{J}})$ and a message m as input, and outputs a "forged" signature σ, where $S \subset \mathcal{PK}$.

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

Given any pp generated by Setup, any key pairs $(pk_{\mathrm{s}}, sk_{\mathrm{s}})$ and $(pk_{\mathrm{r}}, sk_{\mathrm{r}})$ output by KG_{u}, and any key pair $(pk_{\mathrm{J}}, sk_{\mathrm{J}})$ output by KG_{J}, we require that for any $S \subset \mathcal{PK}$ satisfying $pk_{\mathrm{r}} \in S$, any message $m \in \mathcal{M}$, and any $\sigma \leftarrow \mathsf{Frank}(pp, sk_{\mathrm{s}}, S, pk_{\mathrm{J}}, m)$, it holds that:

(1) $\mathsf{Verify}(pp, pk_{\mathrm{s}}, sk_{\mathrm{r}}, pk_{\mathrm{J}}, m, \sigma) = 1$;
(2) $\mathsf{Judge}(pp, pk_{\mathrm{s}}, sk_{\mathrm{J}}, m, \sigma) = 1$.

3.2 Security notions for AGMF

Now we formalize some security notions for AGMF, including the security notions for accountability, deniability and receiver anonymity of AGMF. Note that we consider the adaptive security in the following games. It means that the adversary $\mathcal{A}$ is allowed to query the corruption oracle on different public keys adaptively, obtaining corresponding secret keys.

Accountability. Analogous to the setting of end-to-end communication, one of the most important security requirements in group scenarios is to prevent malicious impersonation. In other words, AGMF needs to ensure that no one will be impersonated successfully as long as her/his secret key is not compromised. Specifically, AGMF needs to guarantee that (i) no receivers can trick the judge or any receiver in the receiver set (except the adversarial receiver herself if she is also in this set) into accepting a message that is not actually sent by the sender, and (ii) no sender can create a signature such that it is accepted by some receiver but meanwhile rejected by the judge. Following the terminology in AMF [36], we also refer to these security requirements as *receiver binding* and *sender binding*, respectively.

$\underline{\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)}$:
$pp \leftarrow \mathsf{Setup}(\lambda)$; $(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$Q_{\mathrm{sig}} := \emptyset$; $U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$
For $i = 1 \ldots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$(pk^*_{\mathrm{s}}, pk^*_{\mathrm{r}}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, U, pk_{\mathrm{J}})$
If $\mathsf{Verify}(pp, pk^*_{\mathrm{s}}, sk^*_{\mathrm{r}}, pk_{\mathrm{J}}, m^*, \sigma^*) = 1$:
  If $pk^*_{\mathrm{s}}, pk^*_{\mathrm{r}} \notin U_{\mathrm{cor}}$:
    If $\nexists (pk^*_{\mathrm{s}}, S', m^*) \in Q_{\mathrm{sig}}$ s.t. $pk^*_{\mathrm{r}} \in S'$:
      Return 1
If $\mathsf{Judge}(pp, pk^*_{\mathrm{s}}, sk_{\mathrm{J}}, m^*, \sigma^*) = 1$:
  If $(pk^*_{\mathrm{s}} \notin U_{\mathrm{cor}}) \wedge (\nexists (pk^*_{\mathrm{s}}, S', m^*) \in Q_{\mathrm{sig}})$:
    Return 1
Return 0
	$\underline{\mathcal{O}^{\mathsf{Cor}}(pk')}$:
	$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
	Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$
	
	$\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathrm{s}}, S', m')}$:
	$Q_{\mathrm{sig}} \leftarrow Q_{\mathrm{sig}} \cup \{(pk'_{\mathrm{s}}, S', m')\}$
	Return $\mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
	
	$\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathrm{s}}, pk'_{\mathrm{r}}, m', \sigma')}$:
	Return $\mathsf{Verify}(pp, pk'_{\mathrm{s}}, sk'_{\mathrm{r}}, pk_{\mathrm{J}}, m', \sigma')$
	
	$\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathrm{s}}, m', \sigma')}$:
	Return $\mathsf{Judge}(pp, pk'_{\mathrm{s}}, sk_{\mathrm{J}}, m', \sigma')$

$\underline{\mathbf{G}^{\text{s-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)}$:
$pp \leftarrow \mathsf{Setup}(\lambda)$; $(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$
For $i = 1 \ldots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$(pk^*_{\mathrm{s}}, pk^*_{\mathrm{r}}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, U, pk_{\mathrm{J}})$
If $pk^*_{\mathrm{r}} \in U_{\mathrm{cor}}$: Return 0
$b_1 \leftarrow \mathsf{Verify}(pp, pk^*_{\mathrm{s}}, sk^*_{\mathrm{r}}, pk_{\mathrm{J}}, m^*, \sigma^*)$
$b_2 \leftarrow \mathsf{Judge}(pp, pk^*_{\mathrm{s}}, sk_{\mathrm{J}}, m^*, \sigma^*)$
Return $b_1 \wedge \neg b_2$
	$\underline{\mathcal{O}^{\mathsf{Cor}}(pk')}$:
	$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
	Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$
	
	$\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathrm{s}}, S', m')}$:
	Return $\mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
	
	$\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathrm{s}}, pk'_{\mathrm{r}}, m', \sigma')}$:
	Return $\mathsf{Verify}(pp, pk'_{\mathrm{s}}, sk'_{\mathrm{r}}, pk_{\mathrm{J}}, m', \sigma')$
	
	$\underline{\mathcal{O}^{\mathsf{Judge}}(pk'_{\mathrm{s}}, m', \sigma')}$:
	Return $\mathsf{Judge}(pp, pk'_{\mathrm{s}}, sk_{\mathrm{J}}, m', \sigma')$

Fig. 2 Games for defining receiver-binding and sender-binding of AGMF

Now, we present the formal definitions as below.

Definition 3. (r-BIND). *An AGMF scheme* AGMF *is* receiver-binding*, if for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := \Pr[\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1]$$

is negligible, where $\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ is defined in Fig. 2.

Definition 4. (s-BIND). *An AGMF scheme* AGMF *is* sender-binding*, if for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathbf{Adv}^{\text{s-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := \Pr[\mathbf{G}^{\text{s-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1]$$

is negligible, where $\mathbf{G}^{\text{s-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ is defined in Fig. 2.

Remark 1. The receiver binding game $\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ is much more complicated than that in AMF [36], essentially because in the setting of group scenarios, there are multiple receivers. For example, compared with the receiver binding game in AMF, here we additionally need to consider the probability that $\mathcal{A}$ tricks the other honest receivers in the same receiver set. We want to stress that this security model implies unforgeability.

Remark 2. In $\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$, if $\mathcal{A}$ outputs $(pk^*_{\mathrm{s}}, pk^*_{\mathrm{r}}, \sigma^*, m^*)$ such that $\mathsf{Verify}(pp, pk^*_{\mathrm{s}}, sk^*_{\mathrm{r}}, pk_{\mathrm{J}}, m^*, \sigma^*) = 1$, then $\mathcal{A}$ wins only if

$$(pk^*_{\mathrm{s}} \notin U_{\mathrm{cor}}) \wedge (pk^*_{\mathrm{r}} \notin U_{\mathrm{cor}}) \wedge (\nexists\ (pk^*_{\mathrm{s}}, S', m^*) \in Q_{\mathrm{sig}} \text{ s.t. } pk^*_{\mathrm{r}} \in S').$$

That's because (i) if $pk^*_{\mathrm{s}} \in U_{\mathrm{cor}}$ or there is some $(pk^*_{\mathrm{s}}, S', m^*) \in Q_{\mathrm{sig}}$ such that $pk^*_{\mathrm{r}} \in S'$, $\mathcal{A}$ can trivially win; (ii) if $pk^*_{\mathrm{r}} \in U_{\mathrm{cor}}$, $\mathcal{A}$ still can generate such a tuple to win this game by running algorithm RForge.

Remark 3. Compared with the security models of receiver-binding and sender-binding in AMF [36], here we provide the adversary $\mathcal{A}$ with more abilities. For example, in $\mathbf{G}^{\text{r-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ and $\mathbf{G}^{\text{s-bind}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$, $\mathcal{A}$ is allowed to query $\mathcal{O}^{\mathsf{Frank}}$ on $(pk'_{\mathrm{s}}, S', m')$ and query $\mathcal{O}^{\mathsf{Verify}}$ on $(pk'_{\mathrm{s}}, pk'_{\mathrm{r}}, m', \sigma')$, where pk'_{s} can be any users' public keys (including pk^*_{s} and pk^*_{r}), and so can pk'_{r}. The ability is not provided in the receiver/sender binding game of AMF in [36].

Deniability. To support deniability, we need to consider *universal deniability*, *receiver compromise deniability*, and *judge compromise deniability* for AGMF. Generally speaking, universal deniability requires that any non-participating party (i.e., no access to the secret key of the sender, the secret key of any user in the receiver set, or the secret key of the judge) can create a signature, such that for other non-participating parties, it is indistinguishable from honestly-generated signatures. Receiver compromise deniability requires that any corrupted users in the receiver set are able to create a signature, such that for other parties with access to these corrupted users' secret keys, it is indistinguishable from honestly-generated signatures. Judge compromise deniability requires that a party with the judge's secret key is able to create a signature, such that for other parties with access to the judge's secret key, it is indistinguishable from honestly-generated signatures.

The formal definitions are presented as follows.

Definition 5. (UnivDen). *An AGMF scheme* AGMF *is* universally deniable, *if for any PPT adversary* $\mathcal{A}$*, its advantage*

$$\mathbf{Adv}^{\mathrm{UnivDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{UnivDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\mathrm{UnivDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ *is defined in Fig. 3.*

Definition 6. (ReComDen). *An AGMF scheme* AGMF *is* receiver-compromise deniable, *if for any PPT adversary* $\mathcal{A}$*, its advantage*

$$\mathbf{Adv}^{\mathrm{ReComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ *is defined in Fig. 3.*

Definition 7. (JuComDen). *An AGMF scheme* AGMF *is* judge-compromise deniable, *if for any PPT adversary* $\mathcal{A}$*, its advantage*

$$\mathbf{Adv}^{\mathrm{JuComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ *is defined in Fig. 3.*

$\underline{\mathbf{G}^{\mathrm{UnivDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda):}$
$b \leftarrow \{0,1\}$; $pp \leftarrow \mathsf{Setup}(\lambda)$
$(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$; $Q^* := \emptyset$
For $i = 1 \dots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(pp, U, pk_{\mathrm{J}})$
Return $(b = b')$

$\underline{\mathcal{O}^{\mathsf{Cor}}(pk'):}$
If $pk' \in Q^*$: Return $\perp$
$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$

$\underline{\mathcal{O}^{\mathsf{F\text{-}F}}(pk'_{\mathrm{s}}, S', m'):}$
If $S' \cap U_{\mathrm{cor}} \neq \emptyset$: Return $\perp$
$Q^* \leftarrow Q^* \cup S'$
$\sigma_0 \leftarrow \mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
$\sigma_1 \leftarrow \mathsf{Forge}(pp, pk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
Return σ_b

$\underline{\mathbf{G}^{\mathrm{ReComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda):}$
$b \leftarrow \{0,1\}$; $pp \leftarrow \mathsf{Setup}(\lambda)$
$(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$; $Q^* := \emptyset$
For $i = 1 \dots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(pp, U, pk_{\mathrm{J}})$
Return $(b = b')$

$\underline{\mathcal{O}^{\mathsf{Cor}}(pk'):}$
If $pk' \in Q^*$: Return $\perp$
$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$

$\underline{\mathcal{O}^{\mathsf{F\text{-}RF}}(pk'_{\mathrm{s}}, S', S'_{\mathrm{cor}}, m'):}$
If $(S'_{\mathrm{cor}} \nsubseteq S') \vee ((S' \setminus S'_{\mathrm{cor}}) \cap U_{\mathrm{cor}} \neq \emptyset)$: Return $\perp$
$Q^* \leftarrow Q^* \cup (S' \setminus S'_{\mathrm{cor}})$
$\sigma_0 \leftarrow \mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
$\sigma_1 \leftarrow \mathsf{RForge}(pp, pk'_{\mathrm{s}}, (pk_{\mathrm{r}_i}, sk_{\mathrm{r}_i})_{pk_{\mathrm{r}_i} \in S_{\mathrm{cor}}}, S', pk_{\mathrm{J}}, m')$
Return σ_b

$\underline{\mathbf{G}^{\mathrm{JuComDen}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda):}$
$b \leftarrow \{0,1\}$; $pp \leftarrow \mathsf{Setup}(\lambda)$
$(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$; $Q^* := \emptyset$
For $i = 1 \dots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(pp, U, pk_{\mathrm{J}}, sk_{\mathrm{J}})$
Return $(b = b')$

$\underline{\mathcal{O}^{\mathsf{Cor}}(pk'):}$
If $pk' \in Q^*$: Return $\perp$
$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$

$\underline{\mathcal{O}^{\mathsf{F\text{-}JF}}(pk'_{\mathrm{s}}, S', m'):}$
If $S' \cap U_{\mathrm{cor}} \neq \emptyset$: Return $\perp$
$Q^* \leftarrow Q^* \cup S'$
$\sigma_0 \leftarrow \mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$
$\sigma_1 \leftarrow \mathsf{JForge}(pp, pk'_{\mathrm{s}}, S', sk_{\mathrm{J}}, m')$
Return σ_b

Fig. 3 Games for defining universal deniability, receiver compromise deniability, and judge compromise deniability of AGMF

Remark 4. In universal deniability game (resp., judge compromise deniability game), for $\mathcal{A}$'s each $\mathcal{O}^{\mathsf{F\text{-}F}}$-oracle (resp., $\mathcal{O}^{\mathsf{F\text{-}JF}}$-oracle) query $(pk'_{\mathrm{s}}, S', m')$, $\mathcal{A}$ is not allowed to see the secret keys of the receivers in S'. In receiver compromise deniability game, for $\mathcal{A}$'s each $\mathcal{O}^{\mathsf{F\text{-}RF}}$-oracle query $(pk'_{\mathrm{s}}, S', S'_{\mathrm{cor}}, m')$, $\mathcal{A}$ is not allowed to see the secret keys of the receivers in $S' \setminus S'_{\mathrm{cor}}$. We use Q^* to specify the receivers whose secret keys are not provided to $\mathcal{A}$.

Remark 5. Note that in these games, the adversary *is* allowed to access the sender's secret key, as long as the sender is not in the receiver set. Compared with the judge compromise deniability formally defined in AMF [36], where the adversary $\mathcal{A}$ is offered both the receiver's and the judge's keys, our judge compromise deniability only provides the judge's key to $\mathcal{A}$. We stress that the judge compromise deniability formally defined in [36] conflicts with strong authentication (i.e., as pointed out in [36], "forgeries by the moderator cannot be detected by the receiver"). Our judge compromise deniability follows one of the ideas of the judge compromise deniability formalization when considering strong authentication, which is also introduced in [36, Appendix B]. Some more discussions on definitions of deniability will be given in the full version of this paper, due to the space limitations.

Receiver Anonymity. Generally speaking, receiver anonymity requires that any one (except for the receivers in the receiver set), including the judge, cannot tell which receiver set a signature is generated for. With receiver anonymity, the receivers in group communication scenarios can report the malicious messages to the moderator with less concerns.

The formal definition is presented as follows.

Definition 8. (RecAnony). *An AGMF scheme* AGMF *is* receiver anonymous*, if for any PPT adversary $\mathcal{A}$, its advantage*

$$\mathbf{Adv}^{\mathrm{RecAnony}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{RecAnony}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\mathrm{RecAnony}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ is defined in Fig. 4.

$\underline{\mathbf{G}^{\mathrm{RecAnony}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda):}$
$b \leftarrow \{0,1\}$; $pp \leftarrow \mathsf{Setup}(\lambda)$; $(pk_{\mathrm{J}}, sk_{\mathrm{J}}) \leftarrow \mathsf{KG}_{\mathrm{J}}(pp)$
$U := \emptyset$; $U_{\mathrm{key}} := \emptyset$; $U_{\mathrm{cor}} := \emptyset$
$Q^*_{\mathrm{tpl}} := \emptyset$; $Q^* := \emptyset$
For $i = 1 \dots n$:
  $(pk_i, sk_i) \leftarrow \mathsf{KG}_{\mathrm{u}}(pp)$; $U \leftarrow U \cup \{pk_i\}$
  $U_{\mathrm{key}} \leftarrow U_{\mathrm{key}} \cup \{(pk_i, sk_i)\}$
$(pk^*_{\mathrm{s}}, S_0, S_1, m^*, st) \leftarrow \mathcal{A}_1^{\mathcal{O}}(pp, U, pk_{\mathrm{J}}, sk_{\mathrm{J}})$
If $|S_0| \neq |S_1|$: Return 0
If $((S_0 \cup S_1) \cap U_{\mathrm{cor}}) \neq \emptyset$: Return 0
$Q^* \leftarrow Q^* \cup (S_0 \cup S_1)$
$\sigma^* \leftarrow \mathsf{Frank}(pp, sk^*_{\mathrm{s}}, S_b, pk_{\mathrm{J}}, m^*)$
$Q^*_{\mathrm{tpl}} \leftarrow Q^*_{\mathrm{tpl}} \cup \{(pk^*_{\mathrm{s}}, pk_{\mathrm{r}}, m^*, \sigma^*) \mid pk_{\mathrm{r}} \in S_0 \cup S_1\}$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}}(\sigma^*, st)$
Return $(b = b')$

$\underline{\mathcal{O}^{\mathsf{Cor}}(pk'):}$
If $pk' \in Q^*$: Return $\perp$
$U_{\mathrm{cor}} \leftarrow U_{\mathrm{cor}} \cup \{pk'\}$
Return sk' s.t. $(pk', sk') \in U_{\mathrm{key}}$

$\underline{\mathcal{O}^{\mathsf{Frank}}(pk'_{\mathrm{s}}, S', m'):}$
Return $\mathsf{Frank}(pp, sk'_{\mathrm{s}}, S', pk_{\mathrm{J}}, m')$

$\underline{\mathcal{O}^{\mathsf{Verify}}(pk'_{\mathrm{s}}, pk'_{\mathrm{r}}, m', \sigma'):}$
If $pk'_{\mathrm{s}} \in U_{\mathrm{cor}}$: Return $\perp$
If $(pk'_{\mathrm{s}}, pk'_{\mathrm{r}}, m', \sigma') \in Q^*_{\mathrm{tpl}}$: Return $\perp$
Return $\mathsf{Verify}(pp, pk'_{\mathrm{s}}, sk'_{\mathrm{r}}, pk_{\mathrm{J}}, m', \sigma')$

Fig. 4 Game for defining receiver anonymity of AGMF

Discussion. In the following sections, we will present an AGMF scheme achieving the above security features. In fact, our scheme can be proved secure under stronger security models. For example, the receiver anonymity game $\mathbf{G}^{\mathrm{RecAnony}}_{\mathsf{AGMF},\mathcal{A},n}(\lambda)$ in Fig. 4 can be strengthened by allowing the adversary to know the secret keys of the users belonging to $S_0 \cap S_1$. Our scheme also achieves the strengthened receiver anonymity. It is an interesting direction to further strengthen these security models.

4 HPS-based KEM supporting Sigma protocols

In this section, we introduce a new primitive, which we will take as a building block to construct AGMF in Section 5. This primitive is a variant of key encapsulation mechanism (KEM) satisfying that (i) it can be interpreted from the perspective of hash proof system (HPS) [17], and (ii) for some special relations (about the public/secret keys, the encapsulated keys and ciphertexts), there exist corresponding Sigma protocols. We call this primitive *HPS-based KEM supporting Sigma protocols (HPS-KEM$^{\Sigma}$)*. We also provide a concrete construction based on the DDH assumption. Note that our construction can be easily extended to be built based on the k-Linear assumption [27,34].

4.1 Definition

A HPS-KEM$^{\Sigma}$ scheme $\mathsf{HPS\text{-}KEM}^{\Sigma} = (\mathsf{KEMSetup}, \mathsf{KG}, \mathsf{CheckKey}, \mathsf{encap_c}, \mathsf{encap_k}, \mathsf{encap_c^*}, \mathsf{decap}, \mathsf{CheckCwel})$ is a tuple of algorithms associated with a secret key space $\mathcal{SK}$, an encapsulated key space $\mathcal{K}$, where $\mathsf{encap_c}$ and $\mathsf{encap_k}$ have the same randomness space $\mathcal{RS}$, and we denote by $\mathcal{RS}^*$ the randomness space of $\mathsf{encap_c^*}$.

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

To generate a well-formed ciphertext and its corresponding encapsulated key, one can invoke $\mathsf{encap_c}$ and $\mathsf{encap_k}$ at the same time with the same randomness r. For simplicity, we introduce another algorithm encap, and use "$(c, k) \leftarrow \mathsf{encap}(pp, pk; r)$" to denote the procedures "$c \leftarrow \mathsf{encap_c}(pp; r), k \leftarrow \mathsf{encap_k}(pp, pk; r)$". Note that only k contains the information about the public key pk. Correctness is as follows.

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

For any pp generated by $\mathsf{KEMSetup}(1^{\lambda})$, we define some relations as follows:

$$\begin{aligned} \mathcal{R}_{\mathrm{s}} &= \{(sk, pk) : \mathsf{CheckKey}(pp, sk, pk) = 1\} \\ \mathcal{R}_{\mathrm{c,k}} &= \{(r, (c, k, pk)) : (c, k) = \mathsf{encap}(pp, pk; r)\} \\ \mathcal{R}_{\mathrm{c}}^* &= \{(r^*, c) : c = \mathsf{encap_c^*}(pp; r^*)\} \end{aligned} \tag{2}$$

We require that for each relation in Eq. (2), there is a Sigma protocol.

We also require the properties: *universality*, *unexplainability*, *indistinguishability*, *SK-2PR* and *smoothness*, the definitions of which are as follows.

Definition 9. (Universality). *We say that a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *is universal, if for any computationally unbounded adversary $\mathcal{A}$, the advantage*

$$\mathbf{Adv}^{\mathrm{univ}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{univ}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1]$$

is negligible, where $\mathbf{G}^{\mathrm{univ}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 5.*

$\mathbf{G}^{\mathrm{univ}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$:
$pp \leftarrow \mathsf{KEMSetup}(1^{\lambda})$, $(pk, sk) \leftarrow \mathsf{KG}(pp)$
$(c, k, w) \leftarrow \mathcal{A}(pp, pk)$ s.t. $((w, c) \in \mathcal{R}^*_{\mathrm{c}}) \wedge (c \notin \mathcal{C}^{\mathrm{well\text{-}f}}_{pp})$
If $k = \mathsf{decap}(pp, sk, c)$: Return 1
Else Return 0

Fig. 5 Game for defining universality of $\mathsf{HPS\text{-}KEM}^{\Sigma}$

Definition 10. (Unexplainability). *We say that a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *is unexplainable, if for any PPT adversary $\mathcal{A}$, the advantage*

$$\mathbf{Adv}^{\mathrm{unexpl}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{unexpl}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1]$$

is negligible, where $\mathbf{G}^{\mathrm{unexpl}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 6.*

$\mathbf{G}^{\mathrm{unexpl}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$:
$pp \leftarrow \mathsf{KEMSetup}(1^{\lambda})$; $(c, w) \leftarrow \mathcal{A}(pp)$ s.t. $(w, c) \in \mathcal{R}^*_{\mathrm{c}}$
If $c \in \mathcal{C}^{\mathrm{well\text{-}f}}_{pp}$: Return 1
Else Return 0

Fig. 6 Game for defining unexplainability of $\mathsf{HPS\text{-}KEM}^{\Sigma}$

Remark 6. Generally, unexplainability requires that for any PPT adversary, it is difficult to explain a well-formed ciphertext as a result generated with $\mathsf{encap}^*_{\mathrm{c}}$.

Definition 11. (Indistinguishability). *We say that a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *is* indistinguishable, *if for any PPT adversary $\mathcal{A}$, the advantage*

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := |\Pr[\mathbf{G}^{\mathrm{ind}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}|$$

is negligible, where $\mathbf{G}^{\mathrm{ind}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 7.*

Definition 12. (SK-2PR). *We say that a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *is* SK-second-preimage resistant, *if for any PPT adversary $\mathcal{A}$, the advantage*

$$\mathbf{Adv}^{\mathrm{sk\text{-}2pr}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) := \Pr[\mathbf{G}^{\mathrm{sk\text{-}2pr}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda) = 1]$$

is negligible, where $\mathbf{G}^{\mathrm{sk\text{-}2pr}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$ *is defined in Fig. 7.*

Definition 13. (Smoothness). *We say that a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *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 $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp)$, $k \leftarrow \mathcal{K}$, $sk \leftarrow \mathcal{SK}_{pp,pk}$ *and* $k' = \mathsf{decap}(pp, sk, c)$.

$\mathbf{G}^{\mathrm{ind}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$:	$\mathbf{G}^{\mathrm{sk\text{-}2pr}}_{\mathsf{HPS\text{-}KEM}^{\Sigma},\mathcal{A}}(\lambda)$:
$pp \leftarrow \mathsf{KEMSetup}(1^{\lambda})$	$pp \leftarrow \mathsf{KEMSetup}(1^{\lambda})$
$b \leftarrow \{0,1\}$	$(pk, sk) \leftarrow \mathsf{KG}(pp)$
$c_0 \leftarrow \mathsf{encap_c}(pp)$	$sk' \leftarrow \mathcal{A}(pp, pk, sk)$
$c_1 \leftarrow \mathsf{encap}^*_{\mathsf{c}}(pp)$	If $(sk' \neq sk) \wedge (\mathsf{CheckKey}(pp, sk', pk) = 1)$:
$b' \leftarrow \mathcal{A}(pp, c_b)$	Return 1
Return $(b' \stackrel{?}{=} b)$	Return 0

Fig. 7 Games for defining indistinguishability and SK-second-preimage resistance of $\mathsf{HPS\text{-}KEM}^{\Sigma}$

Remark 7. Smoothness guarantees that $\frac{1}{|\mathcal{SK}_{pp,pk}|}$ is a negligible function of λ.

4.2 Construction

Here, we present a concrete construction of HPS-KEM$^{\Sigma}$, which satisfies all the aforementioned security properties. The algorithms are described as follows.

- $\mathsf{KEMSetup}(1^{\lambda})$: Given a security parameter λ, choose a prime-order group $\mathbb{G}$ such that the order of $\mathbb{G}$ is p and the bit-length of p is λ. Then, choose the generators g_1, g_2 of $\mathbb{G}$ uniformly at random. The public parameter is
$$pp = (\mathbb{G}, p, g_1, g_2).$$
- $\mathsf{KG}(pp)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$, choose two randomnesses $(x_1, x_2) \in \mathbb{Z}_p^{*2}$, set $h = g_1^{x_1} g_2^{x_2}$ and the pair of public/secret keys is set as
$$(pk = h,\ sk = (x_1, x_2)).$$
- $\mathsf{CheckKey}(pp, sk, pk)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$ and a pair of public/secret keys $(pk = h,\ sk = (x_1, x_2))$, check whether $g_1^{x_1} g_2^{x_2} = h$ holds. If not, output 0; otherwise output 1.
- $\mathsf{encap_c}(pp; r)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$ and a randomness $r \in \mathbb{Z}_p^*$, output a well-formed encapsulated ciphertext
$$c = (u_1 = g_1^r,\ u_2 = g_2^r).$$
- $\mathsf{encap_k}(pp, pk; r)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$, a public key $pk = h$ and a randomness $r \in \mathbb{Z}_p^*$, output an encapsulated key $k = h^r$.
- $\mathsf{encap}^*_{\mathsf{c}}(pp; r^*)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$ and randomness $r^* = (r, r') \in \mathbb{Z}_p^{*2}$, output a ciphertext
$$c = (u_1 = g_1^r,\ u_2 = g_1^{r'}).$$
- $\mathsf{decap}(pp, sk, c)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$, an encapsulated ciphertext $c = (u_1, u_2)$ and a secret key $sk = (x_1, x_2)$, output an encapsulated key $k' = u_1^{x_1} u_2^{x_2}$.
- $\mathsf{CheckCwel}(pp, c, r^*)$: Given the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$, a ciphertext $c = (u_1, u_2)$ and a random number $r^* = (r, r') \in \mathbb{Z}_p^{*2}$, it outputs 1 if $g_2^r = u_2$; otherwise, it outputs 0.

It is clear that the above construction satisfies *correctness*. Then the relations $\mathcal{R}_{\mathrm{s}}$, $\mathcal{R}_{\mathrm{c,k}}$ and $\mathcal{R}_{\mathrm{c}}^*$ are constructed as follows.

$$\begin{aligned}\mathcal{R}_{\mathrm{s}} &= \{((x_1,x_2),pk) : pk = g_1^{x_1}g_2^{x_2}\}\\ \mathcal{R}_{\mathrm{c,k}} &= \{(r,((u_1,u_2),k,pk)) : u_1 = g_1^r \wedge u_2 = g_2^r \wedge k = pk^r\}\\ \mathcal{R}_{\mathrm{c}}^* &= \{((r,r'),(u_1,u_2)) : u_1 = g_1^r \wedge u_2 = g_1^{r'}\}\end{aligned} \tag{3}$$

We show that there are Sigma protocols for relations $\mathcal{R}_{\mathrm{s}}$, $\mathcal{R}_{\mathrm{c,k}}$ and $\mathcal{R}_{\mathrm{c}}^*$: Okamoto's Sigma protocol [31] for $\mathcal{R}_{\mathrm{s}}$, the Chaum-Pedersen protocol [11] for $\mathcal{R}_{\mathrm{c,k}}$ and Schnorr's Sigma protocol [33] for $\mathcal{R}_{\mathrm{c}}^*$.

We now prove that the above construction satisfies *universality*, *unexplainability*, *indistinguishability*, *SK-second-preimage resistance*, and *smoothness*. Formally, we have the following theorems.

Theorem 2. *The above HPS-KEM$^\Sigma$ scheme is* universal.

Theorem 3. *If the* DL *assumption holds in $\mathbb{G}$, the above HPS-KEM$^\Sigma$ scheme is* unexplainable.

Theorem 4. *If the* DDH *assumption holds in $\mathbb{G}$, the above HPS-KEM$^\Sigma$ scheme is* indistinguishable.

Theorem 5. *If the* DL *assumption holds in $\mathbb{G}$, the above HPS-KEM$^\Sigma$ scheme is* SK-second-preimage resistant.

Theorem 6. *The above HPS-KEM$^\Sigma$ scheme is* smooth.

The proofs of Theorem 2-6 are as follows.

Proof (of Theorem 2). For any computationally unbounded adversary $\mathcal{A}$ attacking universality of HPS-KEM$^\Sigma$, let $(pp = (\mathbb{G},p,g_1,g_2), pk = g_1^{x_1}g_2^{x_2})$ be $\mathcal{A}$'s input, where $(pk, sk = (x_1,x_2))$ are generated by $\mathsf{KG}(pp)$. Denote by $a := \log_{g_1} g_2$. Let $(c = (u_1,u_2), k, w = (r,r'))$ be $\mathcal{A}$'s final output satisfying $((w,c) \in \mathcal{R}_{\mathrm{c}}^*) \wedge (c \notin \mathcal{C}_{pp}^{\text{well-f}})$.

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

As a result,

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

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 $\mathsf{decap}(pp,sk,c) = pk^r \cdot g_1^{(r'-ra)x_2}$ is also uniformly distributed.

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

Proof (of Theorem 3). Suppose that there exists a PPT adversary $\mathcal{A}$ winning the game of unexplainability 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 $g_2 = g^a$, and sends the public parameter $pp = (\mathbb{G}, p, g_1, g_2)$ to $\mathcal{A}$. Then, the adversary $\mathcal{A}$ outputs $(w, c) \in \mathcal{R}_{\mathrm{c}}^*$. Parse $c = (u_1, u_2)$ and $w = (r, r')$. Note that $(w, c) \in \mathcal{R}_{\mathrm{c}}^*$ guarantees that $u_1 = g_1^r$ and $u_2 = g_1^{r'}$. If $\mathcal{A}$ wins the game of unexplainability, then $c \in \mathcal{C}_{pp}^{\text{well-f}}$, which means that $u_1 = g_1^r$ and $u_2 = g_2^r$. In this case, we have $u_2 = g_2^r = g_1^{r'}$. Therefore, $\mathcal{B}$ can output $a = \frac{r'}{r}$ as the solution of the DL problem. $\square$

Proof (of Theorem 4). 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 $pp = (\mathbb{G}, p, g_1, g_2)$ 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 $\mathsf{encap}_{\mathrm{c}}$. 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 $\mathsf{encap}_{\mathrm{c}}^*$. 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. $\square$

Proof (of Theorem 5). 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 $pp = (\mathbb{G}, p, g_1, g_2)$ 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 $\mathsf{CheckKey}(pp, sk', pk) = 1$. 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. $\square$

Proof (of Theorem 6). For any fixed $pp = (\mathbb{G}, p, g_1, g_2)$ 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 encap_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,

$$\begin{aligned} \Delta((c,k),(c,k')) &= \frac{1}{2} \sum_{(\hat{c},\hat{k}) \in \mathcal{C}^* \times \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| \\ &= \frac{1}{2} \sum_{\hat{c} \in S_1} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| \\ &\quad + \frac{1}{2} \sum_{\hat{c} \in S_2} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]|. \end{aligned} \quad (4)$$

We present the following two lemmas with postponed proofs.

Lemma 1. $\sum_{\hat{c} \in S_1} \sum_{\hat{k} \in \mathcal{K}} |\Pr[(c,k) = (\hat{c},\hat{k})] - \Pr[(c,k') = (\hat{c},\hat{k})]| = 0.$

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

Combining Eq. (4), Lemma 1 and Lemma 2, 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 1). 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 $\mathcal{SK}$, 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 $sk \leftarrow \mathcal{SK}$, $k' = \mathsf{decap}(pp, sk, c) = g_1^{rx_1} g_1^{r'x_2} = g_1^{r(b-ax_2)+r'x_2} = h^r g_1^{(r'-ar)x_2}$ 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 2). 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 $k' = \mathsf{decap}(pp, sk, c) = g_1^{rx_1} g_1^{r'x_2} = g_1^{r(b-ax_2)+r'x_2} = h^r g_1^{(r'-ar)x_2} = h^r = h^{\log_{g_1} \hat{u}_1}$, 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. □

5 Generic construction of AGMF from HPS-KEM$^{\Sigma}$

In this section, we provide a framework of constructing AGMF from HPS-KEM$^{\Sigma}$, and show that it achieves the required securities.

Let $\mathsf{HPS\text{-}KEM}^{\Sigma} = (\mathsf{KEMSetup}, \mathsf{KG}, \mathsf{CheckKey}, \mathsf{encap_c}, \mathsf{encap_k}, \mathsf{encap_c^*}, \mathsf{decap}, \mathsf{CheckCwel})$ be a HPS-KEM$^{\Sigma}$ scheme supporting universality, unexplainability, indistinguishability, SK-second-preimage resistance and smoothness, where $\mathcal{RS}$ denotes the randomness space of $\mathsf{encap_c}$ and $\mathsf{encap_k}$, $\mathcal{RS}^*$ denotes the randomness space of $\mathsf{encap_c^*}$, and $\mathcal{K}$ denotes the encapsulated key space.

Our generic AGMF scheme $\mathsf{AGMF} = (\mathsf{Setup}, \mathsf{KG_J}, \mathsf{KG_u}, \mathsf{Frank}, \mathsf{Verify}, \mathsf{Judge}, \mathsf{Forge}, \mathsf{RForge}, \mathsf{JForge})$ is described as follows.

Setup, $\mathsf{KG_J}$ and $\mathsf{KG_u}$ are shown in Fig. 8, where Setup directly invokes the setup algorithm of $\mathsf{HPS\text{-}KEM}^{\Sigma}$, and both $\mathsf{KG_J}$ and $\mathsf{KG_u}$ invoke the key generation algorithm of $\mathsf{HPS\text{-}KEM}^{\Sigma}$.

$\underline{\mathsf{Setup}(\lambda)}$: $pp \leftarrow \mathsf{KEMSetup}(1^{\lambda})$; Return pp
$\underline{\mathsf{KG_J}(pp)}$: $(pk_J, sk_J) \leftarrow \mathsf{KG}(pp)$; Return (pk_J, sk_J)
$\underline{\mathsf{KG_u}(pp)}$: $(pk, sk) \leftarrow \mathsf{KG}(pp)$; Return (pk, sk)

Fig. 8 Algorithm descriptions of Setup, $\mathsf{KG_J}$ and $\mathsf{KG_u}$

The main body of AGMF (i.e., Frank, Verify and Judge) is shown in Fig. 9. Specifically, algorithm Frank calls $\mathsf{encap_c}$ and $\mathsf{encap_k}$ of $\mathsf{HPS\text{-}KEM}^{\Sigma}$ to generate a well-formed ciphertext and encapsulated keys respectively. Besides, it calls a NIZK proof algorithms $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}$ to generate a NIZK proof, where the relation $\mathcal{R}$ is defined in Eq. (5) and $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{PoK}, \mathsf{PoKVer})$ is a NIZK proof using the Fiat-Shamir transform from the Sigma protocols induced by $\mathsf{HPS\text{-}KEM}^{\Sigma}$. The

$\mathsf{Frank}(pp, sk_{\mathrm{s}}, S, pk_{\mathrm{J}}, m)$:
$r \leftarrow \mathcal{RS}$; $c \leftarrow \mathsf{encap}_{\mathrm{c}}(pp; r)$; $k_{\mathrm{J}} \leftarrow \mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{J}}; r)$
For $pk_{\mathrm{r}_i} \in S$:
  $k_{\mathrm{r}_i} \leftarrow \mathsf{encap}_{\mathrm{k}}(pp, pk_{\mathrm{r}_i}; r)$
$x \leftarrow (sk_{\mathrm{s}}, r, \perp)$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$
$\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$; $\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}(\overline{m}, x, y)$
Return $\sigma \leftarrow (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$

$\mathsf{Verify}(pp, pk_{\mathrm{s}}, sk_{\mathrm{r}}, pk_{\mathrm{J}}, m, \sigma)$:
$(\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S}) \leftarrow \sigma$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$
$\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoKVer}(\overline{m}, \pi, y) = 0$: Return 0
If $\mathsf{decap}(pp, sk_{\mathrm{r}}, c) \in \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S}$: Return 1
Return 0

$\mathsf{Judge}(pp, pk_{\mathrm{s}}, sk_{\mathrm{J}}, m, \sigma)$:
$(\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S}) \leftarrow \sigma$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$
$\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$
If $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoKVer}(\overline{m}, \pi, y) = 0$: Return 0
If $\mathsf{decap}(pp, sk_{\mathrm{J}}, c) \neq k_{\mathrm{J}}$: Return 0
Return 1

Fig. 9 Algorithm descriptions of Frank, Verify and Judge

verification algorithm Verify and the moderation algorithm Judge are similar. They first call $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoKVer}$ to check if the NIZK proof is valid, and then call decap with the receiver's/judge's secret key to check whether the encapsulated key and the corresponding decapsulated key are identical or not.

The three forging algorithms (i.e., Forge, RForge, JForge), focusing on different compromise scenarios, are described in Fig. 10. They firstly call $\mathsf{encap}^*_{\mathrm{c}}$ of $\mathsf{HPS\text{-}KEM}^{\Sigma}$ to generate an ill-formed ciphertext. Then, for each one of the receivers (and for the judge) whose secret key is not compromised, randomly samples an encapsulated key from $\mathcal{K}$; for each one of the receivers (and for the judge) whose secret key is compromised, employ decap to generate an encapsulated key. Finally, they call $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}$ to generate a NIZK proof.

For the NIZK proof system $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{PoK}, \mathsf{PoKVer})$ used in Fig. 9 and Fig. 10, we obtain it as follows. The relation $\mathcal{R}$ is defined in Eq. (5).

$$\begin{aligned}\mathcal{R} = \{ \ (\ (sk_{\mathrm{s}}, r, r^*), (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}}) \ ) : \\ (\ (sk_{\mathrm{s}}, pk_{\mathrm{s}}) \in \mathcal{R}_{\mathrm{s}} \wedge (r, (c, k_{\mathrm{J}}, pk_{\mathrm{J}})) \in \mathcal{R}_{\mathrm{c,k}} \ ) \\ \vee \ (\ (r^*, c) \in \mathcal{R}^*_{\mathrm{c}} \ ) \ \}\end{aligned} \tag{5}$$

where $\mathcal{R}_{\mathrm{s}}$, $\mathcal{R}_{\mathrm{c,k}}$ and $\mathcal{R}^*_{\mathrm{c}}$ are defined in Eq. (2). Note that for every sub-relation (i.e., $\mathcal{R}_{\mathrm{s}}, \mathcal{R}_{\mathrm{c,k}}, \mathcal{R}^*_{\mathrm{c}}$), the $\mathrm{HPS\text{-}KEM}^{\Sigma}$ scheme guarantees that there is a Sigma protocol. So, with the technique of trivially combining Sigma protocols for AND/OR proofs [7, Sec. 19.7], we obtain a new Sigma protocol for relation $\mathcal{R}$. Then, using the Fiat-Shamir transform, we derive a NIZK proof system $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{PoK}, \mathsf{PoKVer})$ for $\mathcal{R}$ in the random oracle model.

Now, we provide some explanations about relation $\mathcal{R}$.

$\underline{\mathsf{Forge}(pp, pk_{\mathrm{s}}, S, pk_{\mathrm{J}}, m)\text{:}}$
$r^* \leftarrow \mathcal{RS}^*$; $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$; $k_{\mathrm{J}} \leftarrow \mathcal{K}$
For $pk_{\mathrm{r}_i} \in S$: $k_{\mathrm{r}_i} \leftarrow \mathcal{K}$
$x \leftarrow (\perp, \perp, r^*)$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$; $\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}(\overline{m}, x, y)$
Return $\sigma \leftarrow (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$

$\underline{\mathsf{RForge}(pp, pk_{\mathrm{s}}, \{pk_{\mathrm{r}_i}, sk_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S_{\mathrm{cor}}}, S, pk_{\mathrm{J}}, m)\text{:}}$
// S_{cor} here is the set of corrupted receivers
$r^* \leftarrow \mathcal{RS}^*$; $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$; $k_{\mathrm{J}} \leftarrow \mathcal{K}$
For $pk_{\mathrm{r}_i} \in S \backslash S_{\mathrm{cor}}$: $k_{\mathrm{r}_i} \leftarrow \mathcal{K}$
For $pk_{\mathrm{r}_i} \in S_{\mathrm{cor}}$: $k_{\mathrm{r}_i} \leftarrow \mathsf{decap}(pp, sk_{\mathrm{r}_i}, c)$
$x \leftarrow (\perp, \perp, r^*)$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$; $\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}(\overline{m}, x, y)$
Return $\sigma \leftarrow (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$

$\underline{\mathsf{JForge}(pp, pk_{\mathrm{s}}, S, sk_{\mathrm{J}}, m)\text{:}}$
$r^* \leftarrow \mathcal{RS}^*$; $c \leftarrow \mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$; $k_{\mathrm{J}} \leftarrow \mathsf{decap}(pp, sk_{\mathrm{J}}, c)$
For $pk_{\mathrm{r}_i} \in S$: $k_{\mathrm{r}_i} \leftarrow \mathcal{K}$
$x \leftarrow (\perp, \perp, r^*)$; $y \leftarrow (pp, pk_{\mathrm{s}}, pk_{\mathrm{J}}, c, k_{\mathrm{J}})$; $\overline{m} \leftarrow (m || \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$
$\pi \leftarrow \mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoK}(\overline{m}, x, y)$
Return $\sigma \leftarrow (\pi, c, k_{\mathrm{J}}, \{k_{\mathrm{r}_i}\}_{pk_{\mathrm{r}_i} \in S})$

Fig. 10 Algorithm descriptions of Forge, RForge and JForge

The first part (i.e., $((sk_{\mathrm{s}}, pk_{\mathrm{s}}) \in \mathcal{R}_{\mathrm{s}}) \wedge ((r, (c, k_{\mathrm{J}}, pk_{\mathrm{J}})) \in \mathcal{R}_{\mathrm{c,k}}))$ of the expression of $\mathcal{R}$ contains two sub-parts: (i) $((sk_{\mathrm{s}}, pk_{\mathrm{s}}) \in \mathcal{R}_{\mathrm{s}})$ guarantees the authentication of the sender; (ii) $((r, (c, k_{\mathrm{J}}, pk_{\mathrm{J}})) \in \mathcal{R}_{\mathrm{c,k}})$ guarantees that the ciphertext c and the corresponding encapsulated key k_{J} for the judge are well-formed, and further convinces the receiver that c and k_{J} can be verified successfully by the judge. In other words, once the receiver reports to the judge, the judge will accept the report.

The second part (i.e., $((r^*, c) \in \mathcal{R}^*_{\mathrm{c}})$) of the expression of $\mathcal{R}$ is prepared to guarantee deniability. More specifically, it is prepared for the forgers (including the universal, the receivers and the judge) to construct a valid NIZK proof, since they do not know the sender's secret key. The three forging algorithms in Fig. 10 show that the forgers generate the ill-formed ciphertext via $\mathsf{encap}^*_{\mathrm{c}}(pp; r^*)$. Therefore, the forgers can always obtain the witness r^* for the second part of $\mathcal{R}$.

The relation $\mathcal{R}$ combines the two parts with an "OR" operation, so either the sender or the forgers can generate a valid NIZK proof for $\mathcal{R}$.

Remark 8. In our framework AGMF, in order to reduce the size of signature, k_{J} and k_{r_i} are all encapsulated in the same ciphertext c. This suggests that KG_{J} and KG_{u} are built based on the identical HPS-KEM$^{\Sigma}$. Actually, k_{J} can be encapsulated in another ciphertext, which can be generated with an independent HPS-KEM$^{\Sigma}$. Hence, the judge can run KG_{J} based on an independent HPS-KEM$^{\Sigma}$, to generate the public/secret key pair. In this case, the obtained AGMF can support third-party moderation better.

Correctness. Now we show the correctness of the above scheme AGMF here. For any signature $\sigma \leftarrow \mathsf{Frank}(pp, sk_{\rm s}, S, pk_{\rm J}, m)$ and any $pk_{\rm r} \in S$, we parse $\sigma = (\pi, c, k_{\rm J}, \{k_{{\rm r}_i}\}_{pk_{{\rm r}_i} \in S})$, and let $y := (pp, pk_{\rm s}, pk_{\rm J}, c, k_{\rm J})$ and $\overline{m} := (m || \{k_{{\rm r}_i}\}_{pk_{{\rm r}_i} \in S})$.

We first analyze the output of Verify as follows: (i) the correctness of $\mathsf{NIZK}^{\mathcal{R}}$ guarantees that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoKVer}(\overline{m}, \pi, y) = 1$; (ii) the correctness of $\mathsf{HPS\text{-}KEM}^{\Sigma}$ guarantees that $\mathsf{decap}(pp, sk_{\rm r}, c) \in \{k_{{\rm r}_i}\}_{pk_{{\rm r}_i} \in S}$ since $pk_{\rm r} \in S$. So, Verify will return 1.

Next, we analyze the output of Judge as follows: (i) the correctness of $\mathsf{NIZK}^{\mathcal{R}}$ guarantees that $\mathsf{NIZK}^{\mathcal{R}}.\mathsf{PoKVer}(\overline{m}, \pi, y) = 1$; (ii) the correctness of $\mathsf{HPS\text{-}KEM}^{\Sigma}$ guarantees that $\mathsf{decap}(pp, sk_{\rm J}, c) = k_{\rm J}$. Therefore, Judge will also return 1.

Security. For security, we have the following theorem.

Theorem 7. *If a HPS-KEM$^{\Sigma}$ scheme* $\mathsf{HPS\text{-}KEM}^{\Sigma}$ *is universal, unexplainable, indistinguishble, SK-second-preimage resistant and smooth, and* $\mathsf{NIZK}^{\mathcal{R}} = (\mathsf{PoK}, \mathsf{PoKVer})$ *is a Fiat-Shamir NIZK proof system for* $\mathcal{R}$*, then our scheme* AGMF *achieves the accountability (receiver binding and sender binding), deniability (universal deniability, receiver compromise deniability, and judge compromise deniability) and receiver anonymity simultaneously.*

Due to the page limitations, the proof of Theorem 7 will be provided in the full version of this paper.

Acknowledgements

We would like to express our sincere appreciation to the anonymous reviewers for their valuable comments and suggestions! Junzuo Lai was supported by National Natural Science Foundation of China under Grant No. U2001205, Guangdong Basic and Applied Basic Research Foundation (Grant No. 2023B1515040020), Industrial project No. TC20200930001. Siu Ming Yiu was supported by HKU-SCF FinTech Academy and Shenzhen-Hong Kong-Macao Science and Technology Plan Project (Category C Project: SGDX20210823103537030). Xin Mu was supported by the National Natural Science Foundation of China (62106114). Jian Weng was supported by National Natural Science Foundation of China under Grant Nos. 61825203 and U22B2028, Major Program of Guangdong Basic and Applied Research Project under Grant No. 2019B030302008, National Key Research and Development Plan of China under Grant No. 2020YFB1005600, Guangdong Provincial Science and Technology Project under Grant No. 2021A0505030033, Science and Technology Major Project of Tibetan Autonomous Region of China under Grant No. XZ202201ZD0006G, National Joint Engineering Research Center of Network Security Detection and Protection Technology, and Guangdong Key Laboratory of Data Security and Privacy Preserving.

References

1. Mastodon social network (2018), https://joinmastodon.org/

2. Matrix: an open network for secure, decentralized communication (2018), https://matrix.org/
3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: CRYPTO 2020. pp. 248–277. Springer (2020)
4. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Modular design of secure group messaging protocols and the security of MLS. In: ACM CCS 2021. pp. 1463–1483 (2021)
5. Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In: CRYPTO 2004. pp. 273–289. Springer (2004)
6. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: ASIACRYPT 2001. pp. 514–532. Springer (2001)
7. Boneh, D., Shoup, V.: A graduate course in applied cryptography. Draft 0.5 (2020)
8. Camenisch, J.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zurich (1998)
9. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: CRYPTO 1997. pp. 410–424. Springer (1997)
10. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: CRYPTO 2006. pp. 78–96. Springer (2006)
11. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO 1992. pp. 89–105. Springer (1992)
12. Chen, L., Tang, Q.: People who live in glass houses should not throw stones: Targeted opening message franking schemes. Cryptology ePrint Archive, Report 2018/994 (2018)
13. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In: CCS 2018. pp. 1802–1819 (2018)
14. Corrigan-Gibbs, H., Ford, B.: Dissent: accountable anonymous group messaging. In: CCS 2010. pp. 340–350 (2010)
15. Corrigan-Gibbs, H., Wolinsky, D.I., Ford, B.: Proactively accountable anonymous messaging in verdict. In: USENIX Security 2013. pp. 147–162 (2013)
16. Cramer, R.: Modular design of secure yet practical cryptographic protocols. Ph. D. Thesis, CWI and University of Amsterdam (1996)
17. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: EUROCRYPT. pp. 45–64. Springer (2002)
18. Damgård, I., Haagh, H., Mercer, R., Nitulescu, A., Orlandi, C., Yakoubov, S.: Stronger security and constructions of multi-designated verifier signatures. In: TCC 2020. pp. 229–260. Springer (2020)
19. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: CRYPTO 2018. pp. 155–186. Springer (2018)
20. Facebook: Facebook messenger app (2016), https://www.messenger.com/
21. Facebook: Messenger secret conversations technical whitepaper (2016), https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf
22. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO 1986. pp. 186–194. Springer (1986)
23. Goldberg, I., Ustaoğlu, B., Van Gundy, M.D., Chen, H.: Multi-party off-the-record messaging. In: CCS 2009. pp. 358–368 (2009)

24. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: CRYPTO 2017. pp. 66–97. Springer (2017)
25. Hofheinz, D.: Algebraic partitioning: Fully compact and (almost) tightly secure cryptography. In: TCC 2016-A. vol. 9562, pp. 251–281. Springer (2016)
26. Hofheinz, D.: Adaptive partitioning. In: EUROCRYPT 2017. vol. 10212, pp. 489–518 (2017)
27. Hofheinz, D., Kiltz, E.: Secure hybrid encryption from weakened key encapsulation. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007. pp. 553–571. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
28. Issa, R., AlHaddad, N., Varia, M.: Hecate: Abuse reporting in secure messengers with sealed sender. Cryptology ePrint Archive (2021)
29. Jafargholi, Z., Kamath, C., Klein, K., Komargodski, I., Pietrzak, K., Wichs, D.: Be adaptive, avoid overcommitting. In: CRYPTO 2017. vol. 10401, pp. 133–163. Springer (2017)
30. Leontiadis, I., Vaudenay, S.: Private message franking with after opening privacy. Cryptology ePrint Archive, Report 2018/938 (2018), https://eprint.iacr.org/2018/938
31. Okamoto, T.: An efficient divisible electronic cash scheme. In: CRYPTO 1995. pp. 438–451. Springer (1995)
32. Peale, C., Eskandarian, S., Boneh, D.: Secure complaint-enabled source-tracking for encrypted messaging. In: CCS 2021. p. 14841506 (2021)
33. Schnorr, C.: Efficient identification and signatures for smart cards. In: CRYPTO 1989. pp. 239–252. Springer (1989)
34. Shacham, H.: A Cramer-Shoup Encryption Scheme from the Linear Assumption and from Progressively Weaker Linear Variants. Cryptology ePrint Archive, Report 2007/074 (2007)
35. Syta, E., Corrigan-Gibbs, H., Weng, S.C., Wolinsky, D., Ford, B., Johnson, A.: Security analysis of accountable anonymity in dissent. TISSEC **17**(1), 1–35 (2014)
36. Tyagi, N., Grubbs, P., Len, J., Miers, I., Ristenpart, T.: Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In: CRYPTO 2019. pp. 222–250. Springer (2019)
37. Tyagi, N., Miers, I., Ristenpart, T.: Traceback for end-to-end encrypted messaging. In: CCS 2019. pp. 413–430 (2019)
38. Wolinsky, D.I., Corrigan-Gibbs, H., Ford, B., Johnson, A.: Dissent in numbers: Making strong anonymity scale. In: OSDI 2012. pp. 179–182 (2012)
39. Wong, C.K., Gouda, M., Lam, S.S.: Secure group communications using key graphs. IEEE/ACM Transactions on Networking **8**(1), 16–30 (2000)