The proceedings version of this paper appears at CRYPTO '19. This is the full version.

Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption

Nirvan Tyagi[1], Paul Grubbs[1], Julia Len[1], Ian Miers[1,2], and Thomas Ristenpart[1]

[1] Cornell Tech
[2] University of Maryland

Abstract. Content moderation is crucial for stopping abuse and harassment via messaging on online platforms. Existing moderation mechanisms, such as message franking, require platform providers to see user identifiers on encrypted traffic. These mechanisms cannot be used in messaging systems in which users can hide their identities, such as Signal. The key technical challenge preventing moderation is in simultaneously achieving cryptographic accountability while preserving deniability.

In this work, we resolve this tension with a new cryptographic primitive: *asymmetric message franking* schemes (AMFs). We define strong security notions for AMFs, including the first formal treatment of deniability in moderation settings. We then construct, analyze, and implement an AMF scheme that is fast enough for deployment. We detail how to use AMFs to build content moderation for metadata-private messaging.

Keywords: message franking· designated verifier signatures· deniability· end-to-end encryption· content moderation

Table of Contents

1 Introduction

Billions of users communicate via private messaging on platforms like Facebook, Twitter, and Signal. Unfortunately, these platforms are increasingly used for large-scale spam, harassment, and propagation of fake information. One way platform operators attempt to address these threats is via *content moderation*: allowing the receiver of a message to report it to a moderator. If the moderator determines (via human judgment, machine learning algorithm, or both) the message violated the platform's policies, the platform can ban its sender.

To ensure moderation is not itself abused via fake reports, a receiver's report must contain the sender identity and allow a moderator to verify that the identified sender sent the message. This is challenging if messages are end-to-end (E2E) encrypted because encryption ensures the platform does not learn the messages. However, if the platform can see the sender and receiver identities it can verify reports using specially-constructed ciphertexts that support message franking [36, 42]. In message franking, the platform computes a PRF on a commitment to the plaintext, along with the sender and receiver identities, and gives the output to the receiver to include in a report. To verify a reported message and sender, the platform re-computes the PRF. Because this approach only uses symmetric-key cryptography, we call it *symmetric* message franking (SMF). See Figure 1 for a diagram.

Designs for content moderation of E2E-encrypted messaging have to carefully navigate three security requirements. First, messages should be private by default and the platform should only learn messages that are reported. Second, moderation should achieve *accountability*: given a reported sender and message, the moderator should always be able to verify the sender sent that message. Finally, moderation for E2E-encrypted messages should be *deniable*: only the moderator should be able to verify the report. This protects users from backlash or embarrassment if their messages are posted publicly after a compromise. Deniability was an explicit goal of Facebook's SMF-based moderation system [36].

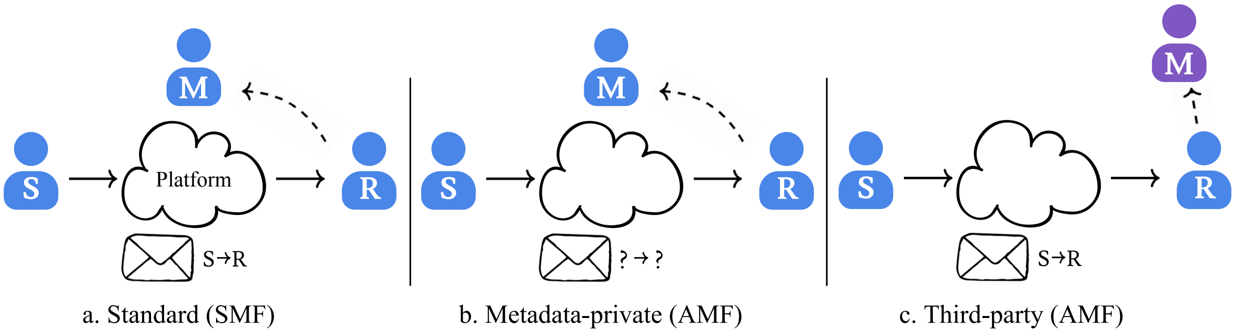
SMF can meet these goals, but care must be taken: for example, Facebook's implementation of SMF did not originally have accountability [35]. If implemented correctly, SMF meets these goals because the commitments are hiding and binding and the PRF includes the sender and receiver identities.

SMF-based approaches require the platform to see the sender and receiver identities when an encrypted message is sent. Therefore, SMFs do not work in *metadata-private* messaging. Such systems (depicted in Figure 1) use E2E encryption and hide the sender or receiver identities from the platform. For example, Signal's recent sealed sender feature, which hides the sender identity, now accounts for over 80% of all Signal traffic [5, 57]. Achieving even stronger metadata privacy, such as also hiding the receiver identity, is an active research area [7, 30, 31, 44, 49–51, 54, 55, 63, 72]. Other approaches to moderation, such as per-message digital signatures, fail to provide deniability. Indeed, obvious approaches using existing cryptographic primitives fail because of a seemingly fundamental tension between accountability and deniability. To make moderation a reality for metadata-private messaging, new cryptography is needed.

Our contributions. This work defines and constructs asymmetric message franking schemes (AMFs). AMFs are public-key and do not require the platform to see metadata. Thus, AMFs resolve the main technical barrier to content moderation for metadata-private messaging. AMFs are similar in some respects to designated-verifier signatures: messages are signed to specific keys, and keys are required for verification. However, building a primitive with the accountability and deniability properties required for moderation necessitates a significant departure from prior work on signatures.

Syntactically, an AMF is a seven-tuple of algorithms: one for generating key pairs, three for accountability, and three for deniability. The accountability algorithms are used to create and verify genuine signatures, and the deniability algorithms are used to forge signatures. Save key generation, each algorithm takes as input either public or private keys from the three parties involved in content moderation: the *sender*, who generates signatures on messages it sends, the *receiver*, who verifies signatures and reports messages, and the *moderator* (or *judge*), who verifies reported content.

The three accountability algorithms are Frank,Verify, and Judge. To generate signatures, the sender uses the signing algorithm Frank. To verify signatures from the sender, the receiver uses its verification algorithm Verify. Similarly, the moderator verifies reports using its verification algorithm Judge. Intuitively, we need two verification algorithms because there are two verifiers who must be able to verify signatures using independent secrets.



a. Standard (SMF) b. Metadata-private (AMF) c. Third-party (AMF)

Fig. 1: Overview of different settings for content moderation of messaging. The solid arrow denotes sending a message across the platform and the dashed arrow denotes reporting a message to the moderator. In the standard setting, messages sent across the platform are associated with sender and receiver identities and the platform is the moderator. In the metadata-private setting, the associated sender and receiver identities of messages are hidden from the platform, and by extension, the moderator. In the third-party setting, the moderator is separate from the platform, and thus also cannot associate sender and receiver identities to messages. Our AMF primitive targets the latter two settings.

To use AMFs for moderation, the sender of a message will use Frank to generate a signature, then E2E-encrypt the message and the signature and send the ciphertext. The receiver first decrypts the ciphertext then verifies the signature using Verify. To make a report, the receiver sends the message and signature to the moderator, who verifies using Judge.

The three deniability algorithms are Forge, RForge, and JForge. These algorithms are not intended to be run by legitimate users: instead, the existence of each of the three algorithms guarantees deniability in a particular compromise scenario. To ensure denials are plausible, we require that there exist concrete and executable algorithms that can forge signatures. Each deniability algorithm takes a message and some combination of public and secret keys of each of the three parties and outputs a forged signature.

AMF security. We formalize security for accountability and deniability of AMFs. For accountability we give three security notions: *unforgeability*, *receiver binding*, and *sender binding*. Unforgeability requires that an attacker that only knows public keys cannot output a forged message and signature pair that fools the receiver. Intuitively, this prevents an external party from impersonating a sender to a receiver. In Appendix A we prove unforgeability is implied by the combination of receiver binding and sender binding.

Unforgeability is similar to the security goal of the same name for regular signatures. If signatures sufficed for moderation we would be done; however, our setting has stronger security requirements: senders should not be able to trick receivers into accepting unreportable messages and receivers should not be able to submit false reports about a sender. Our sender and receiver binding security goals, respectively, address these two threats.

Sender binding requires that an attacker that can choose a sender key pair and adaptively query Verify and Judge oracles cannot create a message and signature pair the receiver will accept but the judge will not. An attacker that can do this can essentially bypass moderation entirely. Such attacks arise in practice: for example, the analogous sender binding goal for SMF was broken for Facebook's SMF scheme [35]. Receiver binding is a complementary notion that requires that no adversarial receiver can trick the moderator into accepting a message not actually sent.

For deniability, we give three security notions: *universal deniability*, *receiver compromise deniability*, and *judge compromise deniability*. Each corresponds to one of the deniability algorithms (Forge, RForge, and JForge respectively) and each models deniability in different key compromise scenarios. Intuitively, each definition guarantees that honestly-generated Frank outputs are indistinguishable from outputs of the forgery algorithm, even to a party that knows some secret keys. Our threat model for deniability is very strong: we want that even if every secret key is compromised, including the judge key, deniability still holds.

The interactions between our accountability and deniability notions are very subtle. Say for example we required the existence of a forge algorithm which can produce an AMF signature indistinguishable from a legitimate one, given just the public keys of sender, receiver, and judge. This contradicts unforgeability and receiver binding since it suggests that anyone can generate signatures that a receiver and judge would accept as valid.

We must therefore limit our first deniability goal, universal deniability, to hold against everyone except the receiver and judge. This goal guarantees the outputs of Forge and Frank are indistinguishable to everyone except those two parties. But this leaves open another way to render AMF signatures undeniable: a receiver (or the judge) could post their secret key to the internet along with a received message and AMF signature. This could serve as undeniable cryptographic evidence that the sender authored the message.

Deniability in these scenarios is ensured by receiver compromise deniability and judge compromise deniability. The former definition corresponds to the case where the receiver's secret key is known and the latter the case where the judge's key is known. The forgeries generated (by RForge and JForge, respectively) should be indistinguishable from the output of Frank, even to distinguishers that know the receiver or both the receiver and judge's secret keys. Again, this intuitively means that a recipient or the judge cannot offer an undeniable proof that a sender sent a message simply by disclosing their secrets: they could equally well have just generated a forgery.

These three deniability definitions are not the only ones possible, and there exists a large space of possible definitions, though many are at odds with accountability. We explore this broader landscape in more detail in Appendix B, and discuss how our deniability targets compare to others in Section 2.

Constructing AMFs. We build a practical AMF scheme. Intuitively the starting point is deniable designated verifier signatures [47], in which a sender signs a message so that only a particular recipient can verify it. We must extend these to allow a moderator (with its own key pair separate from the receiver's) to verify a message; effectively, the moderator is another designated verifier. More subtly, the receiver must be able to verify that a moderator can verify a message, without the moderator's secret key. If this is missing, the scheme would not be sender binding. At the same time, the scheme must support the ability to forge signatures using the three forgery algorithms.

Our eventual construction is based on a non-interactive zero-knowledge proof-of-knowledge of a carefully crafted language that balances the needed verification properties with the ability to forge. At core it uses a Schnorr-style Sigma protocol [70] with a Chaum-Pedersen clause [28]. We build an AMF from the Sigma protocol using the Fiat-Shamir heuristic. The formal analysis is non-trivial. Ultimately we prove theorems that reduce the various required security properties to a combination of standard hardness properties of the underlying group, such as the intractability of computing discrete logs, and the knowledge-of-exponent assumption (KEA), a more exotic but well-studied assumption [10, 32]. In Appendix H, we show a variation of our scheme (with slightly larger signatures) can be proven secure using the standard Gap Diffie-Hellman assumption [16].

We provide a prototype implementation of our AMF construction to showcase its practicality. Our AMF signatures are relatively compact, requiring less than 500 bytes. Even in our unoptimized implementation, signing and verification (by a receiver or the moderator) of a 4 KB message each take 7.3 milliseconds or less. We plan to make our AMF implementations public and open source, to support development of new moderation tools.

Other applications of AMFs. AMFs prove useful in settings beyond metadata-private encrypted messaging. In particular, existing moderation tools based on SMFs cannot support *third-party* moderation, which decouples the platform and moderator. See Figure 1 for a comparison of third-party moderation and other settings. Third-party moderation is necessary in decentralized or federated messaging systems like Matrix [4] or Mastodon [3]. In such systems no single party operates the platform, so the moderator must be distinct. Even in centralized systems like Twitter, third-party moderation is advantageous if the platform cannot adequately moderate messages, or if sub-communities want to enforce their own content policies. Allowing the moderator to be distinct can also enable cross-platform moderation of multiple messaging systems.

Similarly to the metadata-private setting, the moderator does not learn the needed sender and receiver identities associated with messages. In metadata-private settings, this information is cryptographically hidden, whereas in third-party settings it is hidden because the moderator is not inline with the platform. Thus, for third-party moderation, existing approaches, including SMFs, aren't applicable. But AMFs are public-key and so they can easily be used in conjunction with a public-key infrastructure to build third-party moderation.

Summary. This work makes the following contributions:

- We highlight the need for content moderation for metadata-private messaging, and identify a key underlying challenge of balancing accountability with deniability.
- We introduce and formalize a new cryptographic primitive called asymmetric message franking that simultaneously provides the needed authenticity properties for content moderation, while ensuring cryptographic deniability.
- We show how to build an efficient AMF scheme and formally analyze its security. A prototype implementation indicates that AMFs are practical for deployment.

2 Deniability in Messaging

We want AMFs to provide deniability in the event that keys or messages are posted publicly after a compromise. Our setting is therefore most similar to the deniability guarantees sought for designated verifier signatures and proofs [47], but different than settings that allow one to deny encrypted message contents even to an eavesdropper that sees all traffic [23]. An adversary that observes the actual transmission of a message or ciphertext is totally convinced of its origin in our setting. Instead, our concern is not this adversary's conviction, but its ability to convince others. As long as the attacker cannot use what it learns through network manipulation or endpoint compromise to convince others, we have achieved deniability.

The types of deniability guarantees we target have long been a goal in various contexts [25,27], including messaging [36]. The inability to prevent major compromises has made lack of deniability an increasingly pressing concern. In the 2016 United States' and 2017 French presidential elections, certain candidates' systems were compromised and sensitive data was dumped publicly online. DKIM email signatures prevented the Clinton campaign from denying authorship for hacked emails posted by Wikileaks in 2016 [59]. In contrast, in 2017 the Macron campaign was able to effectively deny the authenticity of leaked messages by including decoy messages as a countermeasure [61]. This defense was only possible because of a lack of cryptographic evidence. One result of these breaches is that politicians and others increasingly use E2E encrypted messaging systems that provide deniability [68]. If E2E encryption provides deniability, the cryptography used for moderation must preserve this deniability. This is a crucial reason why AMFs must be deniable.

These examples additionally demonstrate that deniability in messaging is practically important: it is necessary, but not always sufficient, for (what we call) social deniability, i.e., that people are convinced by a denial. Our goal is to ensure that whatever prior belief people have about the likelihood a message is valid should remain unchanged by the use of cryptography, and to have a system that works with other techniques for increasing the success of social deniability (e.g., use of decoys). We do note that because of pervasive propaganda campaigns an awareness has developed among the general public that malicious parties will try to influence popular sentiment by forging content. This would seem to make social deniability more feasible, as people are unlikely to be convinced by an unverified attribution in the era of "fake news".

An important implication of all this is that, to issue a denial that will convince the general public, it is not sufficient to demonstrate the (perhaps non-constructive) *existence* of a forger who *could* have forged a message—there must exist concrete and runnable forgery algorithms that could have been used by influence campaigns or other adversaries. Our eventual construction has three such implemented algorithms for different compromise scenarios; see Section 4 for more details.

3 Syntax and Security Notions

We introduce a new primitive, *asymmetric message franking* (AMF), that provides the cryptographic algorithms needed for secure metadata-private moderation. We will present the algorithms and security definitions of an AMF scheme in three parts. First, we present a brief preliminary on key generation. Then, we describe the accountability algorithms and definitions. Finally, we present the three algorithms used for deniability and definitions.

Formally, an asymmetric message franking scheme AMF $=$ (KeyGen, Frank, Verify, Judge, Forge, RForge, JForge) is a tuple of seven algorithms. An AMF scheme is associated with a public key space $\mathcal{PK}$, secret key space $\mathcal{SK}$, message space $\mathcal{M}$, and signature space Σ. To simplify notation of inputs in the algorithms, we assume all pk inputs are in $\mathcal{PK}$, all sk inputs are in $\mathcal{SK}$, all msg inputs are in $\mathcal{M}$, and all σ inputs are in Σ.

AMF key generation. AMF key generation, $(pk, sk) \leftarrow\$\ \mathsf{KeyGen}$, is a randomized key generation algorithm which outputs a public key pair $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$. We assume the public key pk can be uniquely recovered from the private key sk. Our schemes have this property. We also assume for simplicity that the judge, senders, and receivers all use the same key generation algorithm.

We will assume that key pairs can be confirmed to be valid. More precisely, we will use later a deterministic algorithm $\mathsf{WellFormed}\colon\ \mathcal{PK} \times \mathcal{SK} \to \{0, 1\}$ which takes as input a key pair $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$ and outputs a bit b denoting whether the key pair is a valid pair ($b = 1$) or not ($b = 0$). The purpose of this procedure is to verify that a (possibly adversarially chosen) key pair is *well-formed* relative to some relationship between pk and sk. In our schemes this will be a single exponentiation.

Our formalization of AMFs excludes deployment considerations such as the public key infrastructure and identity-to-public key mappings: see Section 6 and Section 7 for more discussion.

3.1 AMF Algorithms and Security Notions: Accountability

For an AMF = (KeyGen, Frank, Verify, Judge, Forge, RForge, JForge), the three accountability algorithms are Frank, Verify, and Judge. These algorithms are used for creating and verifying signatures. We explain the syntax of each algorithm in turn, then describe the corresponding accountability security notions.

- $\sigma \leftarrow\$\ \mathsf{Frank}(sk_s, pk_r, pk_j, msg)$: The (randomized) message signing or *franking* algorithm takes as input a receiver public key pk_r, a judge public key pk_j, a sender secret key sk_s, and a message msg. It outputs a signature σ.
- $b \leftarrow \mathsf{Verify}(pk_s, sk_r, pk_j, msg, \sigma)$: The deterministic receiver verification algorithm takes as input a sender public key pk_s, receiver secret key sk_r, judge public key pk_j, message msg, and signature σ, then outputs a bit. The receiver runs this to ensure the message, signature pair (msg, σ) is well-formed and reportable to the judge.
- $b \leftarrow \mathsf{Judge}(pk_s, pk_r, sk_j, msg, \sigma)$: The deterministic judge authentication algorithm takes as input a sender public key pk_s, receiver public key pk_r, judge secret key sk_j, message msg, and signature σ, then outputs a bit. This algorithm is used by the judge to check the authenticity of reported messages, ensuring the message was really sent from the sender and was meant for the recipient.

This formalization restricts attention to non-interactive schemes for which franking, verification, and judging requires sending just a single message. Such non-interactive schemes have important practical benefits, but it is conceivable that there might be some benefits of generalizing our treatment to include interactive schemes, which we leave for future work.

Correctness. We require that our schemes be correct. Informally, this means AMF signatures created by the franking algorithm are both verified and judged successfully. Formally, for all messages ,msg, and for all pairs of public keys, $(pk_{\{s,r,j\}}, sk_{\{s,r,j\}})$, it holds that

$$\Pr\left[\, \mathsf{Verify}(pk_s, sk_r, pk_j, msg,\ \mathsf{Frank}(sk_s, pk_r, pk_j, msg)) = 1 \,\right] = 1$$

and

$$\Pr\left[\, \mathsf{Judge}(pk_s, pk_r, sk_j, msg,\ \mathsf{Frank}(sk_s, pk_r, pk_j, msg)) = 1 \,\right] = 1$$

where the probabilities are taken over the random coins used in Frank.

Security notions for accountability. First and foremost an AMF scheme should prevent a party from impersonating a sender to a receiver. This goal, which we call *unforgeability*, is a lifting of standard digital signature unforgeability to the setting of AMF schemes. As discussed above, AMFs should also (1) prevent any sender from creating a signature that can be verified by the receiver but not the moderator, and (2) prevent any receiver from creating a signature on a message that wasn't sent. Following the terminology used in symmetric message franking [42] we refer to these goals as *sender binding* and *receiver binding*, respectively. It turns out sender binding and receiver binding together imply unforgeability. We proceed by formalizing the sender binding and receiver binding accountability notions and defer the formalization of unforgeability along with its reduction to receiver binding and sender binding to Appendix A.

```
$\text{r-BIND}^{\mathcal{A}}_{\text{AMF}}$:
$(pk_s, sk_s) \leftarrow\!\$\ \mathsf{KeyGen}$
$(pk_j, sk_j) \leftarrow\!\$\ \mathsf{KeyGen}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
if $(pk_r, msg) \in Q$:
    return 0
return $\mathsf{Judge}(pk_s, pk_r, sk_j, msg, \sigma)$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
return $\mathsf{Frank}(sk_s, pk'_r, pk_j, msg)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
return $\mathsf{Judge}(pk'_s, pk_r, sk_j, msg, \sigma)$
```

```
$\text{s-BIND}^{\mathcal{A}}_{\text{AMF}}$:
$(pk_r, sk_r) \leftarrow\!\$\ \mathsf{KeyGen}$
$(pk_j, sk_j) \leftarrow\!\$\ \mathsf{KeyGen}$
$(pk_s, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_r, pk_j)$
$b_v \leftarrow \mathsf{Verify}(pk_s, sk_r, pk_j, msg, \sigma)$
$b_j \leftarrow \mathsf{Judge}(pk_s, pk_r, sk_j, msg, \sigma)$
return $b_v \wedge \neg b_j$

$\mathcal{O}^{\mathsf{Verify}}(pk'_s, msg, \sigma)$:
return $\mathsf{Verify}(pk'_s, sk_r, pk_j, msg, \sigma)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
return $\mathsf{Judge}(pk'_s, pk_r, sk_j, msg, \sigma)$
```

Fig. 2: Accountability games for AMF schemes: receiver binding (left) and sender binding (right).

We formalize security using the code-based game approach of Bellare and Rogaway [13]. We will use a concrete security approach in which we account for adversarial resources explicitly in theorem statements, rather than defining security asymptotically. Asymptotic notions can be derived from our treatment in a straightforward way.

Receiver binding is specified formally in game r-BIND on the left-hand side of Figure 2. The adversary is given a Frank oracle for some (honest) sender, to which it can query chosen receiver public key and message pairs. We also give the adversary access to a Judge oracle to query chosen sender public key and message pairs, and the ability to choose receiver keys adversarially. It tries to output a message and signature, distinct from all Frank oracle outputs, for which Judge outputs 1. For an adversary $\mathcal{A}$ and message franking scheme AMF we define the r-BIND advantage of $\mathcal{A}$ against AMF as

$$\mathbf{Adv}^{\text{r-bind}}_{\text{AMF}}(\mathcal{A}) = \Pr\left[\, \text{r-BIND}^{\mathcal{A}}_{\text{AMF}} \Rightarrow 1 \,\right] ,$$

where the probability here (and for subsequent use of games) is over all the random coins used in the game, including those of the adversary.

Sender binding is specified formally in game s-BIND on the right-hand side of Figure 2. The adversary is given a pair of oracles for Verify and Judge to which it can query chosen sender public key, message, and signature triples. Its goal is to generate, for some adversarially chosen sender public key, an AMF signature that Verify validates but Judge rejects. For an adversary $\mathcal{A}$ and message franking scheme AMF we define the s-BIND advantage of $\mathcal{A}$ against AMF as

$$\mathbf{Adv}^{\text{s-bind}}_{\text{AMF}}(\mathcal{A}) = \Pr\left[\, \text{s-BIND}^{\mathcal{A}}_{\text{AMF}} \Rightarrow 1 \,\right] .$$

3.2 AMF Algorithms and Security Notions: Deniability

To support deniability, we equip AMF schemes with three deniability algorithms and associate to each a security notion. We include the forging algorithms as part of the scheme to emphasize their importance in providing practically-meaningful deniability guarantees. They will be efficient to execute and as easy to implement as the other algorithms. The deniability algorithms for an AMF scheme AMF = (KeyGen, Frank, Verify, Judge, Forge, RForge, JForge) are Forge, RForge, and JForge. We give a formal description of each along with some intuition about the deniability setting they correspond to.

Universal deniability requires that any non-participating party (no access to sender, receiver, or judge secret keys) can forge a signature that is indistinguishable from honestly-generated signatures to other non-participating parties. Intuitively, this allows the sender to claim a message originated from any non-participating party. This is the purpose of the Forge algorithm of a AMF scheme.

- $\sigma \leftarrow\!\$\ \mathsf{Forge}(pk_s, pk_r, pk_j, msg)$: The forge algorithm takes a sender public key pk_s, receiver public key pk_r, a judge public key pk_j, and a message msg, then outputs a "forged" AMF signature σ.

```
$\underline{\mathrm{UnivDen}^{\mathcal{A},b}_{\mathrm{AMF}}:}$
$(pk_s, sk_s) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_r, sk_r) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_j, sk_j) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Frank}}}(sk_s, pk_r, pk_j)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$\sigma_0 \leftarrow \mathsf{Frank}(sk_s, pk_r, pk_j, msg)$
$\sigma_1 \leftarrow \mathsf{Forge}(pk_s, pk_r, pk_j, msg)$
$\mathbf{return}\ \sigma_b$
```

```
$\underline{\mathrm{RecCompDen}^{\mathcal{A},b}_{\mathrm{AMF}}:}$
$(pk_s, sk_s) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_j, sk_j) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_r, sk_r, aux) \leftarrow \mathcal{A}_1(pk_s, pk_j)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
$\mathbf{if}\ b^r_{\mathrm{wf}} \neq 1$:
    $\mathbf{return}\ 0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}}(sk_s, sk_r, pk_j, aux)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$\sigma_0 \leftarrow \mathsf{Frank}(sk_s, pk_r, pk_j, msg)$
$\sigma_1 \leftarrow \mathsf{RForge}(pk_s, sk_r, pk_j, msg)$
$\mathbf{return}\ \sigma_b$
```

```
$\underline{\mathrm{JudgeCompDen}^{\mathcal{A},b}_{\mathrm{AMF}}:}$
$(pk_s, sk_s) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_r, sk_r, pk_j, sk_j, aux) \leftarrow \mathcal{A}_1(pk_s)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
$b^j_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_j, sk_j)$
$\mathbf{if}\ b^r_{\mathrm{wf}} \wedge b^j_{\mathrm{wf}} \neq 1$:
    $\mathbf{return}\ 0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}}(sk_s, sk_r, sk_j, aux)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$\sigma_0 \leftarrow \mathsf{Frank}(sk_s, pk_r, pk_j, msg)$
$\sigma_1 \leftarrow \mathsf{JForge}(pk_s, pk_r, sk_j, msg)$
$\mathbf{return}\ \sigma_b$
```

Fig. 3: Deniability security games for AMF schemes: universal deniability (left), receiver compromise deniability (middle), and judge compromise deniability (right).

We formalize universal deniability in game UnivDen, the leftmost in Figure 3. For an adversary $\mathcal{A}$ and asymmetric message franking scheme AMF we define the UnivDen advantage of $\mathcal{A}$ against AMF as

$$\mathbf{Adv}^{\text{univ-den}}_{\mathrm{AMF}}(\mathcal{A}) = \left| \Pr\left[\mathrm{UnivDen}^{\mathcal{A},0}_{\mathrm{AMF}} \Rightarrow 1 \right] - \Pr\left[\mathrm{UnivDen}^{\mathcal{A},1}_{\mathrm{AMF}} \Rightarrow 1 \right] \right| .$$

Receiver compromise deniability requires that a party with access to the receiver's secret key can forge a signature that is indistinguishable from honestly-generated signatures to other parties with access to the receiver's secret key. This captures deniability in the case where the receiver's secret key has become compromised, and allows the sender to claim a message originates from a compromising party or malicious receiver. The RForge algorithm is used for receiver compromise deniability.

- $\sigma \leftarrow\!\!\$\ \mathsf{RForge}(pk_s, sk_r, pk_j, msg)$: The receiver forge algorithm takes a sender public key pk_s, receiver secret key sk_r, a judge public key pk_j, and a message msg, then outputs a "forged" AMF signature σ.

We formalize receiver compromise deniability in game RecCompDen, the middle game in Figure 3. For an adversary $\mathcal{A}$ and message franking scheme AMF we define the RecCompDen advantage of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against AMF as

$$\mathbf{Adv}^{\text{r-den}}_{\mathrm{AMF}}(\mathcal{A}) = \left| \Pr\left[\mathrm{RecCompDen}^{\mathcal{A},0}_{\mathrm{AMF}} \Rightarrow 1 \right] - \Pr\left[\mathrm{RecCompDen}^{\mathcal{A},1}_{\mathrm{AMF}} \Rightarrow 1 \right] \right| .$$

Judge compromise deniability requires that a party with access to the judge's secret key can forge a signature that is indistinguishable from honestly-generated signatures to other parties with access to the judge's secret key. Symmetrically, this captures deniability in the case where the judge's secret key has become compromised, and allows the sender to claim a message originates from a compromising party or malicious judge. The JForge algorithm is used for judge compromise deniability.

- $\sigma \leftarrow\!\!\$\ \mathsf{JForge}(pk_s, pk_r, sk_j, msg)$: The judge forge algorithm takes a sender public key pk_s, receiver public key pk_r, a judge secret key sk_j, and a message msg, then outputs a "forged" AMF signature σ.

We formalize judge compromise deniability in game JudgeCompDen, the right-most game in Figure 3. For an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and message franking scheme AMF we define the JudgeCompDen advantage of $\mathcal{A}$ against AMF as

$$\mathbf{Adv}^{\text{j-den}}_{\mathrm{AMF}}(\mathcal{A}) = \left| \Pr\left[\mathrm{JudgeCompDen}^{\mathcal{A},0}_{\mathrm{AMF}} \Rightarrow 1 \right] - \Pr\left[\mathrm{JudgeCompDen}^{\mathcal{A},1}_{\mathrm{AMF}} \Rightarrow 1 \right] \right| .$$

Random oracle model. Looking ahead, we will prove security in the random oracle model. In this model, to each definition we add another procedure $\mathcal{O}_{\mathrm{ro}}$. The adversary $\mathcal{A}$ and algorithms Forge, Verify, Judge, Forge, JForge, RForge all have access to it as an oracle. The oracle accepts queries on arbitrary length bit strings m and returns a random bit string r of length hlen. It stores r in a table T indexed by m to answer future queries consistently. In some security proofs we will use a technique referred to as programming the random oracle (setting certain RO outputs to values in a way advantageous to a reduction). Importantly, however, our definitions ensure that the AMF forging algorithms only have access to the oracle (as does the adversary), forcing them to forge without modifying the RO mapping. This means that when we apply the ROM heuristic, instantiating the RO with a hash function such as SHA-256, the forge algorithms can still be executed.

Space of deniability definitions. Notice that our deniability definitions are implicitly parameterized by the combination of secrets keys given to the forger and the combination of secret keys given to the distinguisher, i.e., who is able to fool whom. In this work, we target three specific deniability definitions within this space that we believe have real-world significance. However, this is not the only set of meaningful deniability definitions that one might desire from a scheme. Consider the following two examples. First, our definitions give the distinguisher access to the sender's secret key which models deniability in the face of sender compromise. An alternative definition may dispense with this goal in favor of an accountability notion, disavowability, in which a sender has the ability to cryptographically prove forged signatures were not created using their sender secret key, i.e., disavow forgeries. Second, our judge compromise deniability definition conflicts with strong authentication between sender and receiver — forgeries by the moderator cannot be detected by the receiver. Instead, a stronger unforgeability definition could be satisfied in which the judge's secret key alone is not sufficient to forge messages accepted by the receiver.

Ultimately, there exist many different trade-offs between deniability and accountability within this definition space. We provide a more detailed exploration of the space of possible deniability definitions along with their relationships to various accountability notions in Appendix B.

4 Construction

In this section, we present our construction for building an asymmetric message franking scheme. First, we give intuition for our approach by drawing connections to the literature on designated verifier signatures [47]. Then, we describe our particular instantiation built using signatures of knowledge [21] and detailed in Figure 5.

4.1 Intuition: AMF from Designated Verifiers

Designating the moderator as verifier. The tension between accountability and deniability arises from the desire for franking signatures to be forgeable (deniability) as well as verifiable by certain special parties, e.g. the moderator (accountability). This suggests *designated verifier* signatures [47] as a natural starting point from which to build asymmetric message franking. The sender can designate the moderator as a verifier for a signature of the message.

A designated verifier signature or, more generally, a designated verifier proof system allows a prover to provide a proof of a statement that convinces a designated verifier but no one else. The designated verifier can efficiently forge the proof such that the forged proof is indistinguishable from a real proof even with access to the designated verifier's secret key. This security property, known as non-transferability, ensures there are two possible parties that could have created the signature, the alleged sender or the (compromised) moderator. It matches closely to receiver compromise deniability and judge compromise deniability for AMFs which extends the idea of non-transferability to relationships between three parties.

Universal deniability from strong designated verifiers. To expand the set of possible forgers to any non-participating party, i.e. universal deniability, we additionally make use of a strong deniability property of *strong designated verifier* signatures [45,47,69]. This property allows anyone to be able to forge a signature between two parties such that the forgery is indistinguishable from real signatures to anyone without secret key access. Without care, universal deniability poses a problem for accountability, namely sender binding. Consider a franking signature that consists of the sender creating a strong designated verifier signature for the moderator. A sender can send an abusive message and sign with a universal forgery. If the recipient of the message attempts to report to the moderator, the moderator will not be convinced the message was sent by the sender.

Chaining designated verifier proofs. To achieve sender binding, the receiver must have some way of verifying whether messages it receives are reportable to the moderator. Specifically, the receiver must be able to verify the sender's strong designated verifier signature for the moderator is well-formed and not a forgery. This leads us to the final step: the sender can attach a strong designated verifier proof for the receiver *proving* that the strong designated verifier signature for the moderator is well-formed. By using a strong designated verifier proof for this step, the deniability goals are preserved.

The challenge in building AMFs with this approach is in instantiating schemes such that the signing algorithm of the strong designated verifier signature falls into a language compatible with the strong designated verifier proof system. Existing strong designated verifier signatures [45, 47, 69] do not appear to have this desired structure-preserving property [6] that would lend to using efficient proof systems. Additionally, we are not aware of any general-purpose strong designated verifier proof systems for arbitrary languages. While such a proof system can presumably be constructed using non-interactive zero knowledge proof systems for arbitrary languages [41], such a solution would likely be prohibitively expensive for low latency messaging. Despite these challenges, the question of building AMFs from designated verifier primitives remains interesting and we discuss such a generic construction in Appendix G. We next turn to building practical AMFs.

4.2 AMF from Signatures of Knowledge

While we do not build off the abstraction of designated verifiers, our construction is modeled off the intuition that an AMF can be composed of a strong designated verifier proof to the receiver of the well-formedness of a strong designated signature to the moderator. Our construction is inspired by the strong designated verifier signature scheme of Huang et al. built using signatures of knowledge [45], which we modify to allow for proofs of well-formedness.

Our construction can be based on any suitable cyclic group. In the following we let $\mathbb{G}$ be a group, let p be its order, and g be a generator for $\mathbb{G}$. We use multiplicative notation, though note that we use elliptic curve groups in our implementation (Section 6). Secret keys are uniformly chosen from $\mathcal{SK} = \mathbb{Z}_p$, and public keys are set to be $pk = g^{sk}$. We denote this key generation as $\mathsf{PKKeyGen}$. Note that it is easy to check the well-formedness of such keys.

Signatures of knowledge. First, we introduce our treatment of signatures of knowledge. These can be thought of as a cross between non-interactive proofs of knowledge and digital signatures. We use a standard Fiat-Shamir signature scheme [38] in which we can produce signatures of knowledge from basic Sigma protocols by including the message in the hash producing the challenge. Our construction uses Schnorr proofs of knowledge of discrete logarithm [70] and Chaum-Pedersen proofs of equality of discrete log [28], extended with conjunctions and disjunctions (logical ANDs and ORs) [17].

Our notation follows closely to that of Camenisch [21]. A signature of knowledge scheme $\mathsf{SPoK}^{\mathcal{R}} = (\mathsf{prove}, \mathsf{verify})$ is a pair of algorithms associated with a witness-statement relation $\mathcal{R}$. A relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ is defined relative to a set $\mathcal{X}$ called the witness space and set $\mathcal{Y}$ called the statement space. The randomized proving algorithm, prove, outputs a signature proof of the statement for a message given a witness, $\pi \leftarrow\!\!\$\ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x)$. The proving algorithm should return a dedicated symbol $\perp$ if $(x, y) \notin \mathcal{R}$ though for brevity we exclude such checks from pseudocode. The deterministic verification algorithm, verify, takes as input a message, signature proof, and statement, then returns a bit indicating whether verification is successful, $b \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$. As an example, this allows us to create signature proofs of the form: $\mathcal{R} = \big\{\big((\alpha, \beta), (g, A, B)\big) \ :\ A = g^{\alpha} \vee B = g^{\beta}\big\}$, which can be proved with knowledge of either α or β with witnesses $(\alpha, \perp)$ or $(\perp, \beta)$ respectively. Note that the inclusion of $\perp$ symbols in the witness explicitly indicates which side of the disjunction is satisfied.

We will utilize two security properties of the Sigma protocols from which we derive our Fiat-Shamir signatures of knowledge: knowledge soundness and honest-verifier zero knowledge. Briefly, knowledge soundness ensures that a prover that generates a valid signature proof for a message must actually "know" a witness for the statement. A scheme being zero knowledge ensures that verification of a proof does not reveal anything

		How to prove first clause?	**How to prove second clause?**		
Algorithm	**Security notion**	$(pk_s = g^t \vee J = g^u)$	$((J = (pk_j)^v \wedge E_J = g^v) \vee R = g^w)$	Verify?	Judge?
Frank	Correctness	$\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p;\ J \leftarrow (pk_j)^\alpha;\ t = sk_s$	$\beta \overset{\$}{\leftarrow} \mathbb{Z}_p;\ R \leftarrow (pk_r)^\beta;\ v = \alpha$	✓	✓
Forge	Universal deniability	$\gamma \overset{\$}{\leftarrow} \mathbb{Z}_p;\ J \leftarrow g^\gamma;\ u = \gamma$	$\delta \overset{\$}{\leftarrow} \mathbb{Z}_p;\ R \leftarrow g^\delta;\ w = \delta$	×	×
RForge	Receiver compromise deniability	$\gamma \overset{\$}{\leftarrow} \mathbb{Z}_p;\ J \leftarrow g^\gamma;\ u = \gamma$	$\beta \overset{\$}{\leftarrow} \mathbb{Z}_p;\ R \leftarrow (pk_r)^\beta;\ w = \beta \cdot sk_r$	✓	×
JForge	Judge compromise deniability	$\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p;\ J \leftarrow (pk_j)^\alpha;\ u = \alpha \cdot sk_j$	$\beta \overset{\$}{\leftarrow} \mathbb{Z}_p;\ R \leftarrow (pk_r)^\beta;\ v = \alpha$	✓	✓

Fig. 4: Summary of how AMF signing and forging algorithms construct signatures. The rightmost columns indicate with a checkmark (✓) which verification algorithms accept that signature and with a cross (×) which will reject that signature.

$$\mathcal{R} = \big\{\big((t,\, u,\, v, w),\, (g,\, pk_s,\, pk_j,\, J,\, R,\, E_J)\big)\ :\ \big(pk_s = g^t \vee J = g^u\big)\ \wedge\ \big((J = (pk_j)^v \wedge E_J = g^v) \vee R = g^w\big)\big\}$$

```
$\mathsf{Frank}(sk_s, pk_r, pk_j, msg)$:
$(\alpha, \beta) \overset{\$}{\leftarrow} (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (sk_s, \bot, \alpha, \bot)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \overset{\$}{\leftarrow} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
return $(\pi, J, R, E_J, E_R)$
```

```
$\mathsf{Forge}(pk_s, pk_r, pk_j, msg)$:
$(\alpha, \beta, \gamma, \delta) \overset{\$}{\leftarrow} (\mathbb{Z}_p)^4$
$J \leftarrow g^\gamma$
$R \leftarrow g^\delta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\bot, \gamma, \bot, \delta)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \overset{\$}{\leftarrow} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
return $(\pi, J, R, E_J, E_R)$
```

```
$\mathsf{RForge}(pk_s, sk_r, pk_j, msg)$:
$(\alpha, \beta, \gamma) \overset{\$}{\leftarrow} (\mathbb{Z}_p)^3$
$J \leftarrow g^\gamma$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x = (\bot, \gamma, \bot, \beta \cdot sk_r)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \overset{\$}{\leftarrow} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
return $(\pi, J, R, E_J, E_R)$
```

```
$\mathsf{JForge}(pk_s, pk_r, sk_j, msg)$:
$(\alpha, \beta) \overset{\$}{\leftarrow} (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\bot, \alpha \cdot sk_j, \alpha, \bot)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \overset{\$}{\leftarrow} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
return $(\pi, J, R, E_J, E_R)$
```

```
$\mathsf{KeyGen}$:
return $\mathsf{PKKeyGen}$
```

```
$\mathsf{Verify}(pk_s, sk_r, pk_j, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow R = E_R^{sk_r}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
return $b_1 \wedge b_2$
```

```
$\mathsf{Judge}(pk_s, pk_r, sk_j, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
return $b_1 \wedge b_2$
```

Fig. 5: Algorithms for our deniable AMF scheme. The relation $\mathcal{R}$ defining our SPoK is depicted at the top.

about the witness to the verifier other than if it is valid or not. The complete descriptions for constructing signatures of knowledge from Sigma protocols along with formalizations of these security properties are deferred to Appendix D.

Overview of construction. Consider the strong designated verifier signature derived as a signature of knowledge from the following relation:

$$\mathcal{R}_{\mathsf{SDVS}} = \big\{\big((t,\, u),\, (g,\, pk_s,\, J)\big)\ :\ pk_s = g^t \vee J = g^u\big\},$$

in which an honest sender will construct Diffie-Hellman value $J = (pk_j)^\alpha$ for random choice of $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p$, and send ephemeral value $E_J = g^\alpha$ along with the $\mathsf{SPoK}^{\mathcal{R}_{\mathsf{SDVS}}}$ signature proof, where pk_s and pk_j are the public keys of the sender and moderator, respectively. If J is indeed constructed in this manner, $J = g^u = g^{\alpha \cdot sk_j}$, then knowledge of u cannot be proved by anyone who does not know the moderator's secret key sk_j. This means a moderator that receives a valid signature and well-formed J will be convinced that the signature comes from a sender with knowledge of $t = sk_s$.

On the other hand, anyone can create a valid signature of $\mathsf{SPoK}^{\mathcal{R}_{\mathsf{SDVS}}}$ by using a malformed J set as a random group element, $J = g^\gamma$ for $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_p$, proving knowledge of $u = \gamma$, and sending $E_J = g^\alpha$ for independent $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_p$. Importantly, only the moderator has the ability to distinguish between well-formed and malformed J, by using the secret key sk_j to check whether (pk_j, E_J, J) forms a valid Diffie-Hellman triple ($J \overset{?}{=} E_J^{sk_j}$). This means that anyone can create a forged signature that is indistinguishable from a valid sender signature to everyone but the moderator.

Following the intuition from the previous section, to achieve accountability, the sender must prove to the receiver that the strong designated verifier signature for the moderator is well-formed. This corresponds to proving that J is well-formed, i.e., (pk_j, E_J, J) form a Diffie-Hellman triple. Putting it together, our final AMF construction is the signature of knowledge derived from the following relation:

$$\mathcal{R} = \Big\{ \big((t,\, u,\, v, w),\, (g,\, pk_s,\, pk_j,\, J,\, R,\, E_J)\big) : \big(pk_s = g^t \vee J = g^u\big) \,\wedge\, \big((J = (pk_j)^v \wedge E_J = g^v) \vee R = g^w\big) \Big\}.$$

An honest sender constructs $J = (pk_j)^\alpha$ and $R = (pk_r)^\beta$ for $(\alpha, \beta) \leftarrow\!\!\$\, (\mathbb{Z}_p)^2$, and sends ephemeral values $(E_J = g^\alpha, E_R = g^\beta)$ along with the $\mathsf{SPoK}^{\mathcal{R}}$ signature, where pk_r is the public key of the receiver. The first conjunction clause represents the strong designated verifier signature to the moderator and the second conjunction clause represents the strong designated proof to the receiver that the first clause is constructed properly. Forgeries for universal deniability are created with malformed J and R, forgeries for receiver compromise deniability with malformed J, and forgeries for judge compromise deniability do not use any malformed elements. A complete summary of how different signatures and forgeries are proved is given in Figure 4 and our full construction is detailed in pseudocode in Figure 5.

5 Security Analysis

We now explore the security of our deniable AMF scheme, arguing it achieves the accountability and deniability properties detailed in Section 3. We treat each set of properties in turn.

5.1 Accountability

As we discussed in the last section, the accountability properties intuitively follow from the underlying signature of knowledge's soundness properties: demonstrating forgeries that fool the recipient (unforgeability or sender binding) or the judge (receiver binding) implies the ability to generate a proof without a witness. However, it is not clear how to modularly define a suitably strong knowledge soundness property of the signature of knowledge underlying our construction. Our analyses therefore take a different tack, reducing to the soundness properties of the underlying Sigma protocol.

We discuss receiver binding, which shares the same high level strategy as sender binding. Our strategy is to show a winning adversary $\mathcal{A}$ breaks the one-wayness of the witness-statement relation $\mathcal{R}$, which we can use to build a discrete log adversary $\mathcal{B}$ extracting secret keys from the witness. The approach of the proof uses some techniques related to the proof of existential unforgeability under chosen message attack (EUF-CMA) for Fiat-Shamir-derived signatures (c.f., [17]), but the need of $\mathcal{B}$ to simulate $\mathcal{A}$'s oracle queries requires a more nuanced analysis. In fact performing this simulation leads us to make an additional knowledge-of-exponent assumption (KEA) assumption [10] about $\mathbb{G}$. We detail the needed KEA assumption in Appendix C. The full theorem is given below.

Theorem 1. *Let* AMF *be the asymmetric message franking scheme using signature of knowledge* SPoK *defined in Figure 5, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function* $\mathcal{H}$. *If* $\mathcal{H}$ *is modeled as a random oracle, for any* r-BIND *adversary* $\mathcal{A}$ *making at most* Q_{Frank} *franking oracle queries,* Q_{Judge} *judge oracle queries, and* Q_{ro} *random oracle queries, we give adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{ro} + 1)}{p^4} + (Q_{\mathsf{Judge}} + 1) \cdot \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}})$$
$$+ \frac{Q_{ro} + 1}{p} + \sqrt{2(Q_{ro} + 1) \cdot \mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of $\mathbb{G}$ and if $\mathcal{A}$ runs in time T and KEA extractor $\mathcal{E}_{\mathcal{C}}$ runs in time $t_{\mathcal{E}}$, then $\mathcal{B}$ runs in time $T' \approx 2T + 2(Q_{\mathsf{Judge}} + 1) \cdot t_{\mathcal{E}}$ and $\mathcal{C}$ runs in time $T' \approx T$.

We use $\approx$ above to hide small constants. We give a proof sketch here. The theorem statements and proofs for sender binding and unforgeability are similar. We defer the full proof details for all three accountability properties to Appendix E.

Proof sketch: Our proof proceeds via a sequence of games. The first set of game hops show how the game can be modified to answer $\mathcal{A}$'s franking queries without using the sender's secret key sk_s. Similarly to proving non-interactive zero knowledge for Fiat-Shamir-derived proofs [17, Theorem 20.3], this is done by programming the random oracle $\mathcal{H}$ to be consistent with the commitments used in the underlying Sigma protocol. This programming fails if a (randomly chosen) commitment collides with a value previously used as input to the random oracle. This happens with low probability as commitments are four uniformly chosen group elements. The birthday-bound term accounts for the probability of such a commitment collision.

The second set of game hops handles simulating the judge oracle without the judge's secret key. To do so we argue that one can simulate the queries using KEA extractors and, if that fails, we can build an adversary $\mathcal{C}$ that violates the KEA. In fact this step uses a hybrid argument which gradually replaces each oracle call with an extractor-utilizing simulation of the check. This accounts for the second term of the theorem's advantage bound.

Finally we are in a game now in which the only use of the judge and sender secret keys is to define the public keys. We use a rewinding lemma [17, Lemma 19.2]. If $\mathcal{A}$ succeeds at forging in one execution against a particular message, we can rerun $\mathcal{A}$ ("rewind" it) with a different random oracle output for that message. The rewinding lemma lower bounds the probability that $\mathcal{A}$ succeeds twice in a row by the probability that it succeeds once. In turn, if one can forge twice with different hash outputs, this allows extracting a witness from the Fiat-Shamir proof of knowledge. The last step involves a case analysis over the relation $\mathcal{R}$ to show that extracting a witness implies learning sk_s or sk_j, which we use to build our desired discrete log adversary $\mathcal{B}$. A subtlety in this final step is that extracting a witness implies learning $u = sk_j \cdot \alpha$, but not sk_j directly. We use a KEA extractor again to extract α, and thus complete the proof. This accounts for the final two terms of the advantage relation.

Replacing KEA with Gap-CDH. The KEA [10] is a somewhat exotic assumption, and a natural question to ask is if we can prove our scheme secure without it. By extending our franking signature by two group elements and reducing to Gap-CDH instead of DL, we can dispense with KEA. The assumption is used in two places in our proof while building DL adversary $\mathcal{B}$, (1) to answer judge oracle queries, and (2) to learn sk_j from the witness. In our alternate proof, the Gap-CDH oracle is used to answer judge oracle queries, and the extended franking signature directly proves knowledge of α and β so KEA is not needed to learn sk_j from the witness. This gives us the following theorem:

Theorem 2. *Let* AMF *be the asymmetric message franking scheme using signature of knowledge* SPoK *defined in Figure 5 over relation* $\mathcal{R}'$ *defined in Appendix H, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function* $\mathcal{H}$*. If* $\mathcal{H}$ *is modeled as a random oracle, for any* r-BIND *adversary* $\mathcal{A}$ *making at most* Q_{Frank} *franking oracle queries,* Q_{Judge} *judge oracle queries, and* Q_{ro} *random oracle queries, we give adversary* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{ro} + 1)}{p^4} + \frac{Q_{ro} + 1}{p} + \sqrt{2(Q_{ro}+1) \cdot \mathbf{Adv}^{\text{gapcdh}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of $\mathbb{G}$ and if $\mathcal{A}$ runs in time T, then $\mathcal{B}$ runs in time $T' \approx 2T$.

We provide the theorem statements for the other two accountability properties, as well as assumption definitions and proof details in Appendix H.

5.2 Deniability

Intuitively, the deniability properties fall out of the non-interactive zero knowledge property of the signature proofs of knowledge. Our signature proof of knowledge is carefully designed so that a variety of different witnesses can satisfy the statement relation $\mathcal{R}$ (as laid out in Figure 4). This allows forgers to create signatures that can only be caught by checking well-formedness of the statement using secret keys.

In more detail, the deniability proofs all follow the same outline. First notice that there are two high level differences between the frank algorithm and the forge algorithms: (1) the witnesses used to prove the statement are different, and (2) how the elements of the statement are formed is different. Different witnesses are handled by using the zero-knowledge property of the signature proof to switch between witnesses by hopping to a simulated proof and back. In fact, for judge compromise deniability, witness indistinguishability [37] is all that is needed since elements of the statement are well-formed and identical in Frank and

JForge. Extra care needs to be taken for Forge and RForge, since some elements of the statement are malformed. Well-formed means, for example, that J is constructed as $J \leftarrow (pk_j)^\alpha$ forming a Diffie-Hellman triple, $(pk_j = g^{sk_j}, E_J = g^\alpha, J = g^{\alpha \cdot sk_j})$. While malformed means $J \leftarrow g^\gamma$ is constructed as a random group element. In RForge, J is malformed, while in Forge both J and R are malformed. This leads to an additional DDH term to bound the advantage of an adversary in distinguishing between each well-formed and malformed statement elements.

The theorem statement for universal deniability is given below. The first term of the advantage comes from hopping between two witnesses through a simulator. The second term of the advantage comes from a decisional Diffie-Hellman hop for each of the two malformed elements of Forge.

Theorem 3. *Let* AMF *be the asymmetric message franking scheme defined in Figure 5 using signature of knowledge* SPoK *defined in Appendix D.1. For all simulators $\mathcal{S}$ for* SPoK*, for any* UnivDen *adversary $\mathcal{A}$, we give adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{univ-den}}_{\text{AMF}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\text{ddh}}_{\mathbb{G},g}(\mathcal{C}) \ .$$

where if $\mathcal{A}$ runs in time T and makes at most Q queries to the frank oracle, then $\mathcal{B}$ and $\mathcal{C}$ run in time $T' \approx T$ and $\mathcal{B}$ makes at most Q queries to its proof oracle.

The advantage terms for receiver compromise deniability and judge compromise deniability follow a similar structure. The full proofs for all deniability properties are deferred to Appendix F.

5.3 Measuring Concrete Security

Performing a concrete security analysis allows us to verify the efficiency of our reductions and inform parameter choices. The full details of our analysis are given in Appendix I. The reductions for accountability are not tight, due both to inheriting the quadratic loss seemingly fundamental to Schnorr-based Sigma protocols (c.f., [11]) and use of a KEA extractor to respond to each oracle query. The KEA poses a challenge for interpreting the concrete security analyses since the extractor is not concretely instantiated.

Strictly interpreted, our analysis suggests that we need a group $\mathbb{G}$ of more than twice the recommended size — i.e., > 512 bit elliptic curve groups to achieve about 128 bits of security. That said, we are not aware of any attacks against our schemes better than solving a discrete log, which is the same situation for standard Schnorr signatures and other uses of Fiat-Shamir. Therefore, in our implementations we also evaluate using 256-bit groups. This is standard in related settings (see [14, 20]). It remains a long-standing open question to understand if this heuristic is dangerous, i.e., if one can show an attack against Schnorr signatures (or similar) built from groups with conjectured 128-bit hardness that succeeds in time closer to 2^{64}.

6 Implementation and Evaluation

To evaluate our protocol, we implemented our signature proof of knowledge construction in Python 3 using the petlib [34] library which relies on OpenSSL for elliptic curve operations. Our implementation consists of a generic interface for implementing and composing Sigma protocols that may be of independent interest. For our AMF construction, we implemented the Schnorr protocol, Chaum-Pedersen protocol, and conjunction and disjunction protocols, as well as a Fiat-Shamir transform to create non-interactive proofs from the generic Sigma protocol interface.

We aim to evaluate the practicality of integrating our AMF scheme into existing messaging platforms. First, we are interested in the timing overhead in creating franking signatures as well as the space overhead in the signatures themselves. To this end, we present microbenchmarks to evaluate the overhead costs in our scheme. Second, we discuss what the deployment of an end-to-end moderation system incorporating asymmetric message franking would look like and present one such proof-of-concept for direct messaging on the Twitter platform. The AMF library as well as the deployment prototype are available open source at `https://github.com/julialen/asymmetric-message-franking`.

Benchmarks. We present timing and size benchmarks for our implementation of the signature of knowledge AMF construction. These experiments were conducted on an AWS `t3.small` EC2 virtual machine running

Algorithm	**Measured time (ms)**		**Group operations**		
	P-256	P-521	Mul	Add	Inv
Schnorr sig.	0.7 ± 0.12	2.4 ± 0.04	1	0	0
Frank	7.3 ± 0.95	28.3 ± 0.16	11	2	2
Verify	6.6 ± 0.90	29.1 ± 2.5	11	5	0
Judge	6.6 ± 0.16	32.0 ± 1.4	11	5	0
Forge	6.7 ± 0.11	32.0 ± 3.5	12	3	3
RForge	7.2 ± 0.12	34.2 ± 1.7	12	3	3
JForge	6.7 ± 0.11	31.6 ± 4.0	11	2	2

Fig. 6: Measured timing statistics and group operation accounting for the AMF algorithms from Figure 5 including a baseline comparison to a (undeniable) Schnorr signature. The measured times show the average and standard deviation over 1000 runs using a message size of 4KB instantiated over NIST elliptic curve groups P-256 and P-521. The group operations give the count of scalar multiplications (Mul), group additions (Add), and group inversions (Inv).

Ubuntu 18.04 on a 2.5 GHz Intel Scalable Processor using the NIST elliptic curve groups P-256 and P-521 and the hash function SHA-256.

The table in Figure 6 shows the measured time in milliseconds to run each of the algorithms from Figure 5. The measured times are the average over 1000 runs using a message size of 4KB. We compare to a baseline of a basic Schnorr signature of sk_s, which is undeniable. These numbers are as expected—our algorithms perform about ten times as many group operations as a Schnorr signature, and take roughly ten times as long. Though our scheme is slower, it is still fast enough to be used in practical settings where network latency dominates communication cost. We also provide the number of group operations (scalar multiplications, group additions, and group inversions) performed in each algorithm. These experiments were conducted using a fixed message size of 4KB, but we note that the only message-size dependent operation is a single hash for the Fiat-Shamir signature.

The size of an AMF signature is not message-dependent. Our algorithms all output nine group elements (i.e., elliptic curve points) and six scalars in $\mathbb{Z}_p$. In our implementation, AMF signatures are 489 bytes in size for elliptic curve group P-256 and 795 bytes in size for elliptic curve group P-521. In contrast, a Schnorr signature is one group element and one scalar and is 65 bytes in size for P-256 and 99 bytes in size for P-521.

Deployment. We prototype asymmetric message franking by building a proof-of-concept third-party moderation system which we can test by integrating it over already existing messaging platforms. Instantiating a third-party moderation system with asymmetric message franking involves three main services: (1) a judging service that receives and arbitrates abuse reports from users, (2) a publish-subscribe service to maintain an up-to-date community membership list amid new user enrollment and abusive user blocks, and (3) a public key infrastructure (PKI) to map platform identities to public keys. A user registers by enrolling with the membership service and delivering their public key to the PKI. To bind a platform identity to a key, the PKI should check some kind of proof-of-ownership of both the account and the secret key. This can be done using a challenge-response protocol, where the PKI delivers a random challenge to the user, who must sign the challenge with their private key and post the signed challenge on the platform. This will prevent rogue-key attacks that utilize malformed keys (q.v., [66]). Our proof-of-concept interfaces with Keybase [1] which provides the PKI service as described above.

The judging service can be performed by human moderators, automated tools, or some combination of the two. In our proof-of-concept, judging abuse reports is automated through the use of the Perspective conversation API [2] which uses machine learning to assign a "toxicity" score to a message; users are blocked based on a threshold of the score. We note that in a production deployment, use of automated moderation tools would need to be carefully tuned and likely also paired with human decision-making. Finally, we provide a client with a command-line interface to allow users to send, receive, and report direct messages on Twitter. The client automatically creates, appends, and parses franking signatures, as well as filters messages that are malformed or sent from a blocked user.

Lastly, we find that the cryptographic overhead of creating and verifying franking signatures is dwarfed by the overhead incurred by the rest of the infrastructure needed for moderation, e.g. PKI; sending a message over Twitter in our proof-of-concept takes $\approx$ 0.5 seconds. Much of this identity-binding infrastructure is needed for any moderation service — augmenting cryptographic verification using asymmetric message franking is not a significant overhead.

7 Discussion

Here we discuss some limitations of the use of AMF schemes.

Strong authentication. Our scheme does not ensure forgeries by the moderator can be detected by the receiver, and so the receiver cannot rely on AMF signatures alone to authenticate authorship if there is risk of the moderator being malicious. This is fundamental given our strong deniability notions (specifically, judge compromise deniability rules it out). One might weaken our deniability goals to achieve this, however. We explore such an alternate deniability target in Appendix B and informally present a modification to our scheme that achieves it.

Transcript consistency. In Facebook's current moderation solution [36], an abuse report contains context of surrounding messages sent by *both* users. In metadata-private moderation, it is difficult to ensure the moderator, sender, and receiver all have a consistent view of an interleaved message transcript because the moderator does not know what was sent or when. We might include sequence numbers and acknowledgment receipts to protect ordering. However, such techniques should be introduced with care so as not to obviate the system's deniability properties.

Moderator accountability via thresholding. Another issue is that our deniability goals may make holding moderators accountable for their actions more difficult. A fundamental property of asymmetric message franking is that the moderator cannot prove someone authored a message. At best they can prove a message was authored *either* by them *or* by the sender. As a result, the moderator cannot prove they had a valid reason for banning someone.

One potential mitigation for this would be to split the moderator's functionality across multiple parties. To do so, the key would need to be shared and a secure multi-party protocol used to test well-formedness of J in the franking signature, which can be done using techniques from verifiable secret sharing [62,64,71]. With threshold moderation, it takes the parties holding some t out of n moderator key shares to invoke Judge. This makes the moderator functionality more robust to accusations of unfair treatment, since t of them would need to act unfairly to falsely accuse someone. This also provides a natural defense against moderator key compromise, since to reconstruct the moderator key, t distinct parties would need to be breached.

Deniable channels. Finally, care must be taken when composing our AMF scheme with other cryptographic primitives, as those primitives may compromise or prevent deniability. In particular, one might worry about the deniability of the underlying authenticated channels, like TLS, through which AMF signed messages are sent. In general, if the sender uses one-way authentication for TLS, a TLS transcript is (cryptographically) universally deniable. In this authentication mode, the server is authenticated and the client generates and sends a randomly chosen ephemeral prekey to the server from which a session key is derived. Any party can create a session key with the platform server and use that session key to create a forged transcript. The IP address of the sender is learned by the platform server at the time of sending, but any transcript recording the IP address is unconvincing since it is not bound to the client-chosen randomness. Thus while there exist stronger notions of deniable channels [65,73], it seems TLS channels preserve universal deniability for arbitrary message platforms.

However, messaging platforms presumably perform their own user authentication *on top* of TLS, and this may be problematic for receiver compromise deniability in some scenarios. Due to the deniability properties of TLS described above, a transcript of messages served by the platform would be unconvincing. Instead, if the receiver were to reveal their platform credentials, someone can use those credentials to retrieve the messages directly from the platform. This interaction would convince someone that the messages were sent by the sender, given they trust the platform's underlying user authentication. One way to prevent this breach in deniability is if the platform does not serve archived messages. Platforms such as Signal, WhatsApp, and Facebook secret conversations do not back-up messages, and thus, already fit this model.

8 Related Work

Message franking schemes. Symmetric message franking has been studied in several works [29, 35, 36, 42, 46]. All of these schemes consider the symmetric setting where a centralized server holds a MAC key that authenticates the ciphertext, sender, recipient tuple. They do not transfer to settings where this communication metadata is not available. Moreover, while deniability is the motivating goal, the actual studied primitive is compactly committing authenticated encryption [42]. They do not formalize deniability.

Special purpose signature schemes. A variety of special purpose signature schemes have been proposed that do not work in our setting. In undeniable signatures [25, 27], verification requires interacting with the signer precisely to prevent them from denying messages they wrote. This is the same limitation as designated confirmer signatures [26], with the added problem that any compromise of the confirmer — who holds keys which can confirm but not issue signatures — removes any doubt about the authenticity of a signature.

Group signatures [9, 15, 19, 22] allow members of a specified group to sign messages indicating they are a part of the group without revealing the individual signer's identity. Group membership is determined by a group coordinator who has the additional capability of learning the individual signer from a signature. One can imagine a moderation protocol built from group signatures in which all users are part of the "global" group and the moderator is the group coordinator. Other than the efficiency issues of maintaining a global group with dynamic joins and revocations, these schemes do not achieve judge compromise deniability for the group coordinator's secret key. Ring signatures [67] similarly allow verification of group membership, but are not applicable to moderation, since they do not provide a way for the moderator to learn individual signer identities.

Designated verifier signatures [47, 48, 52, 56], in particular, strong designated verifier signatures [45, 69] provide nearly the functionality we need, but do not alone capture the relationship between the moderator and recipient parties. We informally describe an AMF construction (see Section 4.1 and Appendix G) consisting of a strong designated verifier proof [24, 33] to the receiver of the well-formedness of a strong designated verifier signature to the moderator. This approach can also be considered as a new variant of multi-designated verifier signatures [53] with a special relationship between designated verifiers, moderator and receiver.

Anonymous blacklisting systems. An anonymous blacklisting scheme [43] allows a user to produce a series of unlinkable tokens from a private key. To send a message, they provide a fresh token and prove that no tokens linked to their private key are on some blacklist. In this manner, moderators can blacklist sender tokens without learning the sender's identity — different from AMF where the sender's identity is learned by the moderator. However, the need for the sender to be able to identify and disavow tokens on the blacklist means that there is no deniability in the case an attacker compromises the sender (e.g., as in the DNC email breach). Our scheme, in contrast, protects the user's deniability even if their key is compromised.

Other work in deniability. Deniability has also been considered in other cryptographic contexts. Canetti et al. proposed *deniable encryption* [23], which allows the denial of contents of a ciphertext by giving a different opening of it. This doesn't deal with authorship or authentication and hence is not applicable. Borisov et al. [18] explored deniability as a feature for messaging systems. Deployed in OTR [18] and Signal [58], deniable messaging protocols ensure that messages can be authenticated by the receiver but not by third parties. On their own, they do not allow for moderation because the deniability is too strong: no one can authenticate the message, including the moderator. They can be combined with an AMF scheme to get an end-to-end encrypted and moderatable messaging scheme.

Automated moderation systems. A variety of works have explored ad-hoc moderation [40] and automated moderation systems [60]. We do not attempt to provide an exhaustive list here. One of the more notable projects is Google Jigsaw's Perspective API [60], which aims to build automated moderation tools to combat toxicity. While these works are promising, they cannot be used effectively if messages cannot be properly attributed to users.

9 Conclusion

In this paper, we investigated moderation for metadata-private messaging systems like Signal. Because user identities are hidden from the platform, existing moderation tools (including symmetric message franking) cannot be used. Other seeming solutions break deniability. Similar issues prevent third-party moderation, in which the messaging platform and moderator are decoupled.

We showed that the main technical challenge is cryptographic: how to balance the need for accountability in abuse reporting with the desire for deniability. We resolved this tension by introducing a new cryptographic primitive called asymmetric message franking (AMF), and showed how to construct one efficient enough for practice.

References

1. Keybase (2014), `https://keybase.io/docs/server_security`
2. Perspective API (2017), `https://www.perspectiveapi.com/`
3. Mastodon social network (2018), `https://joinmastodon.org/`
4. Matrix: an open network for secure, decentralized communication (2018), `https://matrix.org/`
5. Sealed sender represents 80% of signal traffic (2019), `https://twitter.com/signalapp/status/1075918894521495552`
6. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: CRYPTO (2010)
7. Angel, S., Setty, S.T.: Unobservable communication over fully untrusted infrastructure. In: OSDI (2016)
8. Bellare, M., Goldreich, O., Mityagin, A.: The power of verification queries in message authentication and authenticated encryption. IACR Cryptology ePrint Archive (2004)
9. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: EUROCRYPT (2003)
10. Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In: CRYPTO (2004)
11. Bellare, M., Poettering, B., Stebila, D.: From identification to signatures, tightly: A framework and generic transforms. In: ASIACRYPT (2016)
12. Bellare, M., Ristenpart, T.: Simulation without the artificial abort: Simplified proof and improved concrete security for waters' IBE scheme. In: EUROCRYPT (2009)
13. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: EUROCRYPT (2006)
14. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering (2012)
15. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO (2004)
16. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: ASIACRYPT (2001)
17. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography (2017), version 0.4
18. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use PGP. In: ACM WPES (2004)
19. Boyen, X., Waters, B.: Compact group signatures without random oracles. In: EUROCRYPT (2006)
20. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE S&P (2018)
21. Camenisch, J.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (1998)
22. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: CRYPTO (1997)
23. Canetti, R., Dwork, C., Naor, M., Ostrovsky, R.: Deniable encryption. In: CRYPTO (1997)
24. Chaidos, P., Couteau, G.: Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In: EUROCRYPT (2018)
25. Chaum, D.: Zero-knowledge undeniable signatures. In: EUROCRYPT (1990)
26. Chaum, D.: Designated confirmer signatures. In: EUROCRYPT (1994)
27. Chaum, D., Antwerpen, H.V.: Undeniable signatures. In: CRYPTO (1989)
28. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO (1992)
29. 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)
30. Corrigan-Gibbs, H., Boneh, D., Mazieres, D.: Riposte: An anonymous messaging system handling millions of users. In: IEEE S& P (2015)
31. Corrigan-Gibbs, H., Ford, B.: Dissent: accountable anonymous group messaging. In: ACM CCS (2010)
32. Damgård, I.: Towards practical public key systems secure against chosen ciphertext attacks. In: CRYPTO (1991)
33. Damgård, I., Fazio, N., Nicolosi, A.: Non-interactive zero-knowledge from homomorphic encryption. In: TCC (2006)

34. Danezis, G.: Petlib library (2018), https://github.com/gdanezis/petlib
35. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: CRYPTO (2018)
36. Facebook: Messenger Secret Conversations technical whitepaper (2017), https://fbnewsroomus.files.wordpress.com/2016/07/messenger-secret-conversations-technical-whitepaper.pdf
37. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC (1990)
38. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO (1986)
39. Galindo, D.: The exact security of pairing based encryption and signature schemes. In: Based on a talk at Workshop on Provable Security, INRIA, Paris (2004)
40. Geiger, R.S.: Bot-based collective blocklists in twitter: the counterpublic moderation of harassment in a networked public space. Information, Communication & Society (2016)
41. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. J. ACM (1991)
42. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: CRYPTO (2017)
43. Henry, R., Goldberg, I.: Formalizing anonymous blacklisting systems. In: IEEE S&P (2011)
44. van den Hooff, J., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: scalable private messaging resistant to traffic analysis. In: SOSP (2015)
45. Huang, Q., Yang, G., Wong, D.S., Susilo, W.: Efficient strong designated verifier signature schemes without random oracle or with non-delegatability. Int. J. Inf. Sec. (2011)
46. Huguenin-Dumittan, L., Leontiadis, I.: A message franking channel. Cryptology ePrint Archive, Report 2018/920 (2018)
47. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: EUROCRYPT (1996)
48. Kudla, C., Paterson, K.G.: Non-interactive designated verifier proofs and undeniable signatures. In: Cryptography and Coding (2005)
49. Kwon, A., Corrigan-Gibbs, H., Devadas, S., Ford, B.: Atom: Horizontally scaling strong anonymity. In: SOSP (2017)
50. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle. PoPETs (2016)
51. Kwon, A., Lu, D., Devadas, S.: Xrd: Scalable messaging system with cryptographic privacy. arXiv preprint arXiv:1901.04368 (2019)
52. Laguillaumie, F., Vergnaud, D.: Designated verifier signatures: Anonymity and efficient construction from any bilinear map. In: SCN (2004)
53. Laguillaumie, F., Vergnaud, D.: Multi-designated verifiers signatures. In: ICICS (2004)
54. Lazar, D., Gilad, Y., Zeldovich, N.: Karaoke: Distributed private messaging immune to passive traffic analysis. In: OSDI (2018)
55. Lazar, D., Zeldovich, N.: Alpenhorn: Bootstrapping secure communication without leaking metadata. In: OSDI (2016)
56. Lipmaa, H., Wang, G., Bao, F.: Designated verifier signature schemes: Attacks, new security notions and a new construction. In: ICALP (2005)
57. Lund, J.: Technology preview: sealed sender for Signal (2018), https://signal.org/blog/sealed-sender/
58. Marlinspike, M.: Simplifying OTR deniability. (2013), https://signal.org/blog/simplifying-otr-deniability/
59. Masnick, M.: The Clinton campaign should stop denying that the Wikileaks emails are valid; they are and they're real (2016), https://www.techdirt.com/articles/20161024/22533835878/clinton-campaign-should-stop-denying-that-wikileaks-emails-are-valid-they-are-theyre-real.shtml
60. Mullin, B.: The New York Times is teaming up with Alphabet's Jigsaw to expand its comments (2017), https://www.poynter.org/news/new-york-times-teaming-alphabets-jigsaw-expand-its-comments
61. Nossiter, A., Sanger, D.E., Perlroth, N.: Hackers Came, but the French Were Prepared (2017), https://www.nytimes.com/2017/05/09/world/europe/hackers-came-but-the-french-were-prepared.html
62. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO (1991)
63. Piotrowska, A.M., Hayes, J., Elahi, T., Meiser, S., Danezis, G.: The loopix anonymity system. In: USENIX Security (2017)
64. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: STOC (1989)
65. Raimondo, M.D., Gennaro, R., Krawczyk, H.: Deniable authentication and key exchange. In: CCS (2006)
66. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: EUROCRYPT (2007)

67. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT (2001)
68. Roose, K.: As Elites Switch to Texting, Watchdogs Fear Loss of Transparency (2017), https://www.nytimes.com/2017/07/06/business/as-elites-switch-to-texting-watchdogs-fear-loss-of-transparency.html
69. Saeednia, S., Kremer, S., Markowitch, O.: An efficient strong designated verifier signature scheme. In: ICISC (2003)
70. Schnorr, C.: Efficient identification and signatures for smart cards. In: CRYPTO (1989)
71. Stadler, M.: Publicly verifiable secret sharing. In: EUROCRYPT (1996)
72. Tyagi, N., Gilad, Y., Leung, D., Zaharia, M., Zeldovich, N.: Stadium: A distributed metadata-private messaging system. In: SOSP (2017)
73. Unger, N., Goldberg, I.: Deniable key exchanges for secure messaging. In: CCS (2015)

A Unforgeability from Receiver and Sender Binding

In this appendix, we present the unforgeability security notion, UNF for AMFs and show that it is implied by receiver binding and sender binding.

Unforgeability is specified formally in game UNF on the left-hand side of Figure 2. The adversary has access to a franking oracle for the (honest) sender to which it can query chosen receiver public key and message pairs. We also give the adversary access to verify and judge oracles to query chosen sender public key and message pairs. The adversary's goal is to generate a signature accepted by Verify for the target receiver key, and we disallow trivial wins (submitting a signature for the target receiver key and a message queried to the franking oracle). For an adversary $\mathcal{A}$ and message franking scheme AMF, we define the UNF advantage of $\mathcal{A}$ against AMF as

$$\mathbf{Adv}^{\mathrm{unf}}_{\mathrm{AMF}}(\mathcal{A}) = \Pr\left[\,\mathrm{UNF}^{\mathcal{A}}_{\mathrm{AMF}} \Rightarrow 1\,\right] \;.$$

Next, we prove that unforgeability is implied by receiver binding and sender binding. For intuition on how the proof will proceed, consider an adversary that creates a forgery that wins the unforgeability game, i.e., the forgery output passes receiver verification. Either this forgery passes judge verification or it does not. If it does pass judge verification, the adversary wins receiver binding; receivers can create forgeries from a sender that will be accepted by the judge. If it does not pass judge verification, the adversary wins sender binding; senders can create forgeries that are accepted by the receiver but rejected by the judge. Our theorem statement is as follows:

Theorem 4. *For any asymmetric message franking scheme* AMF *and any* UNF *adversary $\mathcal{A}$, we give adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathrm{unf}}_{\mathrm{AMF}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{r\text{-}bind}}_{\mathrm{AMF}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{s\text{-}bind}}_{\mathrm{AMF}}(\mathcal{C})$$

where if $\mathcal{A}$ runs in time T and makes at most Q_{Frank}, Q_{Verify}, and Q_{Judge} frank, verify, and judge oracle queries respectively, then $\mathcal{B}$ and $\mathcal{C}$ run in time $T' \approx T$, $\mathcal{B}$ makes at most Q_{Frank} frank and Q_{Verify} verify oracle queries, and $\mathcal{C}$ makes at most Q_{Verify} verify and Q_{Judge} judge oracle queries.

Proof. Define r-BIND adversary $\mathcal{B}$ that samples a random receiver key pair and runs $\mathcal{A}$ simulating $\mathcal{A}$'s frank and judge oracles by forwarding queries to its own r-BIND frank and judge oracles, and simulates $\mathcal{A}$'s verify oracle by using the sampled receiver secret key, and finally forwards $\mathcal{A}$'s output as its own. Adversary $\mathcal{B}$'s simulation of $\mathcal{A}$, given in pseudocode in Figure 7, matches exactly with that of the unforgeability game UNF.

Similarly define s-BIND adversary $\mathcal{C}$ that samples a random sender key pair and runs $\mathcal{A}$ simulating $\mathcal{A}$'s verify and judge oracles by forwarding queries to its own s-BIND verify and judge oracles, and simulates $\mathcal{A}$'s frank oracle by using the sampled sender secret key, and finally forwards $\mathcal{A}$'s output as its own. Adversary $\mathcal{C}$'s simulation of $\mathcal{A}$, given in Figure 7, also matches that of the unforgeability game UNF.

Call the event that the message, signature pair output by $\mathcal{A}$ passes the receiver verification algorithm, X_R, and the event that it passes the judge verification algorithm, X_J. Adversary $\mathcal{B}$ wins when judge verification passes:

$$\Pr\left[\,\text{r-BIND}^{\mathcal{B}}_{\mathrm{AMF}} \Rightarrow 1\,\right] = \Pr\left[\,X_J\,\right] \;,$$

and adversary $\mathcal{C}$ wins when receiver verification passes, but judge verification fails:

$$\Pr\left[\,\text{s-BIND}^{\mathcal{C}}_{\mathrm{AMF}} \Rightarrow 1\,\right] = \Pr\left[\,X_R \wedge \neg X_J\,\right] \;,$$

We know from the win condition of UNF that

$$\Pr\left[\,\mathrm{UNF}^{\mathcal{A}}_{\mathrm{AMF}} \Rightarrow 1\,\right] = \Pr\left[\,X_R\,\right] \;.$$

Thus, we have

$$\begin{aligned}
\Pr\left[\,\mathrm{UNF}^{\mathcal{A}}_{\mathrm{AMF}} \Rightarrow 1\,\right] &= \Pr\left[\,X_R\,\right] \\
&= \Pr\left[\,X_R \wedge X_J\,\right] + \Pr\left[\,X_R \wedge \neg X_J\,\right] \\
&\leq \Pr\left[\,X_J\,\right] + \Pr\left[\,X_R \wedge \neg X_J\,\right] \\
&= \Pr\left[\,\text{r-BIND}^{\mathcal{B}}_{\mathrm{AMF}} \Rightarrow 1\,\right] + \Pr\left[\,\text{s-BIND}^{\mathcal{C}}_{\mathrm{AMF}} \Rightarrow 1\,\right]
\end{aligned}$$

```
$\mathrm{UNF}^{\mathcal{A}}_{\mathrm{AMF}}$:
$(pk_s, sk_s) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_r, sk_r) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(pk_j, sk_j) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_r, pk_j)$
if $(pk_r, msg) \in Q$:
    return $0$
return $\mathsf{Verify}(pk_s, pk_r, sk_r, msg, \sigma)$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
return $\mathsf{Frank}(sk_s, pk'_r, pk_j, msg)$

$\mathcal{O}^{\mathsf{Verify}}(pk'_s, msg, \sigma)$:
return $\mathsf{Verify}(pk'_s, sk_r, pk_j, msg, \sigma)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
return $\mathsf{Judge}(pk'_s, pk_r, sk_j, msg, \sigma)$
```

```
$\mathcal{B}^{\mathcal{O}}(pk_s, pk_j)$:
$(pk_r, sk_r) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(msg, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sim}\mathcal{O}}(pk_s, pk_r, pk_j)$
return $(pk_r, msg, \sigma)$

$\mathsf{Sim}\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
return $\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$

$\mathsf{Sim}\mathcal{O}^{\mathsf{Verify}}(pk'_s, msg, \sigma)$:
return $\mathsf{Verify}(pk'_s, sk_r, pk_j, msg, \sigma)$

$\mathsf{Sim}\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
return $\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$
```

```
$\mathcal{C}^{\mathcal{O}}(pk_r, pk_j)$:
$(pk_s, sk_s) \leftarrow\!\!\$\ \mathsf{KeyGen}$
$(msg, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sim}\mathcal{O}}(pk_s, pk_r, pk_j)$
return $(pk_s, msg, \sigma)$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
return $\mathsf{Frank}(sk_s, pk'_r, pk_j, msg)$

$\mathsf{Sim}\mathcal{O}^{\mathsf{Verify}}(pk'_s, msg, \sigma)$:
return $\mathcal{O}^{\mathsf{Verify}}(pk'_s, msg, \sigma)$
$\mathsf{Sim}\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
return $\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$
```

Fig. 7: Unforgeability game for AMF schemes (left). Adversaries used in reduction of unforgeability to receiver binding (top right) and sender binding (bottom right).

B Space of Deniability Definitions

Our deniability definitions are implicitly parameterized by the secret keys available to the forger and the secret keys available to the distinguisher. In our work, we choose three such parameterizations, e.g., universal deniability gives the forger access to no secret keys and gives the distinguisher access to the sender secret key. In addition to our deniability definitions, we also have three accountability definitions, unforgeability, receiver binding, and sender binding. It is not easy to see that the deniability definitions chosen are compatible with the accountability definitions chosen. In fact, we have already pointed out that some of our accountability definitions are directly incompatible with some possible deniability definitions, e.g., a truly universal forging algorithm that requires no secret keys but can fool a distinguisher with all secret keys would violate unforgeability and receiver binding. The chosen deniability definitions must be carefully balanced with the desired accountability definitions. This section explores the space of alternate deniability definitions and provides some guidance on how to choose a set of consistent and complete definitions.

We lay out the space of possible deniability definitions in Figure 8. It is parameterized on the y-axis by the combination of keys given to the forger and on the x-axis by the combination of keys given to the distinguisher. We truncate the table by only considering definitions in which the forger does not have access to the sender's secret key sk_s. Creating indistinguishable "forgeries" with access to the sender's secret key is trivial by simply signing honestly. We populate the table with symbols to indicate which deniability definitions are incompatible with various accountability definitions: Unforgeability and receiver binding are defined as in our paper. Strong authentication is a stronger unforgeability notion ensuring that the receiver will not accept judge forgeries. Informally, it can be considered as an extension of the unforgeability definition in which the adversary knows the judge secret key. Disavowability captures the ability of the sender to disavow certain forgeries. In a more formal treatment, one might imagine there be separate disavowability notions for the ability to disavow different forgeries. Sender binding by itself is not incompatible with any of the deniability definitions, so it does not appear. However, we have seen how receiver binding and sender binding together imply unforgeability which *is* incompatible with some deniability definitions. Our visualization does not capture complexity created by interaction between accountability definitions.

				Distinguisher's keys							
	sk_s			0	1	0	1	0	1	0	1
		sk_r		0	0	1	1	0	0	1	1
			sk_j	0	0	0	0	1	1	1	1
Forger's keys	0	0	0		▼	● ■	● ■▼	◆	◆ ▼	●◆■	●◆■▼
	0	1	0		▼		▼	◆	◆ ▼	◆	◆ ▼
	0	0	1		▼	■	■▼		▼	■	■▼
	0	1	1		▼		▼		▼		▼

● : Incompatible with unforgeability ■ : Incompatible with strong authentication

◆ : Incompatible with receiver binding ▼ : Incompatible with disavowability

Fig. 8: Space of deniability definitions parameterized by the secret keys available to the forger and distinguisher along with their inherent incompatibilities with accountability definitions.

Choosing deniability targets. We next provide reasoning for how to choose deniability targets. Consider the top table in Figure 9. Here, we copy over the incompatibilities for the accountability definitions that we target in our AMF construction, unforgeability, receiver binding, and sender binding. The three circled symbols indicate our deniability targets: universal deniability, receiver compromise deniability, and judge compromise deniability. The uncircled symbols indicate which deniability definitions are implied by our deniability targets. We find that from only the three chosen deniability definitions, all other deniability definitions in the space are either (1) explicitly incompatible with our accountability definitions, or (2) implied by one of our chosen deniability definitions. Thus, our set of deniability and accountability definitions are complete and consistent.

Supporting strong authentication. An alternate trade-off between accountability and deniability that our AMF construction does not target is strong authentication. Strong authentication is incompatible with judge compromise deniability since it requires that a judge should not be able to create a forgery that is accepted by the receiver. To target strong authentication as an accountability goal, we introduce two new deniability targets to replace judge compromise deniability: *weak judge compromise deniability* and *judge-receiver compromise deniability*. Weak judge compromise deniability captures the idea that the judge's forging ability in the previous judge compromise deniability definition was too strong. Namely, that with only the judge's key, a judge forgery can fool a distinguisher with all three secret keys. In weak judge compromise deniability, a judge forgery is only able to fool a distinguisher with the sender secret key and judge secret key; the receiver secret key has distinguishing ability. In order to create a forgery that fools all three secret keys, we can introduce a new forging algorithm where the forger must have both the receiver secret key and the judge secret key, captured by judge-receiver compromise deniability. With these new deniability definitions, we have found a complete and consistent set of deniability and accountability definitions for strong authentication.

We can also consider how to build a scheme that meets this new set of strong authentication definitions. Without proof, we believe that extending our franking signature by adding a strong designated verifier signature to the receiver would meet this set of definitions. A judge would no longer be able to forge to a receiver, since they would not be able to create the strong designated verifier signature. However, a forger with both receiver and judge secret keys could forge the strong designated verifier signature with the receiver key and the remainder of the franking signature as before with the judge key.

C Proof Preliminaries

In this section, we provide formalization for the security properties of signatures of knowledge as well as formalization for security games that will be used in proof reductions.

Signatures of knowledge. A signature of knowledge scheme $\mathsf{SPoK}^{\mathcal{R}} = (\mathsf{prove}, \mathsf{verify})$ is a pair of algorithms associated with a witness-statement relation $\mathcal{R}$. A relation $\mathcal{R}$ is a set of witness-statement pairs; formally, $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the witness space and $\mathcal{Y}$ is the statement space. Further, define $\mathcal{M}$ and Σ as the message

Our AMF

Distinguisher's keys

	sk_s			0	1	0	1	0	1	0	1
		sk_r		0	0	1	1	0	0	1	1
			sk_j	0	0	0	0	1	1	1	1
Forger's keys	0	0	0	U	Ⓤ	●	●	◆	◆	●◆	●◆
	0	1	0	R	R	R	Ⓡ	◆	◆	◆	◆
	0	0	1	J	J	J	J	J	J	J	Ⓙ
	0	1	1	J	J	J	J	J	J	J	J

AMF with strong authentication

	sk_s			0	1	0	1	0	1	0	1
		sk_r		0	0	1	1	0	0	1	1
			sk_j	0	0	0	0	1	1	1	1
Forger's keys	0	0	0	U	Ⓤ	● ■	● ■	◆	◆	●◆■	●◆■
	0	1	0	R	R	R	Ⓡ	◆	◆	◆	◆
	0	0	1	J'	J'	■	■	J'	(J')	■	■
	0	1	1	JR	JR	JR	JR	JR	JR	JR	(JR)

Ⓤ : Universal deniability

Ⓡ : Receiver compromise deniability

Ⓙ : Judge compromise deniability

(J') : Judge-only compromise deniability

(JR) : Judge-receiver compromise deniability

● : Incompatible with unforgeability

◆ : Incompatible with receiver binding

■ : Incompatible with strong authentication

No circle: Implied by circled definition

Fig. 9: The targeted deniability and accountability definitions for the AMF scheme in our paper (top) and for an AMF scheme with a different deniability-accountability trade-off targeting strong authentication (bottom).

space and signature space respectively. The randomized proving algorithm, $\mathsf{prove} : \mathcal{M} \times \mathcal{X} \times \mathcal{Y} \to \Sigma$, outputs a signature proof of the statement for a message given a witness, $\pi \leftarrow\!\!\$\ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x)$. The proving algorithm returns $\perp$ if $(x, y) \notin \mathcal{R}$. The deterministic verification algorithm, $\mathsf{verify} : \mathcal{M} \times \Sigma \times \mathcal{Y} \to \{0, 1\}$, takes as input a message, signature proof, and statement, then returns a bit, $b \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$. We will say a signature proof π on message msg verifies with respect to statement y if $\mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y) = 1$. When the relation $\mathcal{R}$ is clear from context, we may simply write SPoK to mean $\mathsf{SPoK}^{\mathcal{R}}$.

Non-interactive zero knowledge. Let $\mathsf{SPoK}^{\mathcal{R}} = (\mathsf{prove}, \mathsf{verify})$ be a signature of knowledge scheme for witness-statement relation $\mathcal{R}$. Suppose that SPoK makes use of a hash function $\mathcal{H} : \{0, 1\}^* \to \{0, 1\}^{\mathsf{hlen}}$, which is modeled as a random oracle. Let $\mathcal{S}$ be a simulator for SPoK. A non-interactive zero knowledge (niZK) adversary $\mathcal{A}$ is given query access to a proof oracle and a random oracle. In the "real world", the proof oracle responds to queries using $\mathsf{SPoK}.\mathsf{prove}(msg, x, y)$ and the random oracle responds as a random function. In the "ideal world", the proof oracle and random oracle are simulated by $\mathcal{S} = (\mathcal{S}_{\mathsf{prove}}, \mathcal{S}_{\mathrm{ro}})$, and when simulating the proof oracle, $\mathcal{S}_{\mathsf{prove}}$ does not receive the witness x. The advantage of a niZK adversary $\mathcal{A}$ is defined as the ability to distinguish between the real and ideal world as defined by the security games given in Figure 10.

$$\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK}^{\mathcal{R}}, \mathcal{S}}(\mathcal{A}) = \left| \Pr\left[\, \mathrm{REAL}^{\mathcal{A}}_{\mathsf{SPoK}^{\mathcal{R}}} \Rightarrow 1 \,\right] - \Pr\left[\, \mathrm{IDEAL}^{\mathcal{A}}_{\mathcal{S}} \Rightarrow 1 \,\right] \right| \ .$$

Sigma protocols. Our signature of knowledge is formed using the Fiat-Shamir heuristic applied to a Sigma protocol. Here we define knowledge soundness and special honest verifier zero knowledge (HVZK) for a Sigma protocol, which we will make use of in proofs down the line.

Let $\mathcal{R}$ be a witness-statement relation. A *Sigma protocol* (Definition 19.3 [17]) for $\mathcal{R}$ is a pair (P, V). P is an interactive protocol algorithm called the *prover*, which takes as input a witness-statement pair $(x, y) \in \mathcal{R}$.

```
$\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SPoK}}$:
$b \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prove}}, \mathcal{O}_{\mathrm{ro}}}$
return $b$

$\mathcal{O}^{\mathsf{prove}}(msg, x, y)$:
return $\mathsf{SPoK.prove}(msg, x, y)$

$\mathcal{O}_{\mathrm{ro}}(msg)$:
$r \leftarrow\!\!\$ \{0,1\}^{\mathsf{hlen}}$
if $H[msg] \neq \perp$ :
    $r \leftarrow H[msg]$
return $r$
```

```
$\mathrm{IDEAL}^{\mathcal{A}}_{\mathcal{S}}$:
$b \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prove}}, \mathcal{O}_{\mathrm{ro}}}$
return $b$

$\mathcal{O}^{\mathsf{prove}}(msg, x, y)$:
if $(x, y) \notin \mathcal{R}$ then return $\perp$
$(st, \pi) \leftarrow\!\!\$ \mathcal{S}_{\mathsf{prove}}(st, msg, y)$
return $\pi$

$\mathcal{O}_{\mathrm{ro}}(msg)$:
$(st, r) \leftarrow\!\!\$ \mathcal{S}_{\mathrm{ro}}(st, msg)$
return $r$
```

Fig. 10: Non-interactive zero knowledge security game for signatures of knowledge.

V is an interactive protocol algorithm called the *verifier*, which takes as input a statement y, and which outputs bit b. P and V are structured so that an interaction between them always works as follows:

- P computes a message t, called the *commitment*, and sends t to V;
- Upon receiving P's commitment t, V chooses a *challenge* c at random from a challenge space $\mathcal{C}$, and sends c to P.
- Upon receiving V's challenge c, P computes a *response* z, and sends z to V.
- Upon receiving P's response z, V outputs bit b, which must be computed strictly as a function of the statement y and the *conversation* (t, c, z).

A Sigma protocol (P, V) for $\mathcal{R}$ provides *knowledge soundness* (Definition 19.4 [17]) if there exists an efficient extractor algorithm $\mathcal{E}$ which given two accepting conversations for statement y, (t, c, z) and (t, c', z') where $c \neq c'$, then $\mathcal{E}$ outputs witness x for y ($(x, y) \in \mathcal{R}$). Notationally, $x \leftarrow \mathcal{E}((t, c, z), (t, c', z'), y)$.

A Sigma protocol (P, V) for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ provides *special HVZK* (Definition 19.5 [17]) if there exists an efficient probabilistic simulator algorithm $\mathcal{S}$ which given statement and challenge pair $(y, c) \in \mathcal{Y} \times \mathcal{C}$, outputs commitment-response pair (t, z) such that (1) (t, c, z) is an accepting conversation for y, and (2) (t, c, z) has the same distribution as a conversation between P and V. Notationally, $(t, z) \leftarrow\!\!\$ \mathcal{S}(c, y)$.

Other security games. The discrete log (DL) security game, shown in Figure 11, tasks an adversary with calculating the discrete logarithm of a group element of a prime order cyclic group $\mathbb{G}$. The advantage of an adversary is defined as

$$\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{A}) = \Pr\left[\,\mathrm{DL}^{\mathcal{A}}_{\mathbb{G},g} \Rightarrow 1\,\right] \ .$$

The decisional Diffie-Hellman (DDH) security game, shown in Figure 11, tasks an adversary with distinguishing whether a triple of group elements are a random triple or are a Diffie-Hellman triple. The advantage of an adversary is defined as

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{DDH}^{\mathcal{A},0}_{\mathbb{G},g} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{DDH}^{\mathcal{A},1}_{\mathbb{G},g} \Rightarrow 1\,\right]\right| \ .$$

The knowledge of exponent assumption. The knowledge of exponent assumption [32] concerns triples of the form (g^a, g^b, g^{ab}) (i.e. a Diffie-Hellman (DH) triple) for a group $\mathbb{G}$ of order p (with p prime) and generator g. It says, roughly, that an adversary which on input g^a outputs a DH triple must "have knowledge" of the exponent b which can be extracted from its description. Formally, for an adversary $\mathcal{A}$ and extractor $\mathcal{E}_{\mathcal{A}}$ (which, crucially, is relative to $\mathcal{A}$), we define the *knowledge of exponent advantage* of $\mathcal{A}$ relative to $\mathcal{E}_{\mathcal{A}}$ as

$$\mathbf{Adv}^{\mathrm{kea}}_{\mathbb{G},g}(\mathcal{A}, \mathcal{E}_{\mathcal{A}}) = \Pr\left[\,\mathrm{KEA}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathbb{G},g} \Rightarrow 1\,\right] \ .$$

Pseudocode for the game KEA is in Figure 11. Our formalization follows Bellare et al. [10] except with randomized Turing machines instead of families of circuits. Our pseudocode does not depict the random tapes used by $\mathcal{A}$ or $\mathcal{E}_{\mathcal{A}}$, but the extractor is always given the random tape of the adversary.

Since the output of game KEA involves two adversaries (one of which depends on the other) interpreting it is subtle. The KEA game basically measures the probability $\mathcal{E}_{\mathcal{A}}$ *fails* to extract the exponent when $\mathcal{A}$

```
$\mathrm{DDH}^{\mathcal{A},b}_{\mathbb{G},g}$:
$(x, y, z) \leftarrow\!\!\$\ \mathbb{Z}_p^3$
if $b = 0$:
    $b' \leftarrow \mathcal{A}(g^x, g^y, g^z, g, p)$
else :
    $b' \leftarrow \mathcal{A}(g^x, g^y, g^{xy}, g, p)$
return $b'$
```

```
$\mathrm{DL}^{\mathcal{A}}_{\mathbb{G},g}$:
$x \leftarrow\!\!\$\ \mathbb{Z}_p$
$x' \leftarrow \mathcal{A}(g^x, g, p)$
return $x = x'$
```

```
$\mathrm{KEA}^{\mathcal{A},\mathcal{E}_\mathcal{A}}_{\mathbb{G},g}$:
$a \leftarrow\!\!\$\ \mathbb{Z}_p$
$(Y, C) \leftarrow\!\!\$\ \mathcal{A}(g^a, g)$
$c \leftarrow\!\!\$\ \mathcal{E}_\mathcal{A}(g^a, g)$
return $Y = C^a \wedge C \neq g^c$
```

Fig. 11: (Left) Decisional Diffie-Hellman (DDH) game. (Upper right) Discrete logarithm (DL) game. (Lower right) Knowledge of Exponent (KEA) game.

outputs a valid DH triple. The negation of the success condition is a disjunction of two events: either $\mathcal{A}$ does not output a DH triple or $\mathcal{E}_\mathcal{A}$ successfully extracts from $\mathcal{A}$.

D Fiat-Shamir Signature of Knowledge Security

In this section, we present the details of construction and proofs of security properties for our underlying signature of knowledge.

D.1 Construction

Our construction uses a Fiat-Shamir signature scheme [38] derived from basic Sigma protocols, Schnorr proofs of knowledge of discrete logarithm [70] and Chaum-Pedersen proofs of equality of discrete log [28], extended with conjunctions and disjunctions (logical ands and ors) [17]. The Sigma protocol construction for Schnorr can be referenced in [17, Figure 19.1] and for Chaum-Pedersen in [17, Figure 19.7]. The constructions for conjunctions and disjunctions of generic Sigma protocols can be referenced in [17, Section 19.7]. A signature scheme is derived from the composed Sigma protocol using the Fiat-Shamir heuristic [38] which can be referenced in [17, Section 19.6.1]. In short, the Fiat-Shamir heuristic replaces the random challenge of the verifier V with a hash value $c \leftarrow \mathcal{H}(msg, t)$ where t is the Sigma protocol commitment. We denote $c \leftarrow \mathcal{H}(msg, t)$ where the hash function hashes into the challenge space $\mathcal{C}$ of the Sigma protocol. In practice, this is achieved for a hash function that hashes to size hlen by resampling if the output falls outside the challenge space.

D.2 Security

We prove the following theorems regarding the knowledge soundness and zero knowledge properties of our signature of knowledge scheme.

Theorem 5. *The Sigma protocol (P, V) for witness-statement relation $\mathcal{R}$, where (P, V) is constructed as in Appendix D.1 and $\mathcal{R}$ is defined in Figure 5, is special HVZK and provides knowledge soundness.*

Proof. (P, V) is a conjunction of disjunctions of Schnorr proofs of knowledge of discrete log [70] and Chaum-Pedersen proofs of discrete log equality [28]. Since the underlying Schnorr Sigma protocol and Chaum-Pedersen Sigma protocol have special HVZK and provide knowledge soundness [17, Example 19.2, 19.3, Theorem 19.10] and the disjunction and conjunction constructions preserve special HVZK and knowledge soundness [17, Theorem 19.17, Theorem 19.8], (P, V) is special HVZK and provides knowledge soundness.

Theorem 6. *Let SPoK be the signature of knowledge defined in Figure 5 for the witness-statement relation $\mathcal{R}$ over a prime-order cyclic group of order p derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, we give a simulator $\mathcal{S}$, such that for any niZK adversary $\mathcal{A}$ that makes at most Q_{prove} proof queries and Q_{ro} random oracle queries*

$$\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{A}) \leq \frac{Q_{\mathit{prove}}(Q_{\mathit{prove}} + Q_{ro})}{p^4} .$$

Proof. By the above Theorem 5, the Sigma protocol for $\mathcal{R}$ is special HVZK. We apply Theorem 20.3 of [17] which shows non-interactive zero knowledge from arbitrary special HVZK Sigma protocol. The construction of simulator $\mathcal{S}$ is given in [17, Figure 20.2].

E Accountability Proofs

E.1 Receiver binding

Theorem 1. *Let* AMF *be the asymmetric message franking scheme using signature of knowledge* SPoK *defined in Figure 5, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any* r-BIND *adversary $\mathcal{A}$ making at most Q_{Frank} franking oracle queries, Q_{Judge} judge oracle queries, and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{ro} + 1)}{p^4} + (Q_{\mathsf{Judge}} + 1) \cdot \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}}) + \frac{Q_{ro} + 1}{p} + \sqrt{2(Q_{ro} + 1) \cdot \mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of $\mathbb{G}$ and if $\mathcal{A}$ runs in time T and KEA extractor $\mathcal{E}_{\mathcal{C}}$ runs in time $t_{\mathcal{E}}$, then $\mathcal{B}$ runs in time $T' \approx 2T + 2(Q_{\mathsf{Judge}} + 1) \cdot t_{\mathcal{E}}$ and $\mathcal{C}$ runs in time $T' \approx T$.

Proof. The approach of this proof follows closely the proof of existential unforgeability under chosen message attack (EUF-CMA) for Fiat-Shamir derived signatures [17, Theorem 19.15, Lemma 19.16]). The goal is to convert the adversary $\mathcal{A}$ that forges a judge-accepting frank into an adversary $\mathcal{B}$ that breaks the one-wayness of the witness-statement relation $\mathcal{R}$, i.e. attacks discrete log.

In order to do this, we have to simulate $\mathcal{A}$'s oracles with only partial knowledge of witnesses. Simulating the frank oracle takes the same approach as EUF-CMA and is done by using the zero knowledge property of the signature proof. Proofs are simulated by programming the random oracle representing $\mathcal{H}$ to be consistent with commitments. The programming will fail if the commitment chosen in the frank oracle has already been queried to the random oracle, but since the commitments are randomly chosen, this is unlikely to happen. The first term in the advantage arises from this collision probability.

Unlike in EUF-CMA for Fiat-Shamir signatures, we will also have to simulate judge oracle queries. In these queries, the judge secret key is used to check well-formedness of the J, E_J part of the statement, i.e., that they make up a Diffie-Hellman triple with pk_j. We use the extractor from the knowledge of exponents assumption to perform this check, making up the second term in the advantage.

The second part of the proof argues that the only way an adversary is able to successfully forge a frank is by reversing part of the witness-statement relation, i.e. learning sk_s or sk_j from pk_s, pk_j. Again, we exploit the fact that $\mathcal{H}$ is modeled as a random oracle and is used by the adversary to build a forgery. We can use the rewinding lemma [17, Lemma 19.2] and the knowledge soundness property of the proof to extract partial witnesses sk_s or sk_j breaking discrete log.

The proof will proceed as a series of game hops. Game pseudocode is shown in Figure 12 where Frank, Judge procedures are unrolled and $\mathsf{SPoK}^{\mathcal{R}}$.prove is further unrolled into the Fiat-Shamir heuristic derived from Sigma protocol (P, V). G_0 is defined to be the same as r-BIND with $\mathcal{H}$ modeled as a random oracle in which a table of random oracle input-output choices, T, is maintained. Before proceeding with the rest of the proof, recall that from the proof sketch above, we need the adversary $\mathcal{A}$ to query the random oracle with their winning forgery in order to apply the rewinding lemma. Without loss of generality, consider an $\mathcal{A}'$ that runs $\mathcal{A}$ and queries the random oracle on the forgery before returning, thus making at most $Q_{\text{ro}} + 1$ random oracle queries.

G_1 is defined the same as G_0, except in the franking oracle if the message-commitment pair (msg, t) for randomly generated commitment t of the signature proof of knowledge is already in T, i.e. has already been queried to the random oracle, G_1 sets a bad flag and returns false. Since the only difference in game behavior occurs after this event, by the fundamental lemma of game playing [13], the distinguishing power between G_0 and G_1 is bounded by the probability G_1 sets the bad flag. For each query to the franking oracle, the union bound implies the probability that the random oracle was previously queried at point (msg, t) is at most

$(Q_{\mathsf{Frank}} + Q_{\text{ro}} + 1)/p^4$ where p^4 represents the size of the commitment space of our signature of knowledge construction. Applying another union bound over the Q_{Frank} queries to the franking oracle, we have

$$\begin{aligned} |\Pr[\, G_0 \Rightarrow 1 \,] - \Pr[\, G_1 \Rightarrow 1 \,]| &\le \Pr[\, G_1 \text{ sets bad} \,] \\ &\le \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{\text{ro}} + 1)}{p^4} \,. \end{aligned}$$

G_2 is defined the same as G_1 except signature proofs in $\mathcal{O}^{\mathsf{Frank}}$ are now simulated by sampling a challenge c then running the simulator for the Sigma protocol associated with SPoK (from special HVZK) to get a valid proof (t, c, z). This is made to work by programming the random oracle to output c on (msg, t). Since in G_1, only fresh message-commitment pairs in $\mathcal{O}^{\mathsf{Frank}}$ are handled (a collision returns false), G_2 is able to successfully simulate and produces output statistically indistinguishable from the output of G_1.

$$|\Pr[\, G_1 \Rightarrow 1 \,] - \Pr[\, G_2 \Rightarrow 1 \,]| = 0$$

Next, we build a DL adversary $\mathcal{B}$ attacking discrete log (security game in Figure 11) by showing $\mathcal{A}$ inverts part of the witness-statement relation. Let us start by reviewing how $\mathcal{A}$ wins G_2. A winning adversary $\mathcal{A}$ returns a message-frank pair $msg, (\pi, J, R, E_J, E_R)$ for which (1) J is well-formed ($J = E_J^{sk_j}$), (2) π verifies, and (3) msg was not queried to $\mathcal{O}^{\mathsf{Frank}}$. Intuitively, by knowledge soundness, the only way an adversary $\mathcal{A}$ produces a (2) verifying proof π under constraints (1) and (3) is by knowing a witness to relation $\mathcal{R}$. Say the message-commitment pair (msg, t_i) of the winning proof π was the i^{th} query to the random oracle (out of $Q_{\text{ro}} + 1$). Since msg was not queried to $\mathcal{O}^{\mathsf{Frank}}$, we know that the winning proof challenge $c_i \leftarrow T[(msg, t_i)]$ was not programmed by the simulator. Thus, we can rewind back to when $\mathcal{A}$ queried (msg, t_i) and instead of responding with c_i, we can respond with a fresh $c' \neq c_i$. If $\mathcal{A}$ wins again on the same i^{th} message-commitment pair, we can use the knowledge soundness property of the Sigma protocol for SPoK to extract a valid witness for relation $\mathcal{R}$. It is necessary that $\mathcal{A}$ win on the same query in the rewind as in the first run, so as the knowledge soundness extractor can be used on two transcripts with the same commitment. Notice a valid witness must include either $t = sk_s$ or $u = sk_j \cdot \alpha$ as specified by the first disjunction clause of $\mathcal{R}$. We'll use this observation to build two discrete log adversaries $\mathcal{B}'$ and $\mathcal{B}''$ to extract sk_s and sk_j respectively.

$\mathcal{B}'$ will follow directly if $t = sk_s$ is part of the witness. If instead $u = sk_j \cdot \alpha$ is part of the witness, extra steps need to be taken for $\mathcal{B}''$ to extract sk_j since $sk_j \cdot \alpha$ hides sk_j for random $\alpha \leftarrow\!\$\, \mathbb{Z}_p$. There are two issues with constructing a DL $\mathcal{B}''$ extracting sk_j: (1) Without knowledge of sk_j, $\mathcal{B}''$ cannot simulate the judge oracle for $\mathcal{A}$, since $\mathcal{B}''$ will not be able to perform the well-formedness check of J. (2) In order to learn sk_j from $u = sk_j \cdot \alpha$, $\mathcal{B}''$ needs to learn α. We address both issues using the knowledge of exponent assumption (Figure 11). Briefly, the knowledge of exponent assumption says that if algorithm $\mathcal{C}$ returns Diffie-Hellman triple $E_J = g^{\alpha}, J = g^{sk_j \cdot \alpha}$ on input $pk_j = g^{sk_j}$ there exists an extractor $\mathcal{E}_{\mathcal{C}}$ that outputs α.

First we address the issue of learning sk_j from $u = sk_j \cdot \alpha$. Define KEA adversary $\mathcal{C}_0$ as a wrapper around $\mathcal{A}$ for the Diffie-Hellman triple pk_j, E_J, J for E_J, J output by $\mathcal{A}$. Game G_3 is the same as G_2 except the KEA extractor $\mathcal{E}_{\mathcal{C}_0}$ is used to extract t'. A bad flag is set if the well-formedness check of the extractor is not correct, indicating that the extractor failed. The probability of this occurring is bounded by the advantage of $\mathcal{C}_0$ against KEA. By the fundamental lemma of game-playing [13],

$$\begin{aligned} |\Pr[\, G_2 \Rightarrow 1 \,] - \Pr[\, G_3 \Rightarrow 1 \,]| &\le \Pr[\, G_3 \text{ sets bad} \,] \\ &\le \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}_0, \mathcal{E}_{\mathcal{C}_0}) \,. \end{aligned}$$

Next, we address the well-formedness checks performed by the judge oracle. We address this with a series of hybrid games G_4^i for $i \in [1, q]$ for $q = Q_{\mathsf{Judge}}$ where the first i queries to the judge oracle are checked using KEA extractors and the remaining are checked in the normal way with sk_j. Consider KEA adversary $\mathcal{C}_i$ which runs $\mathcal{A}$ up until $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{Judge}}$, then returns the queried E_J, J. The KEA $\mathcal{C}_i$ answers $\mathcal{A}$'s previous judge oracle queries using the extractors $\mathcal{E}_{\mathcal{C}_k}$ for $k < i$ from previous hybrids. Similar to before, a bad flag is set if the well-formedness check of the extractor in the i^{th} query is not correct; the probability of this occurring is bounded by the advantage of $\mathcal{C}_i$ against KEA. By the fundamental lemma of game-playing [13] if we denote G_3 as G_4^0,

$$\begin{aligned} \left|\Pr\left[\, G_4^{i-1} \Rightarrow 1 \,\right] - \Pr\left[\, G_4^i \Rightarrow 1 \,\right]\right| &\le \Pr\left[\, G_4^i \text{ sets bad} \,\right] \\ &\le \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}_i, \mathcal{E}_{\mathcal{C}_i}) \,. \end{aligned}$$

When composed over the q hybrids and including $\mathcal{C}_0$ from the previous hop, we have

$$|\Pr[\, G_2 \Rightarrow 1 \,] - \Pr[\, G_4^q \Rightarrow 1 \,]| \leq \sum_{i=0}^{q} \mathbf{Adv}^{\mathrm{kea}}_{\mathbb{G},g}(\mathcal{C}_i, \mathcal{E}_{\mathcal{C}_i}) \;.$$

Next, we calculate the probability of success of extracting a valid witness through rewinding, which will be used by both $\mathcal{B}'$ and $\mathcal{B}''$. Denote the probability $\mathcal{A}$ wins G_4^q with ϵ. Further define the probability $\mathcal{A}$ wins by outputting a winning proof corresponding to random oracle query i as ϵ_i. Then $\epsilon = \sum_{j=1}^{Q_{\mathrm{ro}}+1} \epsilon_j$. Similarly, define $\epsilon' = \sum_{j=1}^{Q_{\mathrm{ro}}+1} \epsilon'_j$ as the probability rewinding succeeds to extract a valid witness, i.e., the adversary outputs a winning signature for the same query on both the first run and rewind. Then, ϵ'_j is defined as the probability rewinding succeeds on query j, meaning that $\mathcal{A}$ wins on query j and then wins again on query i after rewinding. By a direct application of the rewinding lemma [17, Lemma 19.2],

$$\epsilon'_j \geq \epsilon_j^2 - \epsilon_j / p$$

where p represents the size of the challenge space. Then, we have

$$\epsilon' = \sum_{j=1}^{Q_{\mathrm{ro}}+1} \epsilon'_j \geq \sum_{j=1}^{Q_{\mathrm{ro}}+1} \epsilon_j^2 - \sum_{j=1}^{Q_{\mathrm{ro}}+1} \frac{\epsilon_j}{p} \geq \frac{\epsilon^2}{Q_{\mathrm{ro}}+1} - \frac{\epsilon}{p}$$

Rearranging gives that

$$\epsilon \leq \frac{Q_{\mathrm{ro}}+1}{p} + \sqrt{(Q_{\mathrm{ro}}+1)\epsilon'}$$

by the following steps, where we assume $\epsilon \geq (Q_{\mathrm{ro}}+1)/p$, as otherwise the above trivially holds,

$$\begin{aligned}
\left(\epsilon - \frac{Q_{\mathrm{ro}}+1}{p}\right)^2 &= \epsilon^2 - \frac{2\epsilon(Q_{\mathrm{ro}}+1)}{p} + \frac{(Q_{\mathrm{ro}}+1)^2}{p^2} \\
&\leq \epsilon^2 - \frac{2\epsilon(Q_{\mathrm{ro}}+1)}{p} + \frac{\epsilon(Q_{\mathrm{ro}}+1)}{p} && (\epsilon \geq \tfrac{Q_{\mathrm{ro}}+1}{p}) \\
&= \epsilon^2 - \frac{\epsilon(Q_{\mathrm{ro}}+1)}{p} \leq \epsilon'(Q_{\mathrm{ro}}+1)
\end{aligned}$$

Define $\mathcal{B}'$ to replace pk_s with DL challenge and return $t = sk_s$ if output from successful rewind. Define $\mathcal{B}''$ to replace pk_j with DL challenge and return sk_j if $u = \alpha \cdot sk_j$ is output from successful rewind by calculating $sk_j = u/\alpha'$ for α' extracted using KEA extractor $\mathcal{E}_{\mathcal{C}}$. This gives us,

$$\epsilon' \leq \mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B}') + \mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B}'') \;.$$

Since it is unknown which witness for the disjunction $\mathcal{A}$ will win with, define $\mathcal{B}$ which runs $\mathcal{B}'$ and $\mathcal{B}''$ each with probability $1/2$.

$$\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B}) = \frac{\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B}') + \mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B}'')}{2} \;.$$

We finish the proof with a technique to combine $q+1$ adversaries $\mathcal{C}_i$ into a single adversary $\mathcal{C}$ for $\mathrm{KEA}_{\mathbb{G},g}$. Define $\mathcal{C}$ as running each adversary $\mathcal{C}_i$ with probability $1/(q+1)$ and $\mathcal{E}_{\mathcal{C}}$ as running $\mathcal{E}_{\mathcal{C}_i}$ when $\mathcal{C}$ runs $\mathcal{C}_i$.

$$\begin{aligned}
\sum_{i=0}^{q} \mathbf{Adv}^{\mathrm{kea}}_{\mathbb{G},g}(\mathcal{C}_i, \mathcal{E}_{\mathcal{C}_i}) &= \sum_{i=0}^{q} \Pr\left[\mathrm{KEA}^{\mathcal{C},\mathcal{E}_{\mathcal{C}}}_{\mathbb{G},g} \Rightarrow 1 \;\middle|\; \mathcal{C} \text{ runs } \mathcal{C}'\right] \\
&= (q+1) \cdot \Pr\left[\mathrm{KEA}^{\mathcal{C},\mathcal{E}_{\mathcal{C}}}_{\mathbb{G},g} \Rightarrow 1\right] \\
&= (q+1) \cdot \mathbf{Adv}^{\mathrm{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}})
\end{aligned}$$

E.2 Sender binding

Theorem 7. *Let* AMF *be the asymmetric message franking scheme using signature of knowledge* SPoK *defined in Figure 5, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1*

```
$G_0$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$b_j \leftarrow (b_1 \wedge b_2)$
$b_Q \leftarrow (pk_r, msg) \notin Q$
$\mathbf{return}\ b_j \wedge b_Q$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
$(\alpha, \beta) \leftarrow\$ (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk'_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x \leftarrow (sk_s, \perp, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$t, aux \leftarrow P(x, y)$
$c \leftarrow \mathcal{O}_{\mathrm{ro}}(msg, t)$
$z \leftarrow P(t, c, x, y, aux)$
$\pi \leftarrow (t, z)$
$\mathbf{return}\ (\pi, J, R, E_J, E_R)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk'_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$\mathbf{return}\ b_1 \wedge b_2$

$\mathcal{O}_{\mathrm{ro}}(m)$:
$r \leftarrow\$ \{0,1\}^{\mathsf{hlen}}$
$\mathbf{if}\ T[m] \neq \perp :$
    $r \leftarrow T[m]$
$\mathbf{return}\ r$
```

```
$G_1$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$b_j \leftarrow (b_1 \wedge b_2)$
$b_Q \leftarrow (pk_r, msg) \notin Q$
$\mathbf{return}\ b_j \wedge b_Q$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
$(wE, \beta) \leftarrow\$ (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk'_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x \leftarrow (sk_s, \perp, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$t, aux \leftarrow P(x, y)$
$\mathbf{if}\ T[msg, t] \neq \perp :$
    $\mathsf{flag} \leftarrow \mathsf{bad};\ \mathbf{return}\ \mathsf{false}$
$c \leftarrow \mathcal{O}_{\mathrm{ro}}(msg, t)$
$z \leftarrow P(t, c, x, y, aux)$
$\pi \leftarrow (t, z)$
$\mathbf{return}\ (\pi, J, R, E_J, E_R)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk'_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$\mathbf{return}\ b_1 \wedge b_2$

$\mathcal{O}_{\mathrm{ro}}(m)$:
$r \leftarrow\$ \{0,1\}^{\mathsf{hlen}}$
$\mathbf{if}\ T[m] \neq \perp :$
    $r \leftarrow T[m]$
$\mathbf{return}\ r$
```

```
$G_2$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$b_j \leftarrow (b_1 \wedge b_2)$
$b_Q \leftarrow (pk_r, msg) \notin Q$
$\mathbf{return}\ b_j \wedge b_Q$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
$(\alpha, \beta) \leftarrow\$ (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk'_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$c \leftarrow\$ \mathbb{Z}_p$
$(t, z) \leftarrow \mathcal{S}_{(P,V)}(c, y)$
$\mathbf{if}\ T[msg, t] \neq \perp :$
    $\mathbf{return}\ \mathsf{false}$
$T[msg, t] \leftarrow c$
$\pi \leftarrow (t, z)$
$\mathbf{return}\ (\pi, J, R, E_J, E_R)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk'_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$\mathbf{return}\ b_1 \wedge b_2$

$\mathcal{O}_{\mathrm{ro}}(m)$:
$r \leftarrow\$ \{0,1\}^{\mathsf{hlen}}$
$\mathbf{if}\ T[m] \neq \perp :$
    $r \leftarrow T[m]$
$\mathbf{return}\ r$
```

```
$G_3$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$\alpha' \leftarrow \mathcal{E}_{\mathcal{C}_0}(pk_j, g)$
$b'_1 \leftarrow E_J = g^{\alpha'} \wedge J = (pk_j)^{\alpha'}$
$\mathbf{if}\ b_1 \neq b'_1 :$
    $\mathsf{flag} \leftarrow \mathsf{bad};\ b_1 \leftarrow b'_1$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$b_j \leftarrow (b_1 \wedge b_2)$
$b_Q \leftarrow (pk_r, msg) \notin Q$
$\mathbf{return}\ b_j \wedge b_Q$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
$(\alpha, \beta) \leftarrow\$ (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk'_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$c \leftarrow\$ \mathbb{Z}_p$
$(t, z) \leftarrow \mathcal{S}_{(P,V)}(c, y)$
$\mathbf{if}\ T[msg, t] \neq \perp :$
    $\mathbf{return}\ \mathsf{false}$
$T[msg, t] \leftarrow c$
$\pi \leftarrow (t, z)$
$\mathbf{return}\ (\pi, J, R, E_J, E_R)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk'_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$\mathbf{return}\ b_1 \wedge b_2$

$\mathcal{O}_{\mathrm{ro}}(m)$:
$r \leftarrow\$ \{0,1\}^{\mathsf{hlen}}$
$\mathbf{if}\ T[m] \neq \perp :$
    $r \leftarrow T[m]$
$\mathbf{return}\ r$
```

```
$G_4^i$:
$k \leftarrow 1$
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, msg, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_s, pk_j)$
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\alpha' \leftarrow \mathcal{E}_{\mathcal{C}_0}(pk_j, g)$
$b_1 \leftarrow E_J = g^{\alpha'} \wedge J = (pk_j)^{\alpha'}$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$b_j \leftarrow (b_1 \wedge b_2)$
$b_Q \leftarrow (pk_r, msg) \notin Q$
$\mathbf{return}\ b_j \wedge b_Q$

$\mathcal{O}^{\mathsf{Frank}}(pk'_r, msg)$:
$Q \leftarrow Q \cup \{(pk'_r, msg)\}$
$(\alpha, \beta) \leftarrow\$ (\mathbb{Z}_p)^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk'_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$c \leftarrow\$ \mathbb{Z}_p$
$(t, z) \leftarrow \mathcal{S}_{(P,V)}(c, y)$
$\mathbf{if}\ T[msg, t] \neq \perp :$
    $\mathbf{return}\ \mathsf{false}$
$T[msg, t] \leftarrow c$
$\pi \leftarrow (t, z)$
$\mathbf{return}\ (\pi, J, R, E_J, E_R)$

$\mathcal{O}^{\mathsf{Judge}}(pk'_s, msg, \sigma)$:
$(\pi, J, R, E_J, E_R) \leftarrow \sigma$
$y \leftarrow (g, pk'_s, pk_j, J, R, E_J)$
$b_1 \leftarrow J = E_J^{sk_j}$
$\mathbf{if}\ k \leq i\ :$
    $\alpha' \leftarrow \mathcal{E}_{\mathcal{C}_k}(pk_j, g)$
    $b'_1 \leftarrow E_J = g^{\alpha'} \wedge J = (pk_j)^{\alpha'}$
    $\mathbf{if}\ (b_1 \neq b'_1) \wedge (k = i) :$
        $\mathsf{flag} \leftarrow \mathsf{bad};\ b_1 \leftarrow b'_1$
$b_2 \leftarrow \mathsf{SPoK}^{\mathcal{R}}.\mathsf{verify}(msg, \pi, y)$
$k \leftarrow k + 1$
$\mathbf{return}\ b_1 \wedge b_2$

$\mathcal{O}_{\mathrm{ro}}(m)$:
$r \leftarrow\$ \{0,1\}^{\mathsf{hlen}}$
$\mathbf{if}\ T[m] \neq \perp :$
    $r \leftarrow T[m]$
$\mathbf{return}\ r$
```

Fig. 12: Games for the proof of r-BIND, Theorem 1.

using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any s-BIND *adversary $\mathcal{A}$ making at most Q_{Verify} verify oracle queries and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{s-bind}}_{\text{AMF}}(\mathcal{A}) \leq \ (Q_{\mathsf{Verify}} + 1) \cdot \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}}) + \frac{Q_{ro} + 1}{p} + \sqrt{(Q_{ro} + 1) \cdot \mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of $\mathbb{G}$ and if $\mathcal{A}$ runs in time T and KEA extractor $\mathcal{E}_{\mathcal{C}}$ runs in time $t_{\mathcal{E}}$, then $\mathcal{B}$ runs in time $T' \approx 2T + 2(Q_{\mathsf{Verify}} + 1) \cdot t_{\mathcal{E}}$ and $\mathcal{C}$ runs in time $T' \approx T$.

Proof sketch: The first set of game hops handles simulating the verify oracle for without knowing the verifier's secret key. Like before, we do so by simulating the queries using KEA extractors, gradually replacing each oracle call in a hybrid argument. This accounts for a Q_{Verify} multiple on the first term of the theorem's advantage bound.

Now in a game that only generates the public key of the receiver, we use a rewinding lemma [17, Lemma 19.2] to extract a witness from the winning proof of knowledge. We argue that extracting a witness implies recovering sk_r from pk_r, which we can use to construct a DL adversary. Consider the relation $\mathcal{R}$. Consider the second disjunction in cases:

$C1$: $v = \alpha$: This implies that J is well-formed ($J = g^{\alpha \cdot sk_j}$). However, since a winning query fails Judge this means the J must be malformed (the signature proof verifies since Verify succeeds). This rules out case 1 as a possibility.

$C2$: $w = \beta \cdot sk_r$: Since the signature output by the adversary passes Verify, we know that R is well-formed ($R = g^{\beta \cdot sk_r}$). To recover sk_r from w, we use a knowledge of exponents extractor to extract β. This leads to the added multiple on the KEA advantage over just simulating the adversary's verify oracle.

The discrete log adversary along with the rewinding lemma get the last two terms of the advantage.

E.3 Unforgeability

Corollary 1. *Let* AMF *be the asymmetric message franking scheme using signature of knowledge* SPoK *defined in Figure 5, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any* UNF *adversary $\mathcal{A}$ making at most Q_{Frank} franking oracle queries, Q_{Verify} verify oracle queries, Q_{Judge} judge oracle queries, and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\begin{aligned}\mathbf{Adv}^{\text{unf}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{ro} + 1)}{p^4} &+ 2(Q_{\mathsf{Verify}} + Q_{\mathsf{Judge}} + 1) \cdot \mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}}) \\ &+ \frac{2(Q_{ro} + 1)}{p} + \sqrt{4(Q_{ro} + 1) \cdot \mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})}\end{aligned}$$

where p is the order of $\mathbb{G}$ and if $\mathcal{A}$ runs in time T and KEA extractor $\mathcal{E}_{\mathcal{C}}$ runs in time $t_{\mathcal{E}}$, then $\mathcal{B}$ runs in time $T' \approx 2T + 2(Q_{\mathsf{Verify}} + Q_{\mathsf{Judge}} + 2) \cdot t_{\mathcal{E}}$ and $\mathcal{C}$ runs in time $T' \approx T$.

F Deniability Proofs

F.1 Universal Deniability

Theorem 3. *Let* AMF *be the asymmetric message franking scheme defined in Figure 5 using signature of knowledge* SPoK *defined in Appendix D.1. For all simulators $\mathcal{S}$ for* SPoK*, for any* UnivDen *adversary $\mathcal{A}$, we give adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{univ-den}}_{\text{AMF}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\text{ddh}}_{\mathbb{G},g}(\mathcal{C}) \ .$$

where if $\mathcal{A}$ runs in time T and makes at most Q queries to the frank oracle, then $\mathcal{B}$ and $\mathcal{C}$ run in time $T' \approx T$ and $\mathcal{B}$ makes at most Q queries to its proof oracle.

Proof. We bound the advantage of $\mathcal{A}$ in the UnivDen game by bounding the advantage of each of a series of game hops, depicted in Figure 13. We define G_0 equal to $\mathrm{UnivDen}^0_{\mathrm{AMF}}$ and G_4 equal to $\mathrm{UnivDen}^1_{\mathrm{AMF}}$. The pseudocode in Figure 13 for G_0 and G_4 unrolls the Frank and Forge algorithm in $\mathcal{O}^{\mathsf{Frank}}$ and denotes the random oracle explicitly in the superscript of the adversary. The difference between G_0 and G_4 (Frank and Forge) is in the construction of J, R. In G_0, J and R are well-formed, while in G_4 both are malformed. Well-formed means, for example, that J is constructed as $J \leftarrow (pk_j)^\alpha$ forming a Diffie-Hellman triple, $(pk_j = g^{sk_j}, E_J = g^\alpha, J = g^{\alpha \cdot sk_j})$. While malformed means $J \leftarrow g^\gamma$ is constructed as a random group element. At a high level, the proof proceeds by hopping between the well-formed SPoK statement to the malformed SPoK statement. However, to create valid signatures throughout the hops with unknown witnesses, we first invoke the non-interactive zero knowledge property of the signature of knowledge to simulate the signature without witness knowledge.

G_1 is the same as G_0 except the signature proof of knowledge in the franking oracle is simulated and the random oracle is simulated. We use non-interactive zero knowledge of the signature proof of knowledge to bound this hop. In detail, consider the adversary $\mathcal{B}'$ for niZK game in Figure 10 which simulates G_0 but generates the signature through the proof oracle $\mathcal{O}^{\mathsf{prove}}$. When in the real world, $\mathcal{B}'$ simulates exactly G_0, and in the ideal world, $\mathcal{B}'$ simulates exactly G_1.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}') &= \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{B}'}_{\mathsf{SPoK}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{IDEAL}^{\mathcal{B}'}_{\mathcal{S}} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_0 \Rightarrow 1\,\right] - \Pr\left[\,G_1 \Rightarrow 1\,\right]\right|\end{aligned}$$

Games G_2 and G_3 alter the construction of exponentiations J and R from G_1. Game G_2 replaces the well-formed $J \leftarrow (pk_j)^\alpha$ of Frank from G_1 with the malformed $J \leftarrow g^\gamma$ of Forge for random choice of $\gamma \leftarrow\!\$\, \mathbb{Z}_p$. We apply the decisional Diffie-Hellman (DDH) assumption to bound the advantage of this game hop. Construct adversary $\mathcal{C}'$ for the DDH game (Figure 11) that takes (g^x, g^y, g^z) and simulates G_1/G_2 by setting $E_J \leftarrow g^x; pk_j \leftarrow g^y; J \leftarrow g^z$. When the third element g^z is distributed as in $\mathrm{DDH}^{\mathcal{C}',0}_{\mathbb{G},g}$ (in which z is independent of x and y), adversary $\mathcal{C}'$ simulates G_2 exactly, i.e. g^z acts as a malformed J. When the third element g^z is distributed as in $\mathrm{DDH}^{\mathcal{C}',1}_{\mathbb{G},g}$ (in which $z = xy$), adversary $\mathcal{C}'$ simulates G_1 exactly, i.e. g^z acts a well-formed J.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}') &= \left|\Pr\left[\,\mathrm{DDH}^{\mathcal{C}',0}_{\mathbb{G},g} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{DDH}^{\mathcal{C}',1}_{\mathbb{G},g} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_2 \Rightarrow 1\,\right] - \Pr\left[\,G_1 \Rightarrow 1\,\right]\right|\end{aligned}$$

The game hop from G_2 to G_3 follows similarly. Game G_3 replaces the well-formed $R \leftarrow (pk_r)^\beta$ of Frank with the malformed $R \leftarrow g^\delta$ of Forge for random choice of $\delta \leftarrow\!\$\, \mathbb{Z}_p$. Construct DDH adversary $\mathcal{C}''$ that takes (g^x, g^y, g^z) and sets $E_R \leftarrow g^x; pk_r \leftarrow g^y; R \leftarrow g^z$.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}'') &= \left|\Pr\left[\,\mathrm{DDH}^{\mathcal{C}'',0}_{\mathbb{G},g} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{DDH}^{\mathcal{C}'',1}_{\mathbb{G},g} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_3 \Rightarrow 1\,\right] - \Pr\left[\,G_2 \Rightarrow 1\,\right]\right|\end{aligned}$$

In order to get to a runnable Forge that does not program the random oracle, the final game G_4 replaces the simulated proof with the signature proof of knowledge used in Forge. Again, we apply non-interactive zero knowledge of the signature proof of knowledge to bound this hop. Construct niZK adversary $\mathcal{B}''$ for niZK game which simulates G_4 but generates the signature through the proof oracle $\mathcal{O}^{\mathsf{prove}}$. When in the real world, $\mathcal{B}''$ simulates exactly G_4, and in the ideal world, $\mathcal{B}''$ simulates exactly G_3.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}'') &= \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{B}''}_{\mathsf{SPoK}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{IDEAL}^{\mathcal{B}''}_{\mathcal{S}} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_4 \Rightarrow 1\,\right] - \Pr\left[\,G_3 \Rightarrow 1\,\right]\right|\end{aligned}$$

We finish the proof with a common technique to combine adversaries $\mathcal{C}'$ and $\mathcal{C}''$ into a single adversary $\mathcal{C}$ for $\mathrm{DDH}^b_{\mathbb{G},g}$ and $\mathcal{B}', \mathcal{B}''$ into $\mathcal{B}$. Define $\mathcal{C}$ as flipping a random bit b, if $b = 0$, $\mathcal{C}$ runs $\mathcal{C}'$, otherwise $\mathcal{C}$ runs $\mathcal{C}''$. The advantage of $\mathcal{C}$ is then the average of advantages of $\mathcal{C}'$ and $\mathcal{C}''$,

$$2 \cdot \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}) \;=\; \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}') + \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}'')\;.$$

```
$G_0$:
$sk_s \xleftarrow{\$} \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_r \xleftarrow{\$} \mathbb{Z}_p;\ pk_r \leftarrow g^{sk_r}$
$sk_j \xleftarrow{\$} \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, pk_r, pk_j)$
$\mathbf{return}\ b'$

$\mathcal{O}^{\mathsf{Frank}}(msg)$:
$(\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (sk_s, \perp, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \xleftarrow{\$} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

```
$G_1$:
$sk_s \xleftarrow{\$} \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_r \xleftarrow{\$} \mathbb{Z}_p;\ pk_r \leftarrow g^{sk_r}$
$sk_j \xleftarrow{\$} \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{S}_{\mathrm{ro}}}(sk_s, pk_r, pk_j)$
$\mathbf{return}\ b'$

$\mathcal{O}^{\mathsf{Frank}}(msg)$:
$(\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\perp, \perp, \perp, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \xleftarrow{\$} \mathcal{S}_{\mathsf{prove}}(msg, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

```
$G_2$ $\boxed{G_3}$:
$sk_s \xleftarrow{\$} \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_r \xleftarrow{\$} \mathbb{Z}_p;\ pk_r \leftarrow g^{sk_r}$
$sk_j \xleftarrow{\$} \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{S}_{\mathrm{ro}}}(sk_s, pk_r, pk_j)$
$\mathbf{return}\ b'$

$\mathcal{O}^{\mathsf{Frank}}(msg)$:
$(\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$
$\gamma \xleftarrow{\$} \mathbb{Z}_p;\ J \leftarrow g^\gamma$
$R \leftarrow (pk_r)^\beta$ $\boxed{\delta \xleftarrow{\$} \mathbb{Z}_p;\ R \leftarrow g^\delta}$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\perp, \perp, \perp, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \xleftarrow{\$} \mathcal{S}_{\mathsf{prove}}(msg, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

```
$G_4$:
$sk_s \xleftarrow{\$} \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_r \xleftarrow{\$} \mathbb{Z}_p;\ pk_r \leftarrow g^{sk_r}$
$sk_j \xleftarrow{\$} \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, pk_r, pk_j)$
$\mathbf{return}\ b'$

$\mathcal{O}^{\mathsf{Frank}}(msg)$:
$(\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$
$\gamma \xleftarrow{\$} \mathbb{Z}_p;\ J \leftarrow g^\gamma$
$\delta \xleftarrow{\$} \mathbb{Z}_p;\ R \leftarrow g^\delta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\perp, \gamma, \perp, \delta)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \xleftarrow{\$} \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

Fig. 13: Games for the proof of universal deniability, Theorem 3.

Define $\mathcal{B}$ in the same manner,

$$2 \cdot \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}) = \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}') + \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}'') \ .$$

F.2 Receiver Compromise Deniability

Theorem 9. *Let* AMF *be the asymmetric message franking scheme defined in Figure 5 using signature of knowledge* SPoK *defined in Appendix D.1. in Figure 5. For all simulators $\mathcal{S}$ for* SPoK*, for any* RecCompDen *adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we give adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathrm{r\text{-}den}}_{\mathrm{AMF}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}) \ .$$

where if $\mathcal{A}$ runs in time T and makes at most Q queries to the frank oracle, then $\mathcal{B}$ and $\mathcal{C}$ run in time $T' \approx T$ and $\mathcal{B}$ makes at most Q queries to its proof oracle.

Proof. This proof follows the same outline as for universal deniability. The difference is that in universal deniability, Forge includes two malformed exponentiations (J and R), while in receiver compromise deniability, RForge includes only one malformed exponentiation (J). This leads to dropping the multiple of 2 on the DDH adversary $\mathcal{C}$ since only one DDH hop is needed. We define G_0 equal to $\mathrm{RecCompDen}^0_{\mathrm{AMF}}$ and G_3 equal to $\mathrm{RecCompDen}^1_{\mathrm{AMF}}$. Pseudocode is given in Figure 14, in which G_0 and G_3 unrolls the Frank and RForge algorithm in $\mathcal{O}^{\mathsf{Frank}}$ and denotes the random oracle explicitly.

G_1 is the same as G_0 except the signature proof of knowledge in the franking oracle is simulated and the random oracle is simulated. We apply the non-interactive zero knowledge property of the signature proof of knowledge to bound this hop.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}') &= \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{B}'}_{\mathsf{SPoK}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{IDEAL}^{\mathcal{B}'}_{\mathcal{S}} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_0 \Rightarrow 1\,\right] - \Pr\left[\,G_1 \Rightarrow 1\,\right]\right|\end{aligned}$$

Game G_2 replaces the well-formed $J \leftarrow (pk_j)^\alpha$ of Frank with the malformed $J \leftarrow g^\gamma$ of RForge for random choice of $(\alpha, \gamma) \xleftarrow{\$} \mathbb{Z}_p^2$. Construct DDH adversary $\mathcal{C}$ that takes (g^x, g^y, g^z) and sets $E_J \leftarrow g^x; pk_j \leftarrow g^y; J \leftarrow g^z$.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{C}) &= \left|\Pr\left[\,\mathrm{DDH}^{\mathcal{C},0}_{\mathbb{G},g} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{DDH}^{\mathcal{C},1}_{\mathbb{G},g} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_2 \Rightarrow 1\,\right] - \Pr\left[\,G_1 \Rightarrow 1\,\right]\right|\end{aligned}$$

```
$\underline{G_0}$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, sk_r, aux) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathrm{ro}}}(pk_s, pk_j)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
if $b^r_{\mathrm{wf}} \neq 1$:
    return $0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, sk_r, pk_j, aux)$
return $b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg)}$:
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x \leftarrow (sk_s, \bot, \alpha, \bot)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
return $\sigma$
```

```
$\underline{G_1}$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, sk_r, aux) \leftarrow \mathcal{A}_1^{\mathcal{S}_{\mathrm{ro}}}(pk_s, pk_j)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
if $b^r_{\mathrm{wf}} \neq 1$:
    return $0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{S}_{\mathrm{ro}}}(sk_s, sk_r, pk_j, aux)$
return $b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg)}$:
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^{\alpha}$
$R \leftarrow (pk_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x \leftarrow (\bot, \bot, \bot, \bot)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathcal{S}_{\mathsf{prove}}(msg, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
return $\sigma$
```

```
$\underline{G_2}$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, sk_r, aux) \leftarrow \mathcal{A}_1^{\mathcal{S}_{\mathrm{ro}}}(pk_s, pk_j)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
if $b^r_{\mathrm{wf}} \neq 1$:
    return $0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{S}_{\mathrm{ro}}}(sk_s, sk_r, pk_j, aux)$
return $b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg)}$:
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$\gamma \leftarrow\$ \mathbb{Z}_p;\ J \leftarrow g^{\gamma}$
$R \leftarrow (pk_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x \leftarrow (\bot, \bot, \bot, \bot)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathcal{S}_{\mathsf{prove}}(msg, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
return $\sigma$
```

```
$\underline{G_3}$:
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$sk_j \leftarrow\$ \mathbb{Z}_p;\ pk_j \leftarrow g^{sk_j}$
$(pk_r, sk_r, aux) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathrm{ro}}}(pk_s, pk_j)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
if $b^r_{\mathrm{wf}} \neq 1$:
    return $0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, sk_r, pk_j, aux)$
return $b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg)}$:
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$\gamma \leftarrow\$ \mathbb{Z}_p;\ J \leftarrow g^{\gamma}$
$R \leftarrow (pk_r)^{\beta}$
$E_J \leftarrow g^{\alpha}$
$E_R \leftarrow g^{\beta}$
$x = (\bot, \gamma, \bot, \beta \cdot sk_r)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
return $\sigma$
```

Fig. 14: Games for the proof of receiver compromise deniability, Theorem 9.

The final game G_3 replaces the simulated proof with the signature proof of knowledge used in RForge. Again, we apply the zero knowledge property of the signature proof of knowledge to bound this hop.

$$\begin{aligned}\mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}'') &= \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{B}''}_{\mathsf{SPoK}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{IDEAL}^{\mathcal{B}''}_{\mathcal{S}} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,G_3 \Rightarrow 1\,\right] - \Pr\left[\,G_2 \Rightarrow 1\,\right]\right|\end{aligned}$$

Define $\mathcal{B}$ as flipping a random bit b, if $b = 0$, $\mathcal{B}$ runs $\mathcal{B}'$, otherwise $\mathcal{B}$ runs $\mathcal{B}''$, which gives

$$2 \cdot \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}) \;=\; \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}') + \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B}'')\ .$$

F.3 Judge Compromise Deniability

Theorem 10. *Let* AMF *be the asymmetric message franking scheme defined in Figure 5 using signature of knowledge* SPoK *defined in Appendix D.1. in Figure 5. For all simulators $\mathcal{S}$ for* SPoK*, for any* JudgeCompDen *adversary $\mathcal{A}$, we give adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{j-den}}_{\mathrm{AMF}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{nizk}}_{\mathsf{SPoK},\mathcal{S}}(\mathcal{B})\ .$$

where if $\mathcal{A}$ runs in time T and makes at most Q queries to the frank oracle, then $\mathcal{B}$ runs in time $T' \approx T$ and makes at most Q queries to its proof oracle.

Proof. This proof follows the same outline as for the other two deniability proofs. The difference is that both universal deniability and receiver compromise deniability (Forge and RForge) use malformed exponentiations of J and R. Since in JForge, J and R are well-formed, the proof does not require any DDH hops. The two hops are to the signature proof simulator and back, since the witnesses used in Frank and Judge are different. We define G_0 equal to $\mathrm{JudgeCompDen}^0_{\mathrm{AMF}}$ and G_2 equal to $\mathrm{JudgeCompDen}^1_{\mathrm{AMF}}$. The pseudocode in Figure 15 for G_0 and G_2 unrolls the Frank and JForge algorithm in $\mathcal{O}^{\mathsf{Frank}}$ and explicitly denotes the random oracle. G_1 is the same as G_0 and G_2 except the signature proof of knowledge in the frank oracle is simulated and the random oracle is simulated. We apply the zero knowledge property of the signature proof of knowledge to bound each hop.

G AMFs from Designated Verifier Primitives

In this section, we take the approach of building an AMF scheme from designated verifier primitives [47]. We provide a candidate construction, but do not prove the scheme secure with respect to our accountability and deniability notions. Lastly, we discuss some of the definitional challenges that arise from this approach.

```
$\underline{G_0:}$
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$(pk_r, sk_r, pk_j, sk_j, aux) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathrm{ro}}}(pk_s)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
$b^j_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_j, sk_j)$
$\mathbf{if}\ b^r_{\mathrm{wf}} \wedge b^j_{\mathrm{wf}} \neq 1$:
    $\mathbf{return}\ 0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, sk_r, sk_j, aux)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (sk_s, \perp, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

```
$\underline{G_1:}$
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$(pk_r, sk_r, pk_j, sk_j, aux) \leftarrow \mathcal{A}_1^{\mathcal{S}_{\mathrm{ro}}}(pk_s)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
$b^j_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_j, sk_j)$
$\mathbf{if}\ b^r_{\mathrm{wf}} \wedge b^j_{\mathrm{wf}} \neq 1$:
    $\mathbf{return}\ 0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{S}_{\mathrm{ro}}}(sk_s, sk_r, sk_j, aux)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (sk_s, \perp, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathcal{S}_{\mathsf{prove}}(msg, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

```
$\underline{G_2:}$
$sk_s \leftarrow\$ \mathbb{Z}_p;\ pk_s \leftarrow g^{sk_s}$
$(pk_r, sk_r, pk_j, sk_j, aux) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathrm{ro}}}(pk_s)$
$b^r_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_r, sk_r)$
$b^j_{\mathrm{wf}} \leftarrow \mathsf{WellFormed}(pk_j, sk_j)$
$\mathbf{if}\ b^r_{\mathrm{wf}} \wedge b^j_{\mathrm{wf}} \neq 1$:
    $\mathbf{return}\ 0$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Frank}}, \mathcal{O}_{\mathrm{ro}}}(sk_s, sk_r, sk_j, aux)$
$\mathbf{return}\ b'$

$\underline{\mathcal{O}^{\mathsf{Frank}}(msg):}$
$(\alpha, \beta) \leftarrow\$ \mathbb{Z}_p^2$
$J \leftarrow (pk_j)^\alpha$
$R \leftarrow (pk_r)^\beta$
$E_J \leftarrow g^\alpha$
$E_R \leftarrow g^\beta$
$x \leftarrow (\perp, \alpha \cdot sk_j, \alpha, \perp)$
$y \leftarrow (g, pk_s, pk_j, J, R, E_J)$
$\pi \leftarrow\$ \mathsf{SPoK}^{\mathcal{R}}.\mathsf{prove}(msg, x, y)$
$\sigma \leftarrow (\pi, J, R, E_J, E_R)$
$\mathbf{return}\ \sigma$
```

Fig. 15: Games for the proof of judge compromise deniability, Theorem 10.

We start by introducing the syntax and security notions of designated verifiers. A designated verifier signature scheme DVS is a tuple of algorithms, (KeyGen, Sign, Verify, Forge), associated with a public key space $\mathcal{PK} \times \mathcal{SK}$, a message space $\mathcal{M}$, and a signature space Σ. The randomized key generation algorithm samples a key pair, $(pk, sk) \leftarrow\$ \mathsf{DVS.KeyGen}$. Messages are signed using the randomized sign algorithm that uses the signer's secret key and the designated verifier's public key to produce a signature, $\sigma \leftarrow\$ \mathsf{DVS.Sign}(sk_s, pk_r, msg)$. Signatures are verified by the designated verifier using the deterministic verify algorithm which takes the signer's public key and the verifier's secret key, $b \leftarrow \mathsf{DVS.Verify}(pk_s, sk_r, msg, \sigma)$. Lastly, it is also possible for a designated verifier to forge such a signature using the randomized forge algorithm, $\sigma \leftarrow\$ \mathsf{DVS.Forge}(pk_s,\ sk_r, msg)$. Correctness dictates

$$\mathsf{DVS.Verify}(pk_s, sk_r, msg,\ \mathsf{DVS.Sign}(sk_s, pk_r, msg)) = 1\ .$$

The two main security properties of DVS schemes are unforgeability and non-transferability, which mirror the accountability and deniability security goals of AMF schemes respectively. Unforgeability ensures a party cannot forge signatures from a signer to a designated verifier without knowledge of either secret keys. Non-transferability ensures that a designated verifier cannot transfer conviction of a valid signature to another party by requiring that a signature created from DVS.Sign is indistinguishable from a forgery from DVS.Forge even with access to the designated verifier's secret key.

A *strong* designated verifier signature scheme SDVS extends a DVS scheme by adding a stronger forge algorithm, SForge, which can be used to forge signatures between two parties without knowledge of the parties' secret keys, $\sigma \leftarrow\$ \mathsf{SDVS.\ SForge}(pk_s, pk_r, msg)$. The strong deniability security property of SDVS ensures that a signature from SDVS.Sign is indistinguishable from a forgery from SDVS.SForge to all parties without knowledge of the designated verifier's secret key.

Analogous to our definition of DVS, a non-interactive designated verifier proof system niDV is a tuple of algorithms, (KeyGen, Prove, Verify, Forge), associated with a public key space $\mathcal{PK} \times \mathcal{SK}$ and a witness-statement relation $\mathcal{R}$. A relation $\mathcal{R}$ is a set of witness-statement pairs; formally, $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the witness space and $\mathcal{Y}$ is the statement space. A prover can designate the proof of a witness x for statement y to a verifier using the randomized prove algorithm, $\pi \leftarrow\$ \mathsf{niDV.Prove}_{\mathcal{R}}(pk_r, x, y)$. The designated verifier can both verify the proof ($b \leftarrow \mathsf{niDV.Verify}_{\mathcal{R}}(sk_r, y, \pi)$) and forge the proof ($\pi \leftarrow\$ \mathsf{niDV.Forge}_{\mathcal{R}}(sk_r, y)$) with their secret key. The correctness and non-transferability properties from DVS are defined analogously for niDV. In addition, niDV schemes utilize two more security properties: knowledge soundness and zero knowledge. Briefly, knowledge soundness ensures that a prover that generates a valid proof for a message must actually "know" a witness for the statement. A scheme being zero knowledge ensures that verification of a proof does not reveal anything about the witness to the designated verifier other than if it is valid or not. Finally, a

niDV scheme can be extended to a strong designated verifier proof system niSDV by including a strong forge algorithm SForge and satisfying the strong deniability security property as before.

Following the intuition from Section 4.1, we build our scheme as a strong designated verifier proof to the receiver of the well-formedness of a strong designated verifier signature to the moderator. The pseudocode for the AMF algorithms of this candidate construction is provided in Figure 16.

```
$\mathcal{R} = \{(sk_s, (pk_s, pk_j, msg, \sigma)) \ : \ \mathsf{SDVS.Sign}(sk_s, pk_j, msg) = \sigma\}$
$\mathcal{R}' = \{(sk_j, (pk_s, pk_j, msg, \sigma)) \ : \ \mathsf{SDVS.Forge}(sk_j, pk_j, msg) = \sigma\}$

$\underline{\mathsf{KeyGen}:}$
$pk^{\mathsf{DVS}}, sk^{\mathsf{DVS}} \xleftarrow{\$} \mathsf{SDVS.KeyGen}$
$pk^{\mathsf{niDV}}, sk^{\mathsf{niDV}} \xleftarrow{\$} \mathsf{niSDV.KeyGen}$
$\mathbf{return}\ (pk^{\mathsf{DVS}}, pk^{\mathsf{niDV}}), (sk^{\mathsf{DVS}}, sk^{\mathsf{niDV}})$

$\underline{\mathsf{Frank}(pk_s, sk_s, pk_r, pk_j, msg):}$
$(sk_s^{\mathsf{DVS}}, sk_s^{\mathsf{niDV}}),\ (pk_r^{\mathsf{DVS}}, pk_r^{\mathsf{niDV}}),\ (pk_j^{\mathsf{DVS}}, pk_j^{\mathsf{niDV}}) \leftarrow sk_s, pk_r, pk_j$
$\sigma_j \xleftarrow{\$} \mathsf{SDVS.Sign}(sk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg)$
$\pi_r \xleftarrow{\$} \mathsf{niSDV.Prove}_{\mathcal{R}}(pk_r^{\mathsf{niDV}}, sk_s^{\mathsf{DVS}}, (pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg, \sigma_j))$
$\mathbf{return}\ (\sigma_j, \pi_r)$

$\underline{\mathsf{Verify}(pk_s, sk_r, pk_j, msg, \sigma):}$
$(pk_s^{\mathsf{DVS}}, pk_s^{\mathsf{niDV}}),\ (sk_r^{\mathsf{DVS}}, sk_r^{\mathsf{niDV}}),\ (pk_j^{\mathsf{DVS}}, pk_j^{\mathsf{niDV}}) \leftarrow pk_s, sk_r, pk_j$
$(\sigma_j, \pi_r) \leftarrow \sigma$
$\mathbf{return}\ \mathsf{niSDV.Verify}_{\mathcal{R}}(sk_r^{\mathsf{niDV}}, (pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg, \sigma), \pi_r)$

$\underline{\mathsf{Judge}(pk_s, pk_r, sk_j, msg, \sigma):}$
$(pk_s^{\mathsf{DVS}}, pk_s^{\mathsf{niDV}}),\ (pk_r^{\mathsf{DVS}}, pk_r^{\mathsf{niDV}}),\ (sk_j^{\mathsf{DVS}}, sk_j^{\mathsf{niDV}}) \leftarrow pk_s, pk_r, sk_j$
$(\sigma_j, \pi_r) \leftarrow \sigma$
$\mathbf{return}\ \mathsf{SDVS.Verify}(pk_s^{\mathsf{DVS}}, sk_j^{\mathsf{DVS}}, msg, \sigma_j)$

$\underline{\mathsf{Forge}(pk_s, pk_r, pk_j, msg):}$
$(pk_s^{\mathsf{DVS}}, pk_s^{\mathsf{niDV}}),\ (pk_r^{\mathsf{DVS}}, pk_r^{\mathsf{niDV}}),\ (pk_j^{\mathsf{DVS}}, pk_j^{\mathsf{niDV}}) \leftarrow pk_s, pk_r, pk_j$
$\sigma_j \xleftarrow{\$} \mathsf{SDVS.SForge}(pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg)$
$\pi_r \xleftarrow{\$} \mathsf{niSDV.SForge}_{\mathcal{R}}(pk_r^{\mathsf{niDV}}, (pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg, \sigma_j))$
$\mathbf{return}\ (\sigma_j, \pi_r)$

$\underline{\mathsf{RForge}(pk_s, sk_r, pk_j, msg):}$
$(pk_s^{\mathsf{DVS}}, pk_s^{\mathsf{niDV}}),\ (sk_r^{\mathsf{DVS}}, sk_r^{\mathsf{niDV}}),\ (pk_j^{\mathsf{DVS}}, pk_j^{\mathsf{niDV}}) \leftarrow pk_s, sk_r, pk_j$
$\sigma_j \xleftarrow{\$} \mathsf{SDVS.SForge}(pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg)$
$\pi_r \xleftarrow{\$} \mathsf{niSDV.Forge}_{\mathcal{R}}(sk_r^{\mathsf{niDV}}, (pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg, \sigma_j))$
$\mathbf{return}\ (\sigma_j, \pi_r)$

$\underline{\mathsf{JForge}(pk_s, pk_r, pk_j, sk_j, msg):}$
$(pk_s^{\mathsf{DVS}}, pk_s^{\mathsf{niDV}}),\ (pk_r^{\mathsf{DVS}}, pk_r^{\mathsf{niDV}}),\ (sk_j^{\mathsf{DVS}}, sk_j^{\mathsf{niDV}}) \leftarrow pk_s, pk_r, sk_j$
$\sigma_j \xleftarrow{\$} \mathsf{SDVS.Forge}(pk_s^{\mathsf{DVS}}, sk_j^{\mathsf{DVS}}, msg)$
$\pi_r \xleftarrow{\$} \mathsf{niSDV.Prove}_{\mathcal{R}'}(pk_r^{\mathsf{niDV}}, sk_j^{\mathsf{DVS}}, (pk_s^{\mathsf{DVS}}, pk_j^{\mathsf{DVS}}, msg, \sigma_j))$
$\mathbf{return}\ (\sigma_j, \pi_r)$
```

Fig. 16: AMF construction from SDVS and niSDV primitives.

Proving AMF security definitions with respect to the security definitions of the underlying designated verifier primitives is not a straightforward task. Universal deniability and receiver compromise deniability are relatively straightforward, in which a proof might use two game hops: the first switching the designated verifier proof from prove to forge, and the second switching the designated verifier signature from sign to forge. However, judge compromise deniability and the accountability definitions are more challenging. In judge compromise deniability, it is not clear how the judge forging algorithm should create a proof to the receiver of well-formedness of the forged signature. It may be that we require a stronger non-transferability

```
$\text{Gap-DL}^{\mathcal{A}}_{\mathbb{G},g}$:
$x \leftarrow\!\$\ \mathbb{Z}_p$
$x' \leftarrow \mathcal{A}^{\mathcal{O}}(g^x)$
return $x = x'$

$\mathcal{O}(Y, Z)$:
return $Y^x = Z$
```

```
$\text{Gap-CDH}^{\mathcal{A}}_{\mathbb{G},g}$:
$(x, y) \leftarrow\!\$\ (\mathbb{Z}_p)^2$
$Z \leftarrow \mathcal{A}^{\mathcal{O}}(g^x, g^y)$
return $Z = g^{xy}$

$\mathcal{O}(Y, Z)$:
return $Y^x = Z$
```

Fig. 17: Security games for Gap-CDH and Gap-DL.

notion for the signature called perfect non-transferability in which a signature and a designated verifier's forgery are information theoretically indistinguishable. Alternatively, we can target the strong authentication deniability goals outlined in Appendix B, in which the judge forge algorithm is not accepted by receiver verification. Finally, the accountability definitions pose a challenge due to the mismatch between the verification oracles used in the AMF definitions and the lack of verification oracles in designated verifier signature unforgeability games. Bellare et al. [8] show the power of the verification oracle in the related setting of message authentication. Using their result for strong unforgeability, we can transform the designated verifier unforgeability games to ones with verification oracles for the fixed pk_s, pk_r of the game, but that still does not match our oracles which allow verification queries for arbitrary pk_s, pk_r.

H Replacing KEA with Gap-CDH

In this section, we show how to alter our scheme to dispense with relying on the KEA. We show that extending the size of the franking signature with two extra clauses allows us to reduce to the Gap-CDH assumption. Our proof strategy is to first reduce to a related assumption, we call Gap-DL, which in turn reduces to Gap-CDH. The Gap-CDH security game and our proposed Gap-DL assumption are given in Figure 17. Note that the Gap-CDH game we present allows queries DDH queries only with respect to a fixed element $X \leftarrow g^x$. This is in contrast to the Gap-CDH assumption used by BLS01 [16] which allows arbitrary DDH queries; our variant is stronger. To the best of our knowledge, the Gap-DL assumption has not been previously studied. It is easy to see that Gap-CDH ⇒ Gap-DL in the same way that CDH⇒DL.

Recall that the KEA extractor is used in two places in the accountability proofs. (1) To simulate the judge and verify oracle queries without knowledge of sk_r or sk_j, and (2) To learn sk_j and sk_r from the extracted witness, $u = sk_j \cdot \alpha$ and $w = sk_r \cdot \beta$. We will use the oracle of the Gap-DL game to address (1). To address (2), we will extend the statement of the signature of knowledge to include knowledge of α and β. Consider the extended relation,

$$\mathcal{R}' = \Big\{\big((t,\, u,\, v, w),\, (g,\, pk_s,\, pk_j,\, J,\, R,\, E_J)\big) : \\ \big(pk_s = g^t \vee J = g^u\big) \;\wedge\; \big(J = (pk_j)^v \vee R = g^w\big) \;\wedge\; E_J = g^v \;\wedge\; E_R = g^w\Big\}.$$

We then give the following theorem for receiver binding.

Theorem 2. *Let* AMF *be the message franking scheme using signature of knowledge* SPoK *defined in Figure 5 over relation $\mathcal{R}'$ defined in Appendix H, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any* r-BIND *adversary $\mathcal{A}$ making at most Q_{Frank} franking oracle queries, Q_{Judge} judge oracle queries, and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\text{AMF}}(\mathcal{A}) \le \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}} + Q_{ro} + 1)}{p^4} + \frac{Q_{ro} + 1}{p} + \sqrt{2(Q_{ro} + 1) \cdot \mathbf{Adv}^{\text{gapcdh}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of group $\mathbb{G}$ and if $\mathcal{A}$ runs in time T, then $\mathcal{B}$ runs in time $T' \approx 2T$.

Proof sketch: The proof follows the same as Theorem 1 until G_3. G_3 and G_4, which introduce KEA extractors, are no longer necessary. Instead, we define a Gap-DL adversary $\mathcal{D}$ that simulates G_2 by forwarding

$\mathbf{WF}(\mathcal{A})$	$\mathbf{Adv}^{\text{unf}}_{\text{AMF}}(\mathcal{A})$	Q_{ro}	p	$\mathbf{WF}(\mathcal{B})$	$\mathbf{WF}(\mathcal{P})$
2^{128}	2^{-40}	2^{128}	2^{665}	2^{339}	2^{341}
2^{128}	2^{-40}	2^{57}	2^{521}	2^{268}	2^{268}
2^{128}	2^{-40}	2^{52}	2^{512}	2^{263}	2^{264}
2^{128}	2^{-20}	2^{72}	2^{512}	2^{263}	2^{264}

Fig. 18: A sampling of security parameters for various choices of random oracle query budget Q_{ro}, adversary advantage, and group size p. $\mathbf{WF}\mathcal{A}$ is set so security parameter $\kappa = 128$. Compare work factor of DL adversary $\mathcal{B}$ to work factor of Pollard's rho algorithm.

oracle queries to its own oracle and extracts a witness from the winning signature via rewinding. The rewinding argument follows exactly as in Theorem 1 where the DL adversary rewinds on G_4. As in Theorem 1, adversary $\mathcal{D}$ is actually a composition of two Gap-DL adversaries, one of which breaks the discrete log of pk_s and the other of pk_j. The adversary that break pk_j uses the extended signature proof statement to learn sk_j from α. Finally, we define Gap-CDH $\mathcal{B}$ as a simple wrapper around $\mathcal{D}$ that takes $\mathcal{D}$'s output x and returns $Z = Y^x$.

We provide Gap-CDH statements for the remaining two accountability properties below without additional proof.

Theorem 11. *Let* AMF *be the message franking scheme using signature of knowledge* SPoK *defined in Figure 5 over relation $\mathcal{R}'$ defined in Appendix H, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any* s-BIND *adversary $\mathcal{A}$ making at most Q_{Verify} verify oracle queries and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{s-bind}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{ro}+1}{p} + \sqrt{(Q_{ro}+1) \cdot \mathbf{Adv}^{\text{gapcdh}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of group $\mathbb{G}$ if $\mathcal{A}$ runs in time T, then $\mathcal{B}$ runs in time $T' \approx 2T$.

Corollary 2. *Let* AMF *be the message franking scheme using signature of knowledge* SPoK *defined in Figure 5 over relation $\mathcal{R}'$ defined in Appendix H, where* SPoK *is derived using the Fiat-Shamir heuristic as described in Appendix D.1 using hash function $\mathcal{H}$. If $\mathcal{H}$ is modeled as a random oracle, for any* UNF *adversary $\mathcal{A}$ making at most Q_{Frank} franking oracle queries, Q_{Verify} verify oracle queries, Q_{Judge} judge oracle queries, and Q_{ro} random oracle queries, we give adversary $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{unf}}_{\text{AMF}}(\mathcal{A}) \leq \frac{Q_{\mathsf{Frank}}(Q_{\mathsf{Frank}}+Q_{ro}+1)}{p^4} + \frac{2(Q_{ro}+1)}{p} + \sqrt{4(Q_{ro}+1) \cdot \mathbf{Adv}^{\text{gapcdh}}_{\mathbb{G},g}(\mathcal{B})}$$

where p is the order of group $\mathbb{G}$ if $\mathcal{A}$ runs in time T, then $\mathcal{B}$ runs in time $T' \approx 2T$.

I Measuring Concrete Security

Measuring concrete security allows us to verify the efficiency of our reductions. Here we specifically focus on the security of Corollary 1, unforgeability, because it presents the least efficient reduction. We utilize work factors to measure concrete security, as in BR09 [12]. We define the *work factor* of any adversary $\mathcal{A}$ running in time $\mathbf{T}(\mathcal{A})$ and gaining advantage ϵ as $\mathbf{WF}(\mathcal{A}) = \mathbf{T}(\mathcal{A})/\epsilon$. The work factor of an adversary against a particular security notion constructed by a proof should be less than that of the best known attack for the same security notion. If this is so, then it implies that the adversary constructed by the reduction must have a greater efficiency than that of the best known attack, a contradiction.

The reduction for Corollary 1 constructs both $\mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})$ and $\mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}})$. However, since KEA is a stronger assumption than that of DL, we approximate the best attack against DL as the same as that for KEA and therefore accordingly approximate $\mathbf{Adv}^{\text{kea}}_{\mathbb{G},g}(\mathcal{C}, \mathcal{E}_{\mathcal{C}})$ with $\mathbf{Adv}^{\text{dl}}_{\mathbb{G},g}(\mathcal{B})$ [10]. We then want to compare

$\mathbf{WF}(\mathcal{B})$ to the work factor of the best known attack against DL. We use Pollard's rho algorithm for the best attack, whose work factor is

$$\mathbf{WF}(\mathcal{P}) \approx 0.88 \frac{\sqrt{p} \log^2 p}{10^3}$$

where p is the size of the group [39]. We expect that $\mathbf{WF}(\mathcal{B}) \leq \mathbf{WF}(\mathcal{P})$.

We first find $\mathbf{T}(\mathcal{B})$ by the runtime provided in the theorem statement. A challenge in performing concrete analysis using KEA is that the extractor is not instantiated concretely. We approximate the running time of extractor $\mathcal{E}_{\mathcal{C}}$ by observing it intuitively functions by observing the code of $\mathcal{C}$ to successfully extract the desired exponent, $t_{\mathcal{E}} \approx \mathbf{T}(\mathcal{C}) \approx \mathbf{T}(\mathcal{A})$. Setting the query budgets for Q_{Frank}, Q_{Verify}, and Q_{Judge} to be 2^{40} gives us $\mathbf{T}(\mathcal{B}) \approx 2^{42} \cdot \mathbf{T}(\mathcal{A})$.

Next we compute $\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g}(\mathcal{B})$ instantiated over a prime order cyclic group $\mathbb{G}$ for varying sizes of p. We define the security parameter of our scheme as $\kappa \geq \log \mathbf{WF}(\mathcal{A})$. For $\kappa = 128$, we set $\mathbf{WF}(\mathcal{A}) = 2^{128}$. Figure 18 presents a variety of security parameters for which $\mathbf{WF}(\mathcal{B}) \leq \mathbf{WF}(\mathcal{P})$ varying random oracle query budget Q_{ro}, split of $\mathbf{WF}(\mathcal{A})$ between time and advantage, and group size p.