k-Anonymous Group Signatures: Addressing Strict Content Moderation in End-to-End Secure Messaging Platforms

Shalini Banerjee[†,‡], Andrey Bozhko[⋆], Andy Rupp[⋆,‡]

[†]Karlsruhe Institute of Technology
[‡]KASTEL Security Research Labs
[⋆]University of Luxembourg
shalini.banerjee@kit.edu, {andrey.bozhko, andy.rupp}@uni.lu

October 27, 2025

Abstract

We review k-anonymity in authentication schemes, group signatures and ring signatures. While existing constructions achieve unlinkability, they typically necessitate maintaining the state or relying on computationally expensive tracing algorithms. We propose a *stateless* variant that is efficiently traceable, *albeit necessarily fully linkable*. To the best of our knowledge, our variant, which we call k-Anonymous Group Signatures (k-AGS), is the first scheme to combine statelessness and efficient traceability.

Building upon our k-AGS framework, we design k-Anonymous Set Pre-Constrained Group Signatures (k-ASPCGS) which is a *threshold* extension of the Set Pre-Constrained Group Signatures (SPCGS) introduced by Bartusek et al. (EUROCRYPT 2023).

We show that our notions arise naturally in the context of lawful surveillance, particularly for end-to-end secure messaging platforms, where controlled traceability is essential. Beyond this setting, they may also help mitigate the impact of strict moderation policies in large-scale distributed asynchronous platforms (e.g. Facebook, whistleblowing portals) as well as in spam control, where false positives remain a persistent challenge.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Our Approach . . . 5
- 1.3 Discussions . . . 9

2 Preliminaries **11**
- 2.1 Notations . . . 11
- 2.2 Simulation-based security . . . 11
- 2.3 Public Key Encryption . . . 11
- 2.4 Blind Signatures . . . 12
- 2.5 Polynomial Commitments . . . 14
- 2.6 Subversion-Resistant Non-Interactive Zero-Knowledge . . . 15
- 2.7 Set Pre-Constrained Encryption . . . 17
- 2.8 Hash to Field . . . 19

3 k-Anonymous Group Signatures **19**
- 3.1 Definition and Security Notions . . . 19
- 3.2 Construction . . . 24

4 k-Anonymous Set Pre-constrained Group Signatures **28**
- 4.1 Definition and Security Notions . . . 28
- 4.2 Construction . . . 30
- 4.3 Generic Transform . . . 32

A Traceability Experiment for k-ASPCGS **36**

B Security Proofs of k-AGS **37**
- B.1 Traceability . . . 37
- B.2 k-Anonymity . . . 41

C k-ASPCGS Construction **44**

D k-ASPCGS Generic Transform Security Proofs **47**

1 Introduction

The proliferation of illegitimate contents over the Internet has invariably indicated the significance of implementing *content moderation* technologies that examine user-generated information to identify such contents along with their originators. Nevertheless, the task has proven to be fundamentally incompatible with ubiquitous end-to-end encryption (E2EE) services that enable exclusive content-sharing among endpoints, without disclosing keys to the service providers.

The issue received a lot of attention in the recent literature when various legislative bodies (with regulations such as EARN IT [Pre20], TERREG [EU21]) compelled service providers of apps or online services to perform content moderation, with or lacking E2EE. The most prominent scenario has been in the context of distribution of child sexual abuse material (CSAM), and terrorist and extremist contents online, where the service providers are legally bound to trace such harmful contents and their originators, conceivably with E2EE. Alongside that, contributing to the fundamental right declaration, there is a strong need to protect the privacy of honest users. The 2025 EU proposal for Regulation of European Union and Parliament [CoEU25] has further established that CSAM contents should not only cover distribution of previously detected and confirmed illegal contents, but also new contents likely to constitute CSAM that has not been confirmed before (reminiscent of the "retrospective" approach considered in [GKVL21]).

Responding to the contrasting public-policy and security demands, the service providers of end-to-end messaging platforms (such as Facebook, WhatsApp, Signal) have taken a standard approach: allow a user to report a decrypted content she believes to be illegitimate, and rely on known content detection methodologies or classifier based methods to trace the originators. While the former relies on a database of non-cryptographically hashed *pre-defined illegitimate contents*[1] (perpetual hash functions such as Microsoft PhotoDNA, pHash), the latter detects *new instances of illegitimate contents*.

Strict Content Moderation. The problem of content tracing has already received attention in the existing literature, primarily by means of *message franking*. While there exists a number of expositions targeting variants of definitions, extensions and realizations, the general approach is to trace an originator when a message gets reported to the service provider [TGL+19, IAV22, LZH+23]. However, the methods do not prevent a malicious reporter and server to collude and trigger a traceback for an originator of *any* message, regardless of its content.

Augmenting the framework to incorporate a dataset $\mathcal{D}$ of predefined illegitimate contents, Bartusek, Garg, Jain and Policharla [BGJP22] construct set pre-constrained group signatures (SPCGS) that allow tracing an originator of a reported content, provided the content indeed belongs to $\mathcal{D}$. In fact, the essence of constraining the power of tracing originators of messages is built upon the foundational work of *pre-constrained encryption* (PCE) by Ananth, Jain, Jin and Malavolta [AJ22], which allows encrypting a message that can be decrypted only if it satisfies some predicate, where the predicate is pre-constrained in the system setup phase. At a high level, the [BGJP22] scheme works as follows: a group manager encodes a database $\mathcal{D}$ in the public key of the group during the setup phase. To sign a message msg, a sender encrypts his identity id w.r.t. msg using a PCE scheme, such that id can be recovered from the ciphertext ct iff $msg \in \mathcal{D}$. Given the system is end-to-end encrypted, the server[2] learns no information on the exchanged contents. Only reported illegitimate contents allow tracing its originator by the server, while the harmless contents cannot be traced even if a malicious reporter and a server collude.

While a worthwhile task, what happens when an honest user sends some illegitimate content with a non-malicious motive? To see this, consider the scenario where a man communicated his son's image for medical diagnosis, which was eventually flagged as CSAM by Google, followed by criminal investigations against him [New25]. A recent open letter addressing the EU Regulation asserts how the existing techniques fail to incorporate *context* into content moderation approaches, leading to unfair penalties against honest users [Let25]. Alongside this, the state-of-art moderation systems employ non-cryptographic perpetual hashes, yielding high false positive rates [May19]. Finally, classifier-based content moderation systems,

[1] predefined illegitimate contents are ideally defined by third-parties such as National Center for Missing and Exploited Children (NCMEC), Global Internet Forum to Counter Terrorism (GIFCT), etc.

[2] we use the terms "server" and "service provider" interchangeably throughout the paper.

extensively employed by Facebook and Google, fail to understand cultural nuances and local context in identifying new illegal contents, scaling up the surface of false positives [fJ21, Let25, May19]. Quite evidently, such overtly zealous nature of moderation systems suppresses legitimate expression, undermines trust, and ultimately limits user participation. This makes us ask:

Can we circumvent inadvertent censorship caused by strict content moderation systems?

We believe that factoring in wider considerations while designing traceback criteria would constitute a critical starting point in countering overzealous moderation. One clear and straightforward solution is to consider *thresholding in the sense that anonymity is preserved for a reportee until threshold number k of illegitimate contents has been reported by his peers.* Our rationale behind this approach is to ascertain that a flagged reportee is indeed likely to be malicious given he distributes multiple illegitimate contents to his peers.

1.1 Our Contributions

We now summarize our main results.

- We review k-anonymity in authentication schemes [NSN05, TFS04, TS06], group signatures [ASY06, CHL06], ring signatures [CLXZ25, TNL+25], and traceback systems [BGH+25, LRTY21]. Existing constructions require users to employ a stateful tracking method for achieving traceability and/or unlinkability, with some exceptions where statelessness comes at the expense of slow tracing/linking algorithms. We introduce a new variant of k-anonymity that is *both stateless and efficiently traceable*; we call our variant "k-anonymous group signatures" (k-AGS).
  - *Definition.* A k-AGS scheme refers to a system with a group manager and multiple users who communicate with each other preferably over an end-to-end secure messaging platform. To incorporate the notion of k-anonymity, we embed the scheme anonymity parameter k directly into the group's public key. Users asynchronously send messages along with signatures. When k signatures are reported, a group manager can efficiently trace the identity of the signer.
  - *Construction.* We generically construct k-AGS, assuming a polynomial commitment scheme, a blind signature scheme, a hash to field function and a subversion-resistant NIZK. We note that our scheme is "fully linkable". It remains unclear whether stateless unlinkable group signatures with efficient tracing can be achieved; *we leave this as an open question for future work.*
  - *Applications.* We argue that our k-AGS concept is natural, and is directly applicable to securing anonymity of users in distributed and asynchronous environments where maintaining a global state is impractical. In the interest of establishing significance of k-AGS, we mainly highlight lawful surveillance applications, where k-AGS can be used to circumvent high false positive rates exhibited by overzealous content moderation platforms (classification-based) in end-to-end secure messaging platforms.
- Our next contribution is in formulating "k-anonymous set pre-constrained group signatures" (k-ASPCGS), which can be viewed as a threshold extension of set pre-constrained group signatures introduced in [BGJP22]. Our formalism builds on our k-AGS security definitions, while incorporating an additional property: a stateless sender is traceable only after producing k signatures on messages drawn from a predefined set of illegitimate messages.
  - *Definition.* We encode a database of illegitimate contents $\mathcal{D}$, along with a threshold anonymity parameter k in the group's public key. As in our k-AGS, users send messages asynchronously, along with signatures. A group manager can efficiently deanonymize a signer who has generated k reported signatures on messages from $\mathcal{D}$.

– *Construction.* As a proof-of-concept, we present two constructions of k-ASPCGS—the first one is obtained via a simple modification of a k-AGS, while the second follows from a generic transformation from k-AGS to k-ASPCGS. Both constructions additionally rely on a subversion-resistant NIZK and a set pre-constrained encryption scheme (see Definition 5). Given that our k-ASPCGS extends the properties of an underlying k-AGS, it is stateless and efficiently traceable, albeit fully linkable. We emphasize that, if an unlinkable k-AGS scheme were to be realized, the corresponding k-ASPCGS construction would also achieve unlinkability.

– *Applications.* Our k-ASPCGS notion directly applies to the lawful surveillance applications, where a user is traceable only after disseminating k illegitimate reported messages (pre-defined in $\mathcal{D}$), thereby narrowing the space of false positives in overzealous content moderation systems (known content-based) in end-to-end secure messaging platforms.

1.2 Our Approach

The purpose of this paper is to design a threshold-based content moderation system in the sense that a user's (sender's) identity would remain protected from a server until a pre-defined number k of different illegitimate contents are reported by his communicating peers, where k can be any polynomial in the security parameter. Finally, we require: (a) messages exchanged within peers should remain confidential, (b) originators of legitimate contents should have their identities protected, even if malicious reporter(s) and server collude, and (c) originators of illegitimate contents can be traced *only* after k contents are reported against him. *Said otherwise, we aim to design a k-anonymous set pre-constrained group signatures* scheme.

Settings where Previous Constructions are Limiting. Having established our goal, we note that developing an efficient k-ASPCGS scheme could be a challenging task. In fact, furthering the stronger requirement of thresholding in SPCGS, where multiple matches would deanonymize an originator, has previously been pointed out by [BGJP22] as appealing yet non-trivial to realize with concrete efficiency guarantees.

Intuitively, one might expect to find a solution by employing Shamir's secret sharing [Sha79]. More specifically, instead of encrypting id w.r.t. msg, a sender (functioning as a "dealer") now computes shares of id, and encrypts a fresh share for every message he sends. Unfortunately, this approach alone does not capture our intuition. The reason is that there are "multiple" senders, each encrypting an id-share for every message being communicated, which inarguably indicates that the reconstruction algorithm needs to be able to distinguish k shares corresponding to a specific id from a large pool of reported ciphertexts.

This problem has been studied previously by Eldridge, Beck, Green, Heninger and Jain [EBG+24] as multi-dealer secret sharing (MDSS). They give a solution to MDSS by employing list decoding for Reed-Solomon (RS) codes, which bounds the number of dealers for the reconstruction to succeed. Clearly, we cannot restrict the number of content originators in an SPCGS scheme. Another approach [EBG+24] based on interleaved RS codes imposes better bounds, however at the cost of achieving only heuristic correctness. Additionally, we observe that the MDSS formalization does not incorporate a centralized authority or group manager, which is essential in our setting. Thus, it remains to determine what makes a k-ASPCGS scheme realizable under the aforementioned constraints.

Our starting point is the seminal work by Teranishi, Furukawa and Sako [TFS04], which introduces the notion of *k-Times Anonymous Authentication* (k-TAA). Conceptually, the notion allows a user to "anonymously" authenticate himself until k-times, beyond which his identity is traceable to a server (relevant in e-cash, e-voting, trial content-browsing applications). The key idea behind their construction is to have the server publish k hints, where users are registered by a separate entity called the group manager. During authenticating, a user computes a tag using a published hint, along with a proof that one of the k hints has been used to compute the tag. As long as a different hint is used per authentication session, the computed tags along with the authentication log reveal no information on the identity of the user. Therefore, the user cannot send more than k messages anonymously—they are forced to reuse a hint when computing a tag for the $(k+1)$-th message. Obviously, an honest user in such a scheme has to be stateful, keeping track of which of the hints he has already used. These results are subsequently extended in [TS06, NSN05], all of which achieve k-unlinkability (as defined in [CLXZ25]).

Given our goal is to construct k-ASPCGS, we proceed step-by-step by first designing a *variant* of k-TAA, where we merge the roles of a group manager and an authenticating server (unlike the existing constructions [TS06, TS06, EBG+24]). This is because our setting essentially requires the server to *both* register users, as well as audit originators of illegal contents. We note that having a singular authority gives us an added advantage: the server and the user can now share some additional information hint generated during registration, which can later be used for efficient tracing. In our setting, we have a system with a single server and multiple users. Each user registers with the server to obtain an identity id, and a secret identity signing key sk and some shared information hint. A user then signs messages using sk and hint, and asynchronously sends messages (possibly encrypted) along with signatures to his peers. When peers report signatures, the server can reconstruct id using hints, provided it has received more than k signatures computed by the user having identity id. We additionally extend this intuition to incorporate "statelessness", since neither server nor the users can keep track of which hints are already used by which sender. We call our variant *k-Anonymous Group Signatures*. Note that, for a stateless scheme, a hint essentially serves the same role as a secret key. Hence, we find it natural to include it directly into the secret key sk of the user.

We stress that the existing k-anonymous constructions and variants thereof are stateful (except for [CLXZ25, EBG+24] which are not efficiently traceable). Our k-AGS notion is applicable in scenarios where classifier-based content moderation systems exhibit high false positive rates, often flagging honest users incorrectly. By requiring k reports before deanonymizing a sender mitigates the loss of premature loss of identity due to isolated misclassifications. Beyond moderation, k-AGS applies to whistleblowing portals, anonymous platforms, and other asynchronous and distributed environments where maintaining a global state is impractical.

A Flawed Attempt. We begin with a naïve approach to transform a stateful k-TAA scheme to a stateless one that allows the recovery of a user's identity with a probability $\frac{1}{2}$ after k messages, and with a probability 1 after k^2 messages. During registration, the server generates k^2 hints $\{\mathsf{hint}_1, \ldots, \mathsf{hint}_{k^2}\}$. To send a message msg, the user computes $c = f(msg)$, where f is a function (possibly a random oracle) mapping the message space to $\{1, \ldots, k^2\}$. The user then generates a proof and uses hint_c to produce a tag, following the procedure of the underlying stateful scheme. At a first glance, this approach seems to work: the probability that a hint is reused after ℓ messages is roughly $\frac{l^2}{2k^2}$ (modeling f as a random oracle). Hence, a user who sends a small number of messages is unlikely to be deanonymized.

However, upon closer inspection, we observe two limitations. *Firstly*, if k^2 is chosen to be small, then an honest user sending small number of messages gets deanonymized with noticeable probability due to frequent collisions. *Secondly*, if k^2 is chosen to be large, then an adversary could flood the system with many "bad" messages, since they can track used hints. Hence, this transformation seems to be unsatisfactory. Our goal, instead, is to design a scheme in which even honest users who send a few messages that are mistakenly marked as "bad" retain anonymity, without relying on a delicate choice of k.

In what follows, we provide a technical overview of our contributions and how our approaches work, with details deferred to the respective sections.

k-Anonymous Group Signatures. We design our k-AGS scheme with a primary focus on efficiency of the tracing procedure. As discussed earlier, the straightforward approach—having a signer include a fresh Shamir secret share of their identity id in each signature—leads to a tracing algorithm that must check $(k+1)$ subsets of all reported messages before identifying a signer beyond the threshold. At the same time, sticking to the secret-sharing approach remains preferable: it yields an efficient signing algorithm and guarantees perfect anonymity of the signer until the threshold is reached. Our goal, therefore, is to retain the secret-sharing of id while designing a scheme that enables efficient tracing.

This highlights the main benefit of allowing efficient linkability of signatures produced by the same user. Linkability makes tracing efficient while still retaining the advantages of the secret-sharing approach. The tracing algorithm can first link signatures from the same user and then, once the threshold is reached, reconstruct the identity from the shares. The main challenge in designing such a scheme is to "separate" the mechanism used for linking signatures from the user's actual identity, thereby guaranteeing anonymity before the threshold while preserving unforgeability.

In our k-AGS construction, we adopt a direct approach to achieve linkability. During the key generation

protocol between the user and the group manager, the user—together with their identity id, which is known to the group manager—obtains a unique *pseudo-identity* pid. Crucially, pid remains hidden from the group manager, and can only be generated by a user who successfully completes the key generation protocol. The pseudo-identity is then used to produce signatures and is explicitly included in each signature.

Before explaining the structure of pid, let us revisit what we mean by "having the signer include a fresh Shamir secret share of their identity id into a signature." Since the signer is stateless, the only[3] available source of "freshness" for computing a share is the message itself. Following the thresholding approach of [BBM+21], the signer computes the x-coordinate of a share by hashing the message and their pseudo-identity into a (sufficiently large) finite field, i.e., $x = \mathsf{H2F}(\mathsf{pid} \,\|\, msg)$, and then sets the share as $\mathsf{sh} = (x, f(x))$, where f is a polynomial of degree k with $f(0) = \mathsf{id}$.

At this point, a malicious client[4] could attempt to cheat by including a random y instead of evaluating $f(x)$. Thus, we must enforce that the user (a) correctly evaluates f at x for every signature, and (b) consistently uses the same polynomial f across all signatures. To achieve this, we employ polynomial commitments [KZG10]. During the key generation protocol, the user commits to a chosen polynomial f, proving that it satisfies $f(0) = \mathsf{id}$. Each signature then includes an opening of this commitment at x, along with the commitment com itself.

To prevent the group manager from trivially deanonymizing the user by linking com with id during key generation, the user does not simply hand over com. Instead, the user obtains a blind signature on the commitment, while proving in zero-knowledge that the blindly signed message is indeed a commitment to some polynomial f with $f(0) = \mathsf{id}$, without revealing the commitment itself. As a result, the user obtains a pair $(com, \sigma_{\mathsf{BS}})$, which we designate as their pseudo-identity pid. Due to blindness, the group manager cannot link pseudo-identities appearing in signatures with any key generation session (and thus cannot connect them to any id). At the same time, unforgeability and the binding property of the underlying polynomial commitment scheme ensure that users cannot cheat by producing incorrect shares.

However, the user does not include the y-coordinate of the share or the opening in the signature in plaintext. Including this information would enable a trivial forgery; anyone observing more than k signatures with the same pid could reconstruct the user's secret polynomial f and easily forge a signature. To prevent this, the user encrypts the y-coordinate of the share using the group manager's public key and provides a non-interactive zero-knowledge proof that the resulting ciphertext indeed encrypts a y-coordinate of a share of the polynomial committed in com.

To summarize, during key generation the user commits to a polynomial f with $f(0) = \mathsf{id}$, which constitutes the user's secret key sk, by obtaining a blind signature σ_{BS} on a polynomial commitment com, without revealing the commitment itself. The pair $(com, \sigma_{\mathsf{BS}})$ then serves as the user's pseudo-identity pid. A signature on a message msg consists of the pseudo-identity pid, a half-encrypted Shamir share (x, ct) with $x = \mathsf{H2F}(\mathsf{pid} \,\|\, msg)$, and a proof that ct is encryption of $f(x)$ for f committed in com. The structure of a signature is schematically presented in Figure 1. This design ensures that signatures are linkable through the unforgeable pid, that shares are consistent and verifiable through the commitment, and that the signer's identity remains hidden from the group manager until the threshold k is exceeded and $f(0)$ can be reconstructed.

We present our k-AGS scheme in Section 3.2. Additionally, we briefly discuss some more subtle design choices, such as settling on subversion-resistant NIZKs for computing proofs, at the end of Section 3.2.

k-Anonymous Set Pre-Constrained Group Signatures. We present two constructions of k-ASPCGS. The first is a direct modification of our k-AGS scheme that achieves SPC security. The second is a generic transformation that converts any k-AGS into a k-ASPCGS.

Modification of k-AGS. The modification of our k-AGS is straightforward. Our goal is to ensure that signatures on "good" messages do not reveal information that could help the group manager trace the signer's identity. Recall that encrypting the y-coordinate of a share included in a signature is sufficient to prevent outsiders from reconstructing the secret polynomial f, and thus from identifying the signer. We can now

[3] For simplicitly, we focus on computing a share in a deterministic manner.

[4] We note that [BBM+21] does not consider malicious clients against traceability.

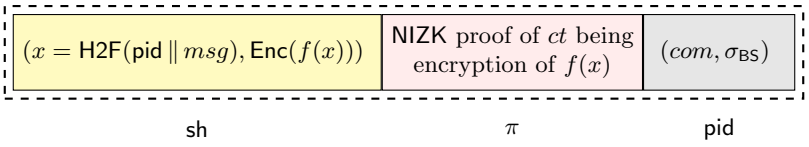

Figure 1: Schematic structure of a signature on msg in our k-AGS scheme.

achieve our goal by replacing public-key encryption with set pre-constrained encryption (SPCE). This approach is similar to the generic SPCGS framework of [BGJP22]: SPC encrypt identifying information—id in the case of [BGJP22], and a part of a share in ours.

The properties of SPCE guarantee that the group manager cannot extract y-coordinates from signatures—the only information, as previously argued, that would allow identity tracing in our k-AGS scheme. At the same time, the correctness of SPCE ensures that a user remains traceable if they sign more than k "bad" messages. We describe this modification in detail in Section 4.2.

Generic construction. In contrast to the pinpointed modification, the generic construction simply encrypts the entire k-AGS signature on the message and then proves that the ciphertext indeed contains a valid k-AGS signature. This design preserves the desired security properties in a modular way. The secrecy of the underlying SPCE scheme, combined with the zero-knowledge property of the NIZK, ensures that the hidden k-AGS signature cannot be learned by the server as long as the message is legitimate. Likewise, the k-anonymity of the k-AGS scheme directly carries over to the resulting k-ASPCGS. Traceability is also preserved: the correctness of the SPCE scheme, the knowledge soundness of the NIZK, and the traceability (and crucially, the statelessness) of the k-AGS together imply traceability of our construction.

Finally, the modularity of the approach makes it flexible enough to incorporate stronger properties in the future. For instance, if a stateless k-unlinkable group signature scheme (see discussion below) were available, then its unlinkability would automatically be inherited by the set pre-constrained version. Similarly, instantiating SPCE with PSI over authenticated inputs [BGJP25] would allow the group manager's set to be authenticated by a trusted third-party authority. We present our k-ASPCGS scheme in Section 4.2.

Security Notions. We introduce security notions for both k-AGS and k-ASPCGS, which are, broadly speaking, extensions of the security properties of group signatures and SPCGS.

In Section 3.1, we define the security properties of k-AGS, namely traceability and k-anonymity, following the conventional properties of group signatures [BMW03]. Traceability corresponds to the standard notion but in a threshold setting: an adversary should not be able to produce more than $k \cdot usr$ valid signatures—where usr is the number of users it registered with the group manager—without any identities being traced by the group manager. Analogous to the classic notion, traceability for k-AGS also encompasses unforgeability and unframeability, since the adversary is allowed to register honest users and obtain (more than k) signatures from them.

In contrast, k-anonymity is not a straightforward generalization. For conventional group signatures, anonymity is defined against everyone except the group manager, whereas in our case even the group manager should not be able to identify the signer until they sign $k+1$ messages. For k-AGS, we formalize this property using both game-based and ideal-functionality frameworks.

In the game-based definition, the group manager, after registering two users, is asked to distinguish whether given signatures were produced by the first or the second user, under the restriction of at most k signatures per user. In other words, the group manager should not be able to link key-generation sessions to the signatures produced by users. In the ideal-functionality setting, we capture the intuition that the group manager should not learn more from the signatures than whether they were produced by the same user.

When defining the ideal functionality, we focus on separating k-anonymity and traceability, i.e., our ideal functionality captures the intuition exclusively behind k-anonymity. For k-AGS, this allows us to show that the game-based definition implies the simulation-based one.

To conclude the discussion on k-anonymity, we note that we do not consider anonymity against parties

other than the group manager when more than k messages are signed. This is due to both the limits of reasonable paper length (we've tested those already) and the fact that this property is not central to our main application—the message receiver knows the sender's identity regardless. However, for our k-AGS scheme, anonymity against other parties should hold beyond k messages as well.

In Section 4.1, we define extensions of these security properties for the set pre-constrained case. For k-anonymity, we only consider a simulation-based definition, as this is the conventional approach when defining security for PSI-based constructions against a dishonest server. Our simulation-based definition extends the functionality of k-AGS by incorporating the functionality of SPCE [BGJP22].

1.3 Discussions

Other Related Work. k-anonymity in traceback systems (or variants thereof) has been extensively explored by the existing literature [TFS04, LRTY21, CLXZ25, TNL+25, BGH+25, ASY06, CHL06].

Linkable Ring Signatures. Choi, Liu, Xia and Zikas [CLXZ25] design *k-linkable ring signatures* (k-LRS) which allow a *stateless* user to sign anonymously at most k times, beyond which his signatures can be linked. The idea is to have the signer generate a fresh pseudo-identity pid as a linear combination of partial secret keys $x_1, \ldots, x_k$ and freshly sampled random weights $a_1, \ldots, a_k$ in the exponent w.r.t event-label e, and the signature also includes the weights. A link algorithm then solves linear equations over groups for every set of k signatures, with the overall complexity growing combinatorially. In a nutshell, the scheme is stateless and non-traceable while enforcing unlinkability, however only at the cost of an expensive link algorithm.

In [TNL+25], Tran, Nguyen, Liu, Pieprzyk and Susilo introduce a "weaker" variant of k-LRS, which instead of linking a group of $k+1$ signatures by a user, allows linking at least two signatures from a group of $k+1$ linked signatures. Their signing algorithm employs a PRF that is evaluated on a signing key and a counter that resets after k.

Bounded Anonymity E-cash. Camenisch, Hohenberger and Lysyanskaya [CHL06] propose a bounded-anonymity business model, where a *stateful* user can anonymously spend k e-coins per merchant, beyond which he is traceable by the merchant. The transactions are also unlinkable (as in [CLXZ25]) up to k spends per merchant.

Linkable Group Signatures. Au, Susilo and Yiu [ASY06] construct event-oriented linkable group signatures, where anonymity is event-specific. A signer remains untraceable and signatures are unlinkable (as in [CLXZ25]) until threshold k is reached. Nevertheless, the signer must maintain a counter specific to each event to track how many times he has signed.

Traceback Systems. In the setting of tracing originators of fake news, Liu, Roche, Theriault and Yerukhimovich [LRTY21] design a $(k+\epsilon)$-traceback system over an end-to-end messaging platform that traces the originator of a message reported by more than k users, where ϵ is an error bound. Their solution is to let the server maintain a counter whose status is updated obliviously by a user when reporting a message msg. After receiving more than $(1 \pm \epsilon)k$ reports on msg, the server can trace the originator. Note that such a system restricts the number of reports a user can submit, and *a traceback is triggered when the same message has been reported by more than k users.*

Along this line of work, Branco, Green, Hedge, Jain and Kaptchuk [BGH+25] propose a traceback system for misinformation that identifies the originator of a message which has traversed at least k hops (forwarded k-times), before getting reported.

These are worthy tasks, and we do not disagree with it. However, our k-AGS formalism applies to a different context, which we think is also relevant. In particular, we allow a server to trigger a traceback on an originator once any k (necessarily different) of his messages have been reported by his peers. Moreover, we do not enforce any restrictions on how many messages a user can report. Finally, we consider a stateless system, where the server and the users need not rely on counters. Our construction restrains a malicious user who reports the same signature k times—a share in a signature is computed as $x = \mathsf{H2F}(\mathsf{pid} \,\|\, msg)$, hence the shares will be identical for all the k reports. Since tracing requires k distinct shares, the sender cannot be traced.

On the Impossibility of Stateless k-unlinkable Group Signatures. We note that the existing k-anonymous schemes achieve unlinkability only at the cost of *either* requiring to maintain a state or having an expensive tracing algorithm, to the best of our knowledge. Elaborating on this, if the signer keeps no state, then all the necessary information required for tracing must be included in the signature. Let $\mathsf{pid} := f(\mathsf{sk}, \mathsf{r})$ denote the tracing information, derived from the signer's secret key sk, possibly combined with some randomness r that hides sk. Then, if the output of f is random each time (which ensures unlinkability), tracing would require brute-forcing, i.e. exploring all possible combinations of $k+1$ signatures from a pool of n signatures—which blows up exponentially. Hence, to allow efficient tracing, pid must include "some" deterministic component derived from the signer's secret key sk, such that even if the signature encodes some randomness r, the deterministic component can still be recovered. As a consequence, signatures become linkable prior to reaching the scheme anonymity parameter k.

For the clarity of exposition, we instantiate this deterministically (since com and σ_{BS} are uniquely determined during the key generation, pid is fixed for the signer) by setting $\mathsf{pid} = (com, \sigma_{BS})$, which is repeated in every signature generated by the signer. Hence, our k-AGS scheme is fully linkable via pid.
To our knowledge, it remains an open problem (which we believe we are the first to point out) whether there exists a stateless group signature scheme that achieves k-unlinkability and admits a tracing algorithm whose runtime scales sub-exponentially in both number of signatures n and the anonymity parameter k. Nevertheless, proving/disproving this is out of the scope of the current work, while we consider this to be an interesting topic for future study.

While full-linkability is a weakness, can it be leveraged as a feature in certain scenarios? For example, in moderation systems with overtly broad and relaxed rules—where first time or minor offenses are treated leniently—can linkability help track repeated reports against the same user without immediate tracing? We find some conceptual links with the report on UK Summer Riot 2024 [Gua25] that highlights the value of controlled linkability in aggregating repeated bad behavior and focusing on users who consistently generate concerning content. Nevertheless, given the limited empirical justification, we choose to remain conservative and refrain from leveraging linkability in our framework.

Avenues for Further Research. We now highlight several prospective avenues for future research. To begin, recall the previous discussion on our k-AGS construction being fully-linkable. This makes us think whether such limiting restrictions are inherently linked to the design of a stateless k-AGS scheme that is efficiently traceable/linkable. We consider it as an interesting open problem.

We attempted to "broaden" the moderation rules in content moderation systems where a single reported (bad) message can trace the sender. In particular, our threshold based anonymity approach allows tracing only after multiple reports have been submitted. Nevertheless, there could be other ways to incorporate context in the enforced rules. For instance, capturing the sender's intent (e.g. benign intent as in [New25]), the semantic properties of messages (e.g. literal meaning vs intended use, nuanced distinctions such as tone), temporal patterns in messages (e.g. if a user sends a "bad" message, but has years of normal, benign history), etc. We believe that exploring alternative approaches to mitigate overzealous content moderation is an important direction for future study.

We did not consider the scenario where the same (illegitimate) message from the sender is reported by k peers. If such a functionality were to be implemented, it may be handled at the application level, for instance by including receiver's identity, timestamp, etc. in the metadata of the message. Nevertheless, we do without it, and aim furthering this direction in future.

Given the set of illegitimate contents may increase over time, a straightforward approach as suggested in [BGJP22, BBM+21] is to re-run the system setup and publish the updated public information as a part of the system update. However, such methods could impede a moderator's ability to retrospectively check contents [HLZW24], where augmenting moderation rules require linking prior reported bad messages. Though full-linkability of k-AGS might be conceptually leveraged to achieve this, it is not a desirable solution as (a) it leaks relationship between past signatures, and (b) new messages added to $\mathcal{D}$ after the signatures are generated still require a way to determine which past messages match the updated set. This could be explored in future work.

Finally, as in [BGJP22, BGJP25], we let $\mathcal{D}$ denote a database of illegitimate contents, rather than relying

on perpetually hashed images as in real world deployments. Since, perpetual hashes do not inherit the full collision resistance guarantees of a standard cryptographic hash function, they introduce additional attack vectors, which are beyond the scope of this work. We consider investigating this as a meaningful direction for future work.

2 Preliminaries

2.1 Notations

We denote by $|S|$, the size of a finite set S. For any $n \in \mathbb{N}$, we consider $[n] \coloneqq \{1, \ldots, n\}$. We use $x \leftarrow y$ to assign x the value of y, and $x \leftarrow S$ to denote uniform sampling of x from the set S. For a probabilistic algorithm $\mathcal{A}$, we let $y \leftarrow \mathcal{A}(x)$ denote the process of running $\mathcal{A}(x)$ with fresh randomness and assigning the result to y. We write $(y; R) \leftarrow \mathcal{A}(x)$ when we want to store the random tape used by $\mathcal{A}$ in R, and $y \leftarrow \mathcal{A}(x; R)$ to denote running $\mathcal{A}$ with random tape R. We use $(x_1, \ldots, x_\ell) \Leftarrow y$ to parse a tuple y and assign its components to $x_1, \ldots, x_\ell$.

In our pseudocodes we use lists, denoted by L with a suitable subscript. To initialize a list we write $L \leftarrow \emptyset$. A list contains tuples $(x_1, \ldots, x_\ell)$, and we index a list by the first element of a tuple: $L[x_1]$ returns $(x_2, \ldots, x_\ell)$ if such a tuple exists in L, and $\emptyset$ otherwise. The expression $(x_1, \ldots, x_\ell) \in L$ (resp. $\notin L$) evaluates to 1 (resp. 0) if the tuple is present in the list, and 0 (resp. 1) otherwise. The construct **for** $x \in L$ iterates over all tuples in L. To add a tuple to a list we write $L \leftarrow L \cup \{(x_1, \ldots, x_\ell)\}$, and to update the value indexed by x with y we write $L[x] \leftarrow y$.

We let λ denote the security parameter. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be polynomial if there exists a constant c such that $f(n) \leq n^c$, for all $n \in \mathbb{N}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ is called negligible in n, if it grows slower than n^{-c}, for every constant c. We measure negligibility with respect to the security parameter λ. We write PPT to denote probabilistic polynomial-time algorithms. We model adversaries and simulators as non-uniform PPT algorithms.

2.2 Simulation-based security

To define some of the security notions, we adopt the Universal Composability (UC) framework [Can00]. For a functionality $\mathcal{F}$, an adversary $\mathcal{A}$, an environment $\mathcal{Z}$, and a protocol Π, we denote by

$$\mathsf{IDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(\lambda)$$

the random variable consisting of the output of $\mathcal{Z}$ in the *ideal execution* with $\mathcal{F}$, where $\mathcal{A}$ interacts with $\mathcal{Z}$ and corrupts parties as allowed. Similarly,

$$\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$$

denotes the output of $\mathcal{Z}$ in the *real execution*, where the parties execute protocol Π with adversary $\mathcal{A}$.

We say that Π *realizes* $\mathcal{F}$ if for every real-world adversary $\mathcal{A}$ there exists an ideal-world adversary (simulator) $\mathcal{S}$ such that for all environments $\mathcal{Z}$, random variables $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda)$ and $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ are computationally indistinguishable.

2.3 Public Key Encryption

We recall the standard syntax of PKE which we require to have indistinguishable encryptions under the chosen plaintext attack (INDCPA).

Definition 1 (Public Key Encryption) *We say that a triplet* $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *of PPT algorithms with a message space* $\mathcal{M}$ *is an* INDCPA *secure public key encryption (PKE) scheme if the following holds:*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$: *The key generation algorithm takes as input the security parameter* 1^λ*, and outputs a public key* pk *and a secret key* sk*.*

$\underline{ct \leftarrow \mathsf{Enc}(\mathsf{pk}, msg)}$: *The encryption algorithm takes as input the public key* pk*, and a message* $msg \in \mathcal{M}$*, and outputs a ciphertext* ct*.*

$\underline{msg \leftarrow \mathsf{Dec}(\mathsf{sk}, ct)}$: *The decryption algorithm takes as input the secret key* sk *and a ciphertext* ct*, and outputs a message* msg*.*

Furthermore, the algorithms must satisfy the following correctness and security properties.

Correctness: *For every message* $msg \in \mathcal{M}$*, the following holds:*

$$\Pr\left[msg = \mathsf{Dec}(\mathsf{sk}, ct) : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ ct \leftarrow \mathsf{Enc}(\mathsf{pk}, msg)\end{array}\right] = 1.$$

INDCPA **security**: *For a PPT adversary* $\mathcal{A}$*, its advantage against the* INDCPA *security of a public key encryption scheme* PKE *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(\mathcal{A}, n) = \Pr\left[\mathbf{Exp}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(1^n, \mathcal{A}) \to 1 \mid b = 0\right] - \Pr\left[\mathbf{Exp}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(1^n, \mathcal{A}) \to 1 \mid b = 1\right],$$

where experiment $\mathbf{Exp}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(1^n, \mathcal{A})$ *is defined on Figure 2.*
A PKE scheme PKE *is* INDCPA *secure if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$ *the following holds:*

$$\mathsf{Adv}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(\mathcal{A}, n) \leqslant \mathsf{negl}(\lambda).$$

$\mathbf{Exp}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(1^n, \mathcal{A})$

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PKE.KeyGen}(1^n)$
2 : $b \leftarrow \{0, 1\}$
3 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}}(\mathsf{pk})$
4 : **return** b'

Oracle $\mathcal{O}^{\mathsf{Enc}}(msg_0, msg_1)$

1 : $ct \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, msg_b)$
2 : **return** ct

Figure 2: Experiment $\mathbf{Exp}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(1^n, \mathcal{A})$

2.4 Blind Signatures

Definition 2 *A two-move blind signature scheme* BS *between a user* U *and a server* S *is a tuple*

$$\mathsf{BS} = (\mathsf{KeyGen}, \mathsf{U}_1, \mathsf{S}_1, \mathsf{U}_2, \mathsf{Verify})$$

of PPT algorithms defined as follows:

$\underline{(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^n)}$: *The key generation algorithm takes as input a security parameter* 1^n *and outputs a public key* pk *and a secret key* sk*.*

$\underline{(m_1, \mathsf{st}_\mathsf{U}) \leftarrow \mathsf{U}_1(\mathsf{pk}, msg)}$: *The first user algorithm takes as input a public key* pk *and a message* msg *to be signed, and outputs a message* m_1 *to be sent to the server and a state* st_U *to be kept locally.*

$\underline{m_2 \leftarrow \mathsf{S}_1(\mathsf{sk}, m_1)}$: *The server algorithm takes as input a secret key* sk *and a user's first message* m_1*, and outputs a response message* m_2*.*

$\sigma \leftarrow \mathsf{U}_2(\mathsf{st}_\mathsf{U}, m_2)$: *The second user algorithm takes as input a user's state* st_U *and a server's response* m_2*, and outputs a signature* σ *on the message* msg*.*

$b \leftarrow \mathsf{Verify}(\mathsf{pk}, msg, \sigma)$: *The verification algorithm takes as input a public key* pk*, a message* msg*, and a candidate signature* σ*, and outputs* $b = 1$ *if* σ *is a valid signature on* msg*, and* $b = 0$ *otherwise.*

We recall the following security properties of blind signatures.

Blindness: *For a PPT adversary* $\mathcal{A}$*, its advantage against the blindness of a blind signature scheme* BS *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{Blind}}_{\mathsf{BS}}(\mathcal{A}, \lambda) = \Pr[\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}(1^\lambda, \mathcal{A}) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}(1^\lambda, \mathcal{A}) \to 1 \mid b = 1],$$

where the blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}$ *is defined in Figure 3.*
A BS scheme BS *is blind if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$ *the following holds:*

$$\mathsf{Adv}^{\mathrm{Blind}}_{\mathsf{BS}}(\mathcal{A}, \lambda) \leqslant \mathsf{negl}(\lambda).$$

$\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}(1^\lambda, \mathcal{A})$

1 : $(\mathsf{pk}, msg_0, msg_1, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(1^n)$
2 : $\mathsf{st}_0, \mathsf{st}_1 \leftarrow \emptyset$
3 : $sess_0, sess_1 \leftarrow \mathsf{init}$
4 : $\sigma_0, \sigma_1 \leftarrow \bot$
5 : $b \leftarrow \{0,1\}$
6 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{U}_1}, \mathcal{O}^{\mathsf{U}_2}}(\mathsf{st}_\mathcal{A})$
7 : **return** b'

Oracle $\mathcal{O}^{\mathsf{U}_1}(i)$

1 : **if** $i \notin \{0,1\} \vee sess_i \neq \mathsf{init}$:
2 : **return** $\bot$
3 : $sess_i \leftarrow \mathsf{open}$
4 : $(\mathsf{st}_i, umsg) \leftarrow \mathsf{U}_1(\mathsf{pk}, msg_{i \oplus b})$
5 : **return** $umsg$

Oracle $\mathcal{O}^{\mathsf{U}_2}(i, smsg)$

1 : **if** $i \notin \{0,1\} \vee sess_i \neq \mathsf{open}$:
2 : **return** $\bot$
3 : $sess_i \leftarrow \mathsf{closed}$
4 : $\sigma_{i \oplus b} \leftarrow \mathsf{U}_2(\mathsf{st}_i, smsg)$
5 : **if** $sess_0 = sess_1 = \mathsf{closed}$:
6 : **if** $\sigma_0 = \bot \vee \sigma_1 = \bot$:
7 : **return** (σ_0, σ_1)
8 : **return** (i, closed)

Figure 3: Blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}$

Remark 1 For convenience, in addition to the presented definition, we will also make use of an n-message definition of the experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}$, in which the adversary, upon initialization, outputs a permutation π on $\{0, \ldots, n-1\}$ along with other values, and is allowed to output n messages. If bit $b = 1$, when $\mathcal{A}$ queries to sign i-th message, message $msg_{\pi(i)}$ is signed instead. It is easy to see that, since every permutation can be represented as a product of transpositions, the n-message security follows from the 2-message case via a hybrid argument.

One-More Unforgeability: *For a PPT adversary $\mathcal{A}$, its advantage against the one-more unforgeability of a blind signature scheme* BS *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{omUnforg}}_{\mathsf{BS}}(\mathcal{A},\lambda) = \Pr[\mathbf{Exp}^{\mathrm{omUnforg}}_{\mathsf{BS}}(1^\lambda,\mathcal{A}) \to 1],$$

where the one-more unforgeability experiment $\mathbf{Exp}^{\mathrm{omUnforg}}_{\mathsf{BS}}$ *is defined in Figure 4.*
A BS scheme BS *is one-more unforgeable if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$ *the following holds:*

$$\mathsf{Adv}^{\mathrm{omUnforg}}_{\mathsf{BS}}(\mathcal{A},\lambda) \leqslant \mathsf{negl}(\lambda).$$

$\mathbf{Exp}^{\mathrm{Blind}}_{\mathrm{omUnforg}}(1^\lambda,\mathcal{A})$

1 : $count \leftarrow 0$
2 : $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^n)$
3 : $(s,\{(msg_i,\sigma_i)\}_{i=1,\ldots,s\}}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{S}_1}}(1^n,\mathsf{pk})$
4 : **if** $s \leq count$:
5 : **return** 0
6 : **if** $\exists(i,j)$: $(msg_i,\sigma_i) = (msg_j,\sigma_j)$:
7 : **return** 0
8 : **for** $i = 1,\ldots,s$:
9 : **if** $\mathsf{Verify}(\mathsf{pk},msg_i,\sigma_i) = 0$:
10 : **return** 0
11 : **return** 1

Oracle $\mathcal{O}^{\mathsf{S}_1}(m)$

1 : $count \leftarrow count + 1$
2 : $m_{out} \leftarrow \mathsf{S}_1(\mathsf{sk},m)$
3 : **return** m_{out}

Figure 4: One-More Unforgeability Experiment $\mathbf{Exp}^{\mathrm{omUnforg}}_{\mathsf{BS}}$

2.5 Polynomial Commitments

Definition 3 *A polynomial commitment scheme* PC *is a tuple* $\mathsf{PC} = (\mathsf{Setup},\mathsf{Commit},\mathsf{Open},\mathsf{Verify})$ *of PPT algorithms defined as follows:*

$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda,d)$: *The setup algorithm takes as input a security parameter* 1^λ *and a maximum degree bound* d*, and outputs a commitment key* ck*.*

$com \leftarrow \mathsf{Commit}(\mathsf{ck},f)$: *The* deterministic *commitment algorithm takes as input a commitment key* ck *and a polynomial* $f \in \mathbb{Z}_p[x]$ *of degree at most* d*, and outputs a commitment* com*.*

$\omega \leftarrow \mathsf{Open}(\mathsf{ck},f,x,y)$: *The opening algorithm takes as input a commitment key* ck*, a committed polynomial* f*, and points* $x,y \in \mathbb{Z}_p$*, and outputs an opening* ω*.*

$b \leftarrow \mathsf{Verify}(\mathsf{ck},com,x,y,\omega)$: *The verification algorithm takes as input a commitment key* ck*, a commitment* com*, points* x,y*, and a proof* π*, and outputs a bit* b*.*

We recall the following security properties of polynomial commitments.

Binding: *A PC scheme* PC *is binding if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$ *and for every* $d, \lambda \in \mathbb{N}$ *the following holds:*

$$\Pr\left[\begin{matrix} y_1 \neq y_2 \\ \wedge \mathsf{Verify}(\mathsf{ck}, com, x, y_1, \omega_1) : \\ \wedge \mathsf{Verify}(\mathsf{ck}, com, x, y_2, \omega_2) \end{matrix} \begin{matrix} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (com, x, y_1, y_2, \omega_1, \omega_2) \leftarrow \mathcal{A}(\mathsf{pk}) \end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

Hiding: *For a PPT adversary* $\mathcal{A}$*, its advantage against the hiding property of a polynomial commitment scheme* PC *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{Hiding}}_{\mathsf{PC}}(\mathcal{A}, \lambda) = \Pr[\mathbf{Exp}^{\mathrm{Hiding}}_{\mathsf{PC}}(1^\lambda, \mathcal{A}) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{Hiding}}_{\mathsf{PC}}(1^\lambda, \mathcal{A}) \to 1 \mid b = 1],$$

where the hiding experiment $\mathbf{Exp}^{\mathrm{Hiding}}_{\mathsf{PC}}$ *is defined in Figure 5.*
A PC scheme PC *is hiding if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$ *the following holds:*

$$\mathsf{Adv}^{\mathrm{Hiding}}_{\mathsf{PC}}(\mathcal{A}, \lambda) \leqslant \mathsf{negl}(\lambda).$$

$\mathbf{Exp}^{\mathrm{Hiding}}_{\mathsf{PC}}(1^\lambda, \mathcal{A})$

$(\mathsf{ck}, d, (x_0, y_0), \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda)$
$b \leftarrow \{0, 1\}$
$count \leftarrow 0$
if $b = 0$:
  $f \leftarrow \{f \in \mathbb{Z}_p[x] : deg(f) = d \wedge f(x_0) = y_0\}$
else :
  $f \leftarrow \{f \in \mathbb{Z}_p[x] : deg(f) = d\}$
$com \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, f)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Open}}}(\mathsf{st}, com)$
return b'

Oracle $\mathcal{O}^{\mathsf{Open}}(x)$

if $count = d \vee x = x_0$
  return $\perp$
$y \leftarrow f(x)$
$\omega \leftarrow \mathsf{PC.Open}(\mathsf{ck}, f, x, y)$
$count \leftarrow count + 1$
return $(\omega, (x, y))$

Figure 5: Hiding experiment $\mathbf{Exp}^{\mathrm{Hiding}}_{\mathsf{PC}}$

2.6 Subversion-Resistant Non-Interactive Zero-Knowledge

In this section, we recall definition of subversion-resistant non-interactive zero-knowledge (NIZK) schemes [BFS16, ALSZ21]. These are NIZK schemes that maintain the zero-knowledge property even when the common reference string (CRS) is generated by an adversary, rather than by a trusted setup. In other words, even if the CRS is maliciously crafted, the adversary should not learn any additional information from the proof beyond its validity. For example, slight modifications of Groth16 were proven to be subversion zero-knowledge [ALSZ21, Fuc18]. Along with conventional knowledge-soundness, we will use no less conventional, but stronger definition of simulation-extractability.

Let $\mathsf{RelGen}(1^n)$ be a relation generator that outputs an NP-relation $\mathcal{R}(x, w) \to \{0, 1\}$, where x is a statement and w is a witness. We denote the corresponding NP language by $\mathcal{L}_{\mathcal{R}}$. For convenience, we also write $(x, w) \in \mathcal{R}$ for pairs (x, w) such that $\mathcal{R}(x, w) = 1$.

Definition 4 *A subversion-resistant non-interactive zero-knowledge (NIZK) argument system* NIZK *for a relation generator* RelGen *is a tuple* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{CVerify}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *of PPT algorithms defined as follows:*

$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$: *The setup algorithm takes as input a security parameter* 1^λ *and a relation* $\mathcal{R}$ *and outputs a common reference string* crs *and a simulation trapdoor* td.

$b \leftarrow \mathsf{CVerify}(\mathcal{R}, \mathsf{crs})$: *The CRS verification algorithm takes as input a relation* $\mathcal{R}$ *and a CRS* crs*, and outputs* $b = 1$ *if* crs *is well-formed, and* $b = 0$ *otherwise.*

$\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$: *The prove algorithm takes as input a CRS* crs*, a statement–witness pair* $(x, w) \in \mathcal{R}$*, and outputs a proof* π.

$b \leftarrow \mathsf{Verify}(\mathsf{crs}, x, \pi)$: *The verify algorithm takes as input a CRS* crs*, a statement* x*, and a proof* π*, and outputs and outputs* $b = 1$ *if the proof is valid, and* $b = 0$ *otherwise.*

$\pi \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x)$: *The simulator algorithm takes as input a CRS* crs*, a simulation trapdoor* td*, and a statement* x*, and outputs a simulated proof* π.

Subversion Zero-Knowledge: *Let* $\mathcal{S}$ *be a PPT subverter and* $\mathcal{E}_\mathcal{S}$ *be a PPT simulation trapdoor extractor. Let* NIZK *be a NIZK scheme for* $\mathsf{RelGen}(1^n)$ *and* $\mathcal{R}$ *some relation from* $\mathsf{RelGen}(1^n)$*. For a pair of the subverter and a PPT adversary* $(\mathcal{S}, \mathcal{A})$*, an advantage against the subversion zero-knowledge of* NIZK*, the extractor* $\mathcal{E}_\mathcal{S}$ *and relation* $\mathcal{R}$ *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{S},\mathcal{R}}((\mathcal{S},\mathcal{A}),\lambda) = \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{S},\mathcal{R}}(1^\lambda,(\mathcal{S},\mathcal{A})) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{S},R}(1^\lambda,(\mathcal{S},\mathcal{A})) \to 1 \mid b = 1],$$

where the subversion zero-knowledge experiment $\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{S}}$ *is defined in Figure 6.*
A NIZK scheme NIZK *is subversion zero-knowledge (Sub-ZK) if there exists a negligible function* $\mathsf{negl}(\lambda)$ *and for every subverter* $\mathcal{S}$ *there exists a PPT simulation trapdoor extractor* $\mathcal{E}_\mathcal{S}$ *such that for every* $\lambda \in \mathbb{N}$*, every* $\mathcal{R} \in \mathsf{RelGen}(1^\lambda)$ *and every PPT adversary* $\mathcal{A}$*, the following holds:*

$$\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{S},\mathcal{R}}((\mathcal{S},\mathcal{A}),\lambda) \leqslant \mathsf{negl}(\lambda).$$

Knowledge Soundness: *Let* NIZK *be a NIZK scheme for* $\mathsf{RelGen}(1^n)$ *and* $\mathcal{R}$ *some relation from* $\mathsf{RelGen}(1^n)$*. For an adversary* $\mathcal{A}$*, an advantage against the knowledge soundness of a NIZK scheme* NIZK*, a PPT extractor* $\mathcal{E}_\mathcal{A}$ *and a relation* $\mathcal{R}$ *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{knSound}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{A},\mathcal{R}}(\mathcal{A},\lambda) = \Pr[\mathbf{Exp}^{\mathrm{knSound}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{A},\mathcal{R}}(1^\lambda,\mathcal{A}) \to 1],$$

where the knowledge soundness experiment $\mathbf{Exp}^{\mathrm{knSound}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{A},\mathcal{R}}$ *is defined in Figure 6.*
A NIZK scheme NIZK *is knowledge-sound if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for PPT every adversary* $\mathcal{A}$ *there exists a PPT extractor* $\mathcal{E}_\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$*, every* $\mathcal{R} \in \mathsf{RelGen}(1^\lambda)$*, the following holds:*

$$\mathsf{Adv}^{\mathrm{knSound}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{A},\mathcal{R}}(\mathcal{A},\lambda) \leqslant \mathsf{negl}(\lambda).$$

Simulation-Extractability: *We call a* NIZK *scheme simulation-extractable if the above definition of knowledge-soundness continues to hold even when* $\mathcal{A}$ *is additionally given oracle access to* $\mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x)$ *for statements* x *of its choice, provided that* $\mathcal{A}$ *is restricted from outputting proofs for statements it queried to* Sim*. We denote corresponding experiment by* $\mathbf{Exp}^{\mathrm{SE}}_{\mathsf{NIZK},\mathcal{E}_\mathcal{A},\mathcal{R}}$.

$\mathbf{Exp}^{\text{knSound}}_{\mathsf{NIZK},\mathcal{E}_{\mathcal{A}},\mathcal{R}}(1^\lambda,\mathcal{A})$

$(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{R})$

$(x,\pi;R) \leftarrow \mathcal{A}(1^\lambda,\mathsf{crs})$

$w \leftarrow \mathcal{E}_{\mathcal{A}}(1^\lambda,\mathsf{crs},\mathsf{td},R)$

if $(x,w) \notin \mathcal{R} \wedge \mathsf{NIZK.Verify}(\mathsf{crs},x,\pi) = 1$

  return 1

return 0

$\mathbf{Exp}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{E}_{\mathcal{S}},\mathcal{R}}(1^\lambda,(\mathcal{S},\mathcal{A}))$

$b \leftarrow \{0,1\}$

$(\mathsf{crs},\mathsf{st};R) \leftarrow \mathcal{S}(1^\lambda,\mathcal{R})$

if $\mathsf{CVerify}(\mathcal{R},\mathsf{crs}) = 0$: **return** 0

$\mathsf{td} \leftarrow \mathcal{E}_{\mathcal{S}}(1^\lambda,\mathcal{R},R)$

$(x,w,\mathsf{st}) \leftarrow \mathcal{A}(1^\lambda,\mathsf{td},\mathsf{st})$

if $(x,w) \notin \mathcal{R}$: **return** 0

if $b = 0$: $\pi \leftarrow \mathsf{Prove}(\mathsf{crs},x,w)$

if $b = 1$: $\pi \leftarrow \mathsf{Sim}(\mathsf{crs},\mathsf{td},x)$

$b' \leftarrow \mathcal{A}(\mathsf{st},\pi)$

return b'

Figure 6: Experiments $\mathbf{Exp}^{\text{knSound}}_{\mathsf{NIZK},\mathcal{E}_{\mathcal{A}},\mathcal{R}}$ and $\mathbf{Exp}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{E}_{\mathcal{S}},\mathcal{R}}$

2.7 Set Pre-Constrained Encryption

We now present the background definition of set pre-constrained encryption from the literature, along with its correctness and security notions. We follow the formalization by [BGJP22, BGJP25], as it more suited to our constructions.

Definition 5 (Set Pre-Constrained Encryption) *A set pre-constrained encryption scheme (SPCE)* Π_{SPCE} *over a universe* U *of items and a message space* $\mathcal{M}$ *is a tuple of PPT algorithms*

$$\Pi_{\mathsf{SPCE}} = (\mathsf{Gen},\mathsf{Enc},\mathsf{Dec}),$$

parameterized by a maximum set size n*. It is defined as follows:*

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda,\mathcal{D})$: *The key-generation algorithm takes as input the security parameter* 1^λ *and a set* $\mathcal{D} \subseteq \mathsf{U}$ *of size at most* n*, and outputs a public key* pk *and a secret key* sk*.*

$ct \leftarrow \mathsf{Enc}(\mathsf{pk},x,msg)$: *The encryption algorithm takes as input a public key* pk*, an item* $x \in \mathsf{U}$*, and a message* $msg \in \mathcal{M}$*, and outputs a ciphertext* ct*.*

$msg' \leftarrow \mathsf{Dec}(\mathsf{sk},ct)$: *The decryption algorithm takes as input a secret key* sk *and a ciphertext* ct*, and outputs either a message* $msg' \in \mathcal{M}$ *or the failure symbol* $\perp$*.*

We note that, to achieve the notion of authenticated-set security defined below, the definition can be relaxed so that Gen is an interactive protocol between the server and a third party Auth that 'authenticates' the set used.

Perfect Correctness: *An SPCE scheme is said to be perfectly correct if for every* $\lambda \in \mathbb{N}$ *and* $\mathcal{D} \subseteq \mathsf{U}, |\mathcal{D}| \leq n$*, there exists a negligible function* $\mathsf{negl}(\lambda)$*, such that for all* $msg \in \mathcal{M}, x \in \mathcal{D}$ *and any random tape* R *it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},ct) \neq msg : \begin{matrix}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda,\mathcal{D}) \\ ct \leftarrow \mathsf{Enc}(\mathsf{pk},x,msg;R)\end{matrix}\right] \leq \mathsf{negl}(\lambda),$$

and for all $msg \in \mathcal{M}, x \notin \mathcal{D}$ *and any random tape* R *it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},ct) \neq \{msg,\perp\} : \begin{matrix}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda,\mathcal{D}) \\ ct \leftarrow \mathsf{Enc}(\mathsf{pk},x,msg;R)\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

Simulation-based Security: We formalize security of SPCE in the simulation-based framework, following the definitions of Bartusek et al. [BGJP22] with minor stylistic modifications to match our presentation. The security notion is captured via the ideal functionality $\mathcal{F}_{\mathsf{SPCE}}$ (Figure 7).

Functionality $\mathcal{F}_{\mathsf{SPCE}}$

On input $(\mathsf{pp}, \mathsf{sp})$ from Auth:

Send pp to C and $(\mathsf{pp}, \mathsf{sp})$ to $\mathcal{S}$

On input $\mathcal{D}$ from S:

if $\mathcal{P}(\mathsf{pp}, \mathsf{sp}, \mathcal{D}) = 0$:

Send $\perp$ to $\mathsf{Auth}, \mathsf{GM}, \mathsf{C}$ and halt

On input (id, msg) from C:

if $x \in \mathcal{D}$:

Send (x, msg) to GM and x to Auth

else :

Send x to GM and Auth

Figure 7: SPC Encryption Functionality $\mathcal{F}_{\mathsf{SPCE}}$

The functionality involves three parties: a server S (running Gen and Dec), a client C (running Enc), and an authority Auth, who initializes the system with public and secret parameters $(\mathsf{pp}, \mathsf{sp})$. The predicate $\mathcal{P}(\mathsf{pp}, \mathsf{sp}, \mathcal{D})$ enforces validity of the server's input set $\mathcal{D}$. We distinguish two instantiations of the predicate $\mathcal{P}$:

- *Bounded-set security:* here $\mathsf{pp} = n$, $\mathsf{sp} = \emptyset$, and $\mathcal{P}(\mathsf{pp}, \mathsf{sp}, \mathcal{D}) = 1$ if and only if $|\mathcal{D}| \leq n$.
- *Authenticated-set security:* here $\mathsf{pp} = |\mathcal{D}^*|$, $\mathsf{sp} = \mathcal{D}^*$ for a hidden reference set $\mathcal{D}^*$, and $\mathcal{P}(\mathsf{pp}, \mathsf{sp}, \mathcal{D}) = 1$ if and only if $\mathcal{D} \subseteq \mathcal{D}^*$.

Both real and ideal executions are carried out in the presence of an environment $\mathcal{Z}$ that provides all inputs and receives all outputs. We allow the adversary to corrupt at most one party among the server, client, and Auth.

- If the server is corrupted, it may attempt to learn more information about client messages than what is allowed by $\mathcal{F}_{\mathsf{SPCE}}$.
- If the client is corrupted, it may attempt to deviate from Enc, but it should not learn anything about the server's set $\mathcal{D}$.
- If Auth is corrupted, it may attempt to learn client messages, but it should remain unable to do so beyond what $\mathcal{F}_{\mathsf{SPCE}}$ specifies.

Outsider INDCPA: *For a PPT adversary $\mathcal{A}$, its advantage against the* oINDCPA *security of a set pre-constrained encryption scheme* SPCE *is defined as follows:*

$$\mathsf{Adv}^{\mathrm{oINDCPA},\mathcal{D}}_{\mathsf{SPCE}}(\mathcal{A}, n) = \Pr\left[\mathbf{Exp}^{\mathrm{oINDCPA}}_{\mathsf{SPCE},\mathcal{D}}(1^n, \mathcal{A}) \to 1 \mid b = 0\right] - \Pr\left[\mathbf{Exp}^{\mathrm{oINDCPA}}_{\mathsf{SPCE},\mathcal{D}}(1^n, \mathcal{A}) \to 1 \mid b = 1\right],$$

where experiment $\mathbf{Exp}^{\mathrm{oINDCPA}}_{\mathsf{SPCE}}(1^n, \mathcal{A})$ *is defined on Figure 8.*
An SPCE scheme SPCE *is* oINDCPA *secure if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every PPT adversary $\mathcal{A}$ and for every $\lambda \in \mathbb{N}$, $\mathcal{D} \subseteq \mathcal{M}, |\mathcal{D}| \leq n$ the following holds:*

$$\mathsf{Adv}^{\mathrm{oINDCPA}}_{\mathsf{SPCE}}(\mathcal{A}, n) \leqslant \mathsf{negl}(\lambda).$$

$\mathbf{Exp}^{\text{oINDCPA},\mathcal{D}}_{\mathsf{SPCE}}(1^n,\mathcal{A})$

1 : $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{SPCE.Gen}(1^n)$
2 : $b \leftarrow \{0,1\}$
3 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}}(\mathsf{pk})$
4 : **return** b'

Oracle $\mathcal{O}^{\mathsf{Enc}}((x_0 msg_0),(x_1,msg_1))$

1 : $ct \leftarrow \mathsf{SPCE.Enc}(\mathsf{pk},x_b,msg_b)$
2 : **return** ct

Figure 8: Experiment $\mathbf{Exp}^{\text{oINDCPA},\mathcal{D}}_{\mathsf{SPCE}}(1^n,\mathcal{A})$

2.8 Hash to Field

Definition 6 *A hash-to-field function for a prime field $\mathbb{Z}_p$ is a family of functions $\mathcal{HF} = \{\mathsf{H2F}_s : \mathcal{M} \to \mathbb{Z}_p\}_{s\in\{0,1\}^\lambda}$, where each function takes as input an arbitrary message $m \in \mathcal{M}$, and outputs an element $y \in \mathbb{Z}_p$.*

Collision Resistance: *A function family $\mathcal{HF}$ is said to be* collision-resistant *if there exist a negligible function* $\mathsf{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$ and for every PPT adversary $\mathcal{A}$ the following holds:*

$$\Pr[s \leftarrow \{0,1\}^\lambda, (x_1,x_2) \leftarrow \mathcal{A}(s) : x_1 \neq x_2 \ \wedge\ \mathsf{H2F}_s(x_1) = \mathsf{H2F}_s(x_2)] \leqslant \mathsf{negl}(\lambda).$$

3 k-Anonymous Group Signatures

3.1 Definition and Security Notions

We now define k-AGS along with its correctness and security notions.

Definition 7 *A k-Anonymous Group Signature (k-AGS) scheme $\Pi[k,\mathcal{M},\mathcal{ID}]$ with a message space $\mathcal{M}$, an identity space $\mathcal{ID}$ is a tuple* $(\mathsf{Gen},\mathsf{Sign},\mathsf{Verify},\mathsf{Trace})$ *of PPT algorithms along with an interactive protocol* KeyGen *defined as follows:*

$(\mathsf{mpk},\mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda,k)$: *The master key generation algorithm takes as input a security parameter 1^λ, and the scheme anonymity parameter k, and outputs a master public key* mpk *and a master secret key* msk.

$\langle \mathsf{id},(\mathsf{id},\mathsf{sk})\rangle \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}),\mathsf{C}\rangle$: *The interactive key generation protocol is run by a group manager* GM *with an input* msk*, and by a client* C*. It delivers an identity* $\mathsf{id} \in \mathcal{ID}$ *to both* GM *and* C*, and an identity signing key* sk *to* C*. The master public key* mpk *is an implicit input for both parties.*

$\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk},\mathsf{sk},msg)$: *The signing algorithm takes as input a master public key* mpk*, an identity signing key* sk*, and a message $msg \in \mathcal{M}$, and outputs a signature σ.*

$\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{mpk},msg,\sigma)$: *The signature verification algorithm takes as input a master public key* mpk*, a message $msg \in \mathcal{M}$, and a signature σ, and outputs either 1 or 0, indicating accept or reject.*

$\{\mathsf{id}_1,\ldots,\mathsf{id}_m\} \leftarrow \mathsf{Trace}(\mathsf{msk},k,\{\sigma_i\}_{i\in\mathcal{I}})$: *The identity reconstruction algorithm takes as input a master secret key* msk*, the scheme anonymity parameter k, a set of signatures $\{\sigma_i\}_{i\in\mathcal{I}}$, and outputs a potentially empty set of corresponding identities $\{\mathsf{id}_1,\ldots,\mathsf{id}_m\}, \mathsf{id}_i \in \mathcal{ID}$.*

Key Generation Protocol: The key generation protocol is defined by a tuple of PPT algorithms

$$(\mathsf{GM}_1,\mathsf{C}_1,\mathsf{GM}_2,\mathsf{C}_2,\ldots,\mathsf{GM}_i,\mathsf{C}_i).$$

We consider the first message in the protocol to be sent by the group manager. GM_1 takes as input the group manager's protocol input and outputs a message to be sent to the client along with the group manager's

state. C_1 takes the client's protocol input and the first message, and outputs a response message together with the client's state. Each subsequent algorithm takes as input a party's state and the received message, and outputs an updated state and a new message. The final algorithms output, within the state, the values designated for each party at the end of the protocol, and additionally include the keyword accept. If any algorithm fails, it outputs the symbol $\perp$ for each of its outputs.

Correctness: *For every $\lambda \in \mathbb{N}$, every $k = k(\lambda)$, and every $msg \in \mathcal{M}$, there exists a negligible function* $\mathsf{negl}(\lambda)$*, such that*

$$\Pr\left[\mathsf{Verify}(\mathsf{mpk}, msg, \sigma) \neq 1 : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, k) \\ \langle(\mathsf{id}, \mathsf{sk}), \mathsf{id}\rangle \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}), \mathsf{C}\rangle \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where probability is taken over the random coin tosses of Gen, KeyGen *and* Sign.

Security. We now regard the security notions of a k-AGS scheme. We first define *traceability* which ensures that any user who signs more than k messages that are accepted by the Verify algorithm, can be traced by the master key holder.

Traceability: *For a PPT adversary $\mathcal{A}$, let its advantage against traceability of a k-AGS scheme Π be defined as follows:*

$$\mathsf{Adv}^{\mathrm{Trace}}_{\Pi}(\mathcal{A}, \lambda) = \Pr[\mathbf{Exp}^{\mathrm{Trace}}_{\Pi}(1^\lambda, \mathcal{A}) \to 1],$$

where the traceability experiment $\mathbf{Exp}^{\Pi}_{\mathrm{Trace}}$ is defined in Figure 9.

A k-AGS scheme Π is traceable if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, the following holds:

$$\mathsf{Adv}^{\mathrm{Trace}}_{\Pi}(\mathcal{A}, \lambda) \leqslant \mathsf{negl}(\lambda).$$

Next, we formulate the notion of *k-anonymity* which says that no adversary can deanonymize a user who has signed less than $k+1$ messages. We present both game-based and simulation based definitions of k-anonymity. The latter ensures a smoother compatibility with more complex multiparty protocols.

k-Anonymity (Game-Based): *For a PPT adversary $\mathcal{A}$, let its advantage against k-anonymity of a k-AGS scheme Π be defined as follows:*

$$\mathsf{Adv}^{\mathrm{kAnon}}_{\Pi}(\mathcal{A}, \lambda) = \Pr[\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}(1^\lambda, \mathcal{A}) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}(1^\lambda, \mathcal{A}) \to 1 \mid b = 1],$$

where the k-anonymity experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}$ is defined in Figure 10.

The k-AGS scheme Π is k-anonymous if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, the following holds:

$$\mathsf{Adv}^{\mathrm{kAnon}}_{\Pi}(\mathcal{A}, \lambda) \leqslant \mathsf{negl}(\lambda).$$

The idea behind the k-anonymity experiment is to consider two users. An adversary $\mathcal{A}$ can collude with the GM and all the users except the target users. $\mathcal{A}$ determines the threshold k, along with master public key mpk and state st. When $\mathcal{A}$ queries the Sign oracle with a message msg, the oracle signs msg on behalf of one of the target users, but does not disclose its choice. For a target user, $\mathcal{A}$ is only allowed to query the oracle within k times. Finally, $\mathcal{A}$ wins if she could make the correct guess.

Remark 2 For convenience, in addition to the presented definition, we will also make use of an n-user definition of the experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}$, in which the adversary, upon initialization, outputs a permutation π on $\{0, \ldots, n-1\}$ along with other values, and is allowed to register n users. If bit $b = 1$, to sign a message for user i, the secret key $\mathsf{sk}_{\pi(i)}$ is used. It is easy to see that, since every permutation can be represented as a product of transpositions, the n-user security follows from the 2-user case via a hybrid argument.

$\mathbf{Exp}^{\text{Trace}}_{\Pi[k,\mathcal{M},\mathcal{ID}]}(1^\lambda, \mathcal{A})$

$L_{\mathsf{id}}, L_{\mathsf{st}}, L_{\mathsf{sk}}, L_{msg} \leftarrow \emptyset$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, k)$
$(s, \{(msg_i, \sigma_i)\}_{i=1,\ldots,s}\}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen},\ldots}}(1^\lambda, \mathsf{mpk})$
if $\exists (i,j)\colon msg_i \neq msg_j \vee msg_i \in L_{msg}$:
  return 0
for $i = 1, \ldots, s$:
  if $\mathsf{Verify}(\mathsf{mpk}, msg_i, \sigma_i) = 0$:
    return 0
// Check if the adversary abused the system
if $s \leqslant k \cdot |L_{\mathsf{id}}|$: **return** 0
// Trace identities
$\{\mathsf{id}_1, \ldots, \mathsf{id}_m\} \leftarrow \mathsf{Trace}(\mathsf{msk}, k, \{\sigma_1, \ldots, \sigma_s\})$
// If there is at least one id that belongs
// to the adversary, they lose
if $\{\mathsf{id}_1, \ldots, \mathsf{id}_m\} \cap L_{\mathsf{id}} \neq \emptyset$:
  return 0
// Else they succeeded in hiding their identity
return 1

Oracle $\mathcal{O}^{\mathsf{HonestKG}}()$

// Oracle for $\mathcal{A}$ to register honest users
$(\mathsf{id}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}), \mathsf{C}\rangle$
$\mathcal{L}_{\mathsf{sk}} \leftarrow (\mathsf{id}, \mathsf{sk})$
return id

Oracle $\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{sid}, msg_{in})$

// An interface for $\mathcal{A}$ to run
// the KeyGen protocol with GM
if $L_{\mathsf{st}}[\mathsf{sid}] = \emptyset$:
  $(step, \mathsf{st}) \leftarrow (1, (\mathsf{msk}, \mathsf{mpk}))$
else :
  $(step, \mathsf{st}) \leftarrow L_{\mathsf{st}}[\mathsf{sid}]$
$(\mathsf{st}, msg_{out}) \leftarrow \mathsf{KeyGen.GM}_{step}(\mathsf{st}, msg_{in})$
$L_{\mathsf{st}}[\mathsf{sid}] \leftarrow (step + 1, \mathsf{st})$
if $\mathsf{accept} \in \mathsf{st}$:
  $L_{\mathsf{id}} \leftarrow L_{\mathsf{id}} \cup \mathsf{st.id}$
return msg_{out}

Oracle $\mathcal{O}^{\mathsf{Sign}}(\mathsf{id}, msg)$

// An oracle for $\mathcal{A}$ to get
// signatures from honest users
$L_{msg} \leftarrow L_{msg} \cup msg$
if $L_{\mathsf{sk}}[\mathsf{id}] = \emptyset$:
  return 0
$\mathsf{sk} \leftarrow L_{\mathsf{sk}}[\mathsf{id}]$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg)$
return σ

Figure 9: Traceability Experiment $\mathbf{Exp}^{\text{Trace}}_{\Pi}$

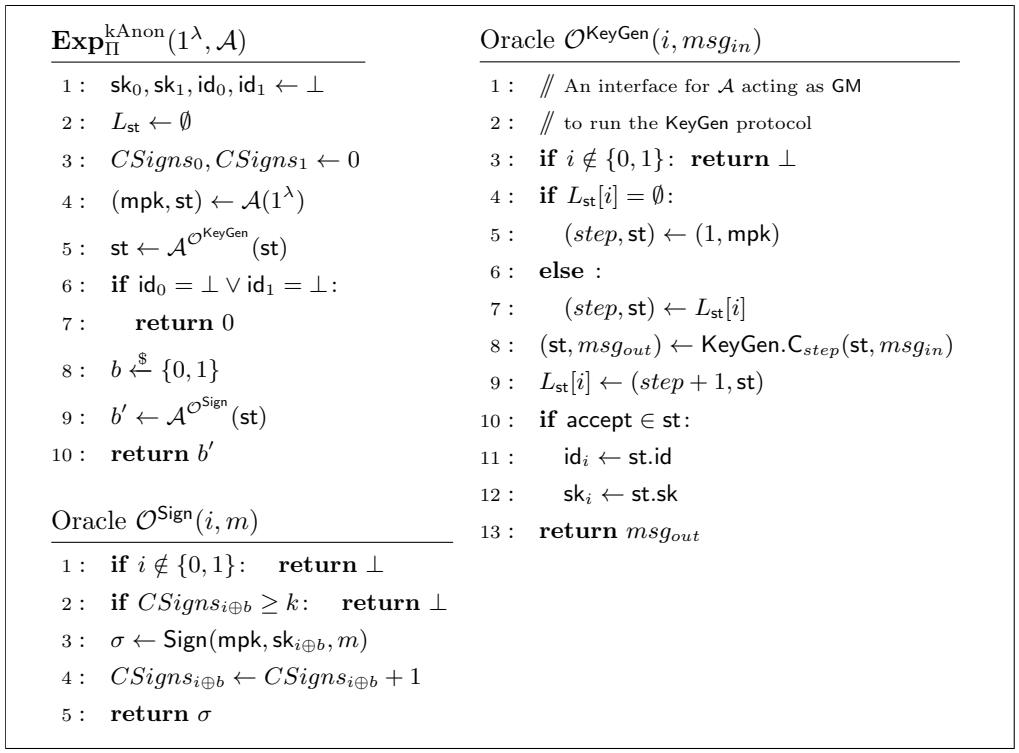

Figure 10: k-anonymity Experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}$

k-Anonymity (Simulation-Based): We now introduce a simulation-based definition of k-anonymity. For this purpose, we define an ideal functionality $\mathcal{F}^{\mathrm{kAnon}}_{\mathsf{kAGS}}$ that captures the notion of k-anonymity. The functionality is defined on Figure 11.

The functionality $\mathcal{F}^{\mathrm{kAnon}}_{\mathsf{kAGS}}$ involves two parties: the group manager GM and the client C. At the beginning of the execution, GM provides the functionality with a set of user identities representing registered users. The functionality forwards these identities to C.

The client may then "sign" messages by sending pairs (id, msg) to the functionality. Upon receiving such a request, the functionality replaces the real identity id with a fresh random identity fid, while maintaining a consistent bijection between real identities and fresh identities, and forwards the resulting "signature" to GM. In this way, the real identity of the user who signed a message is hidden from GM, yet all signatures originating from the same user are linked via the same fresh identity.

Both the ideal and real protocol executions take place in the presence of an environment $\mathcal{Z}$, which provides inputs to all parties and receives their outputs. We restrict $\mathcal{Z}$ as follows:

- It may prompt the client to sign at most k messages under the same identity.
- It may only use identities that were received from the functionality.

The first restriction enforces a separation between the properties of k-anonymity and traceability: after k "honest" signatures from the same user, a k-AGS scheme is designed to enable deanonymization of that user. An alternative would be to define $\mathcal{F}^{\mathrm{kAnon}}_{\mathsf{kAGS}}$ such that, after k signatures, the real identity is revealed in the signature delivered to GM. However, that would correspond precisely to the traceability property (under a malicious adversary).

We therefore restrict the environment and consider security only against a malicious group manager, i.e., the adversary may fully corrupt GM. This choice isolates k-anonymity as a standalone property, making the overall analysis more modular.

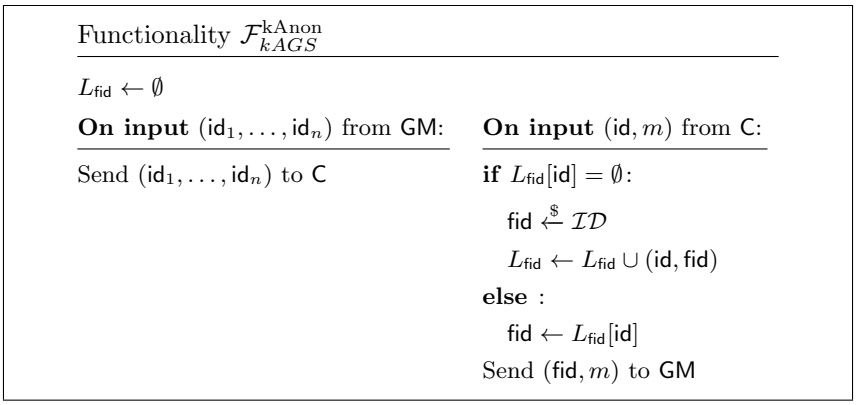

Figure 11: k-anonymity Functionality $\mathcal{F}_{kAGS}^{\text{kAnon}}$

Finally, we show that the standard game-based definition of k-anonymity implies this simulation-based formulation.

Claim 1 A k-AGS scheme $\Pi[k, \mathcal{M}, \mathcal{ID}]$ realizes the ideal functionality $\mathcal{F}_{kAGS}^{\text{kAnon}}$ if it is k-anonymous in the game-based sense.

Proof. We begin the proof with constructing a simulator and then proceed with proving indistinguishability of real- and ideal- words executions.

Simulator. Given an adversary $\mathcal{A}$ that corrupts a group manager GM in a real protocol execution, we construct a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{kAGS}}^{\text{kAnon}}$ as a group manager GM and simulates the outputs of $\mathcal{A}$ and honest parties to an environment for a distinguisher $\mathcal{D}$.

The simulator $\mathcal{S}$ starts the adversary $\mathcal{A}$ and receives mpk from it.

When $\mathcal{A}$, acting as a group manager, initiates a key generation session with a client by sending an identifier id, the simulator runs the KeyGen protocol with $\mathcal{A}$. Upon completing the protocol, $\mathcal{S}$ adds the pair $(\mathsf{id}, \mathsf{sk})$ to a list $\mathcal{L}_{\mathsf{sk}}$. Once the adversary finishes all key generation sessions, $\mathcal{S}$ sends all id's from the list $\mathcal{L}_{\mathsf{sk}}$ to the functionality in a single message.

Next, $\mathcal{S}$ receives anonymized "signatures" from the functionality. For each "signature" (fid, m), the simulator checks whether it has previously received a "signature" with the same fid—that is, whether there is an entry $(\mathsf{fid}, \mathsf{id}, \mathsf{sk})$ in a list $\mathcal{L}_{\mathsf{fid}}$. If so, it retrieves the corresponding sk. Otherwise, it selects a random id such that $(\mathsf{id}, \cdot) \in \mathcal{L}_{\mathsf{id}}$ but $(\cdot, \mathsf{id}, \cdot) \notin \mathcal{L}_{\mathsf{fid}}$ (i.e., a previously unused identity), retrieves the corresponding sk from $\mathcal{L}_{\mathsf{id}}$, and adds the triple $(\mathsf{fid}, \mathsf{id}, \mathsf{sk})$ to $\mathcal{L}_{\mathsf{pid}}$. Finally, $\mathcal{S}$ computes $\sigma = \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, m)$ and sends (σ, m).

Indistinguishability. We now observe that, given a distinguisher $\mathcal{D}$, we can construct an adversary $\mathcal{B}$ against the n-user version of the game-based k-anonymity experiment. Specifically, $\mathcal{B}$ starts the environment and the adversary $\mathcal{A}$ and simulates the real protocol execution using its own oracles. When $\mathcal{A}$ registers all n users, $\mathcal{B}$ chooses a random permutation π and outputs it to its experiment. It then uses its oracle to sign messages. Finally, $\mathcal{B}$ runs $\mathcal{D}$ on the generated transcript and outputs whatever $\mathcal{D}$ outputs.

It is easy to see that if the bit b in $\mathcal{B}$'s experiment is 0, then $\mathcal{B}$ perfectly simulates the real protocol execution for the environment and $\mathcal{A}$, and the generated transcript is distributed as in the real world. However, if $b = 1$, key generation remains normal, but signatures are computed using randomly permuted keys—this is exactly how $\mathcal{S}$ behaves when simulating signing for $\mathcal{A}$ in the ideal world. Indeed, after the registration phase, $\mathcal{S}$ effectively permutes the secret keys randomly and uses at most n fresh identities sid from the functionality as indices. Clearly, in that case, the transcript is distributed as in the ideal world. Hence, $\mathcal{B}$'s advantage equals $\mathcal{D}$'s advantage in distinguishing real and ideal transcripts. From Remark 2, follows the claim.

□

3.2 Construction

In this section, we introduce our generic construction $\Pi[k, \mathcal{M}, \mathcal{ID}]$ of a k-AGS for an arbitrary message space $\mathcal{M}$ and an identity space $\mathcal{ID} = \mathbb{Z}_p$. We make use of the following primitives in a black-box manner:

- a polynomial commitment scheme PC for polynomials over $\mathbb{Z}_p$;
- a two-round blind signature scheme BS;
- a public-key encryption scheme PKE for the message space $\mathbb{Z}_p$;
- a hash to field function $\mathcal{HF}$ from the message space $\mathcal{M}$ to $\mathbb{Z}_p$;
- a subversion-resistant non-interactive zero-knowledge argument system $\mathsf{NIZK}_{\mathsf{KG}}$ for a relation generator $\mathsf{RelGen}(1^\lambda)$, where the generator outputs relations $\mathcal{R}^{\mathsf{KG}}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{BS}}}$ corresponding to languages of the form[5]:

$$\mathcal{L} = \left\{ (msg, \mathsf{id}) \;\middle|\; \exists f \in \mathbb{Z}_p[x],\, R\colon \begin{array}{c} f(0) = \mathsf{id},\;\; com = \mathsf{PC.Commit}(\mathsf{ck}, f), \\ msg = \mathsf{BS.U}_1(\mathsf{pk}_{\mathsf{BS}}, com; R) \end{array} \right\}.$$

- a subversion-resistant non-interactive zero-knowledge argument system $\mathsf{NIZK}_{\mathsf{S}}$ for a relation generator $\mathsf{RelGen}_{\mathsf{S}}(1^\lambda)$, where the generator outputs relations $\mathcal{R}^{\mathsf{S}}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{PKE}}}$ corresponding to languages of the form[6]:

$$\mathcal{L} = \left\{ (x, ct, com) \;\middle|\; \exists y \in \mathbb{Z}_p,\, \omega_{\mathsf{PC}},\, R\colon \begin{array}{c} ct = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, y; R) \\ \mathsf{PC.Verify}(\mathsf{ck}, com, \omega_{\mathsf{PC}}, x, y) = 1 \end{array} \right\}.$$

The algorithms and the key generation protocol of Π are defined in Figure 12 and Figure 13 correspondingly. The master key generation algorithm Gen selects a hash function $\mathsf{H2F}$ from $\mathcal{HF}$ and includes its description in the public key—this step is omitted from the pseudocode for brevity. In the Trace algorithm, $\mathsf{Interpolate}$ denotes any efficient polynomial interpolation procedure that, given $d+1$ points $(x_i, y_i) \in \mathbb{Z}_p^2$, outputs a polynomial $f \in \mathbb{Z}_p[x]$ of degree at most d.

The key generation protocol KeyGen is specified by four algorithms: $\{\mathsf{GM}_1, \mathsf{C}_1, \mathsf{GM}_2, \mathsf{C}_2\}$. For the sake of compactness, we omit in the pseudo code the details of how the output states of GM_1 and C_1 are constructed and later parsed by GM_2 and C_2. We assume that the states contain intermediate values required for subsequent computations. The output messages of GM_1, C_1, and GM_2 are shown above the arrows.

The correctness of Π follows straightforwardly from the correctness of the underlying primitives.

Security. The theorems below state the security properties of our k-AGS scheme, namely traceability and k-anonymity. We provide brief proof sketches in the main body—more precisely, intuitions—for each theorem, while the full proofs are deferred to Appendix B.

Theorem 1 *Let $\Pi[k, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a k-AGS scheme as defined before. Assume the following building blocks:*

- $\mathsf{NIZK}_{\mathsf{KG}}$ *is a knowledge-sound NIZK scheme,* $\mathsf{NIZK}_{\mathsf{S}}$ *is simulation-extractable, and zero-knowledge;*
- BS *is a one-more unforgeable blind signature scheme,*
- PKE *is a correct and* IND-CPA*-secure public-key encryption scheme,*
- $\mathcal{HF}$ *is a collision-resistant hash function family,*
- PC *is a binding and hiding polynomial commitment scheme.*

Then $\Pi[k, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ is a traceable k-AGS scheme.

[5] $\mathsf{RelGen}(1^\lambda)$ runs $\mathsf{PC.Setup}(1^\lambda, k)$ and $\mathsf{BS.KeyGen}(1^\lambda)$ to generate such a relation.
[6] $\mathsf{RelGen}(1^\lambda)$ runs $\mathsf{PC.Setup}(1^\lambda, k)$ and $\mathsf{PKE.KeyGen}(1^\lambda)$ to generate such a relation.

$\mathsf{Gen}(1^\lambda, k)$

$\mathsf{ck} \leftarrow \mathsf{PC.Setup}(1^\lambda, k)$
$(\mathsf{pk_{BS}}, \mathsf{sk_{BS}}) \leftarrow \mathsf{BS.KeyGen}(1^\lambda)$
$(\mathsf{pk_{PKE}}, \mathsf{sk_{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$\mathsf{crs_{KG}} \leftarrow \mathsf{NIZK_{KG}.Setup}(1^\lambda, \mathcal{R}^{\mathsf{KG}}_{k,\mathsf{ck},\mathsf{pk_{BS}}})$
$\mathsf{crs_S} \leftarrow \mathsf{NIZK_S.Setup}(1^\lambda, \mathcal{R}^{\mathsf{S}}_{k,\mathsf{ck},\mathsf{pk_{PKE}}})$
$\mathsf{mpk} \leftarrow (\mathsf{ck}, \mathsf{pk_{BS}}, \mathsf{pk_{PKE}}, \mathsf{crs_{KG}}, \mathsf{crs_S})$
$\mathsf{msk} \leftarrow \mathsf{sk_{BS}}$
return $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{Trace}(\mathsf{msk}, k, \{\sigma_i\}_{i\in\mathcal{I}})$

$\mathsf{sk_{PKE}} \Leftarrow \mathsf{msk}$
$L_{\mathsf{sh}} \leftarrow \emptyset$
$L_{\mathsf{id}} \leftarrow \emptyset$
for $i \in \mathcal{I}$:
  $((x, ct), \pi_{\mathsf{KG}}), \mathsf{pid}) \Leftarrow \sigma_i$
  $y \leftarrow \mathsf{PKE.Dec}(\mathsf{sk_{PKE}}, ct)$
  if $(x, y) \notin L_{\mathsf{sh}}[\mathsf{pid}]$:
    $L_{\mathsf{sh}}[\mathsf{pid}] \leftarrow L_{\mathsf{sh}}[\mathsf{pid}] \cup (x, y)$
for $shares \in L_{\mathsf{sh}}$:
  if $|shares| \geq k + 1$:
    $f \leftarrow \mathsf{Interpolate}(shares)$
    $\mathsf{id} \leftarrow f(0)$
    $L_{\mathsf{id}} \leftarrow L_{\mathsf{id}} \cup \mathsf{id}$
return L_{id}

$\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg)$

$(\mathsf{ck}, \mathsf{pk_{PKE}}, \mathsf{crs_S}) \Leftarrow \mathsf{mpk}$
$(f, \mathsf{pid}) \Leftarrow \mathsf{sk}$
$x \leftarrow 0, i \leftarrow 0$
while $x = 0$:
  $x \leftarrow \mathsf{H2F}(\mathsf{pid} \| msg \| i)$
  $i \leftarrow i + 1$
$\omega_{\mathsf{PC}} \leftarrow \mathsf{PC.Open}(\mathsf{ck}, f, x, f(x))$
$(ct; R) \leftarrow \mathsf{PKE.Enc}(\mathsf{pk_{PKE}}, f(x))$
$\pi_{\mathsf{S}} \leftarrow \mathsf{NIZK_S.Prove}(\mathsf{crs_S}, (x, ct, com), (f(x), \omega_{\mathsf{PC}}, R))$
$\mathsf{sh} \leftarrow (x, ct)$
$\sigma \leftarrow ((\mathsf{sh}, \pi_{\mathsf{S}}), \mathsf{pid})$
return σ

$\mathsf{Verify}(\mathsf{mpk}, msg, \sigma)$

$(\mathsf{crs_{KG}}, \mathsf{pk_{BS}}) \Leftarrow \mathsf{mpk}$
$((\mathsf{sh}, \pi_{\mathsf{KG}}), \mathsf{pid}) \Leftarrow \sigma$
$(x, ct) \Leftarrow \mathsf{sh}$
$(com, \sigma_{\mathsf{BS}}) \Leftarrow \mathsf{pid}$
// Verify pseudoidentity
if $\mathsf{BS.Verify}(\mathsf{pk_{BS}}, \sigma_{\mathsf{BS}}, com) = 0$:
  return 0
// Verify share computed correctly
if $x \neq \mathsf{H2F}(\mathsf{pid} \| msg)$:
  return 0
if $\mathsf{NIZK_S.Verify}(\mathsf{crs_S}, (x, ct, com), \pi_{\mathsf{KG}}) = 0$:
  return 0
return 1

Figure 12: k-AGS construction $\Pi[k, \mathcal{M}, \mathcal{ID}]$

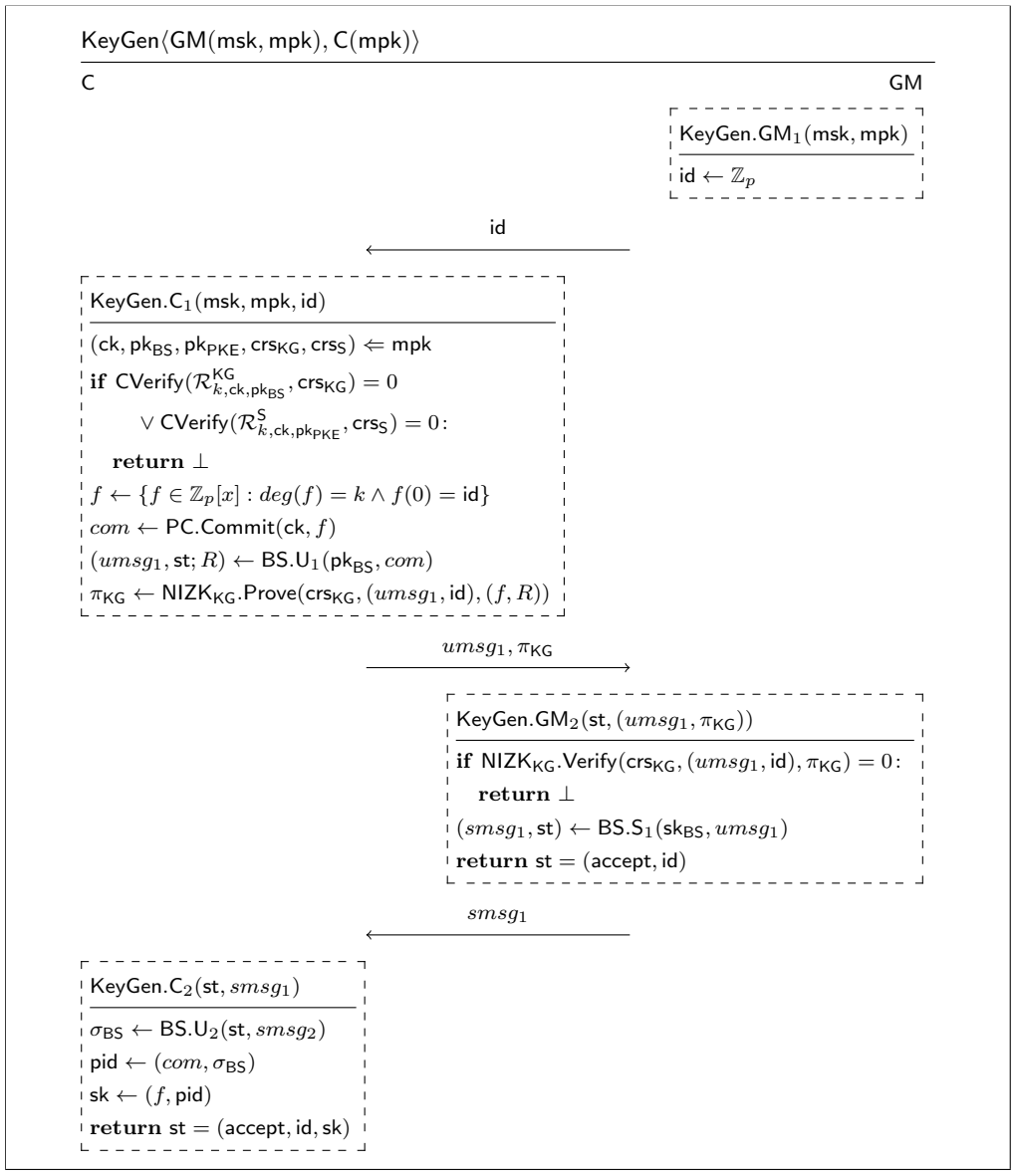

Figure 13: KeyGen protocol of k-AGS $\Pi[k, \mathcal{M}, \mathcal{ID}]$

Proof. (Sketch) As usual, we prove traceability via a sequence-of-games argument. In the initial games, we use the knowledge-soundness of $\mathsf{NIZK}_{\mathsf{KG}}$ and the one-more unforgeability of BS to show that all pseudo-identities in the adversary's output signatures were correctly formed with the participation of the group manager. That is, each pid is a commitment to a polynomial f of degree k, with $f(0) = \mathsf{id}$, blindly signed by GM.

Next, by relying on the simulation-extractability of $\mathsf{NIZK}_{\mathsf{S}}$—that is, knowledge soundness in the presence of simulated proofs (in the full proof, we must replace real NIZK proofs with simulated ones in signatures produced by honest users from the outset)—along with the correctness of PKE and the binding property of PC, we show that the PKE ciphertext in each signature decrypts to the y-coordinate of a share corresponding to the polynomial f committed in the associated $\mathsf{pid}.com$. From this, it follows that any $k+1$ signatures with the same pid will reveal the polynomial f, such that $f(0)$ equals some id previously registered with GM.

It remains to show that the adversary can only submit signatures from users it controls. In other words, $\mathcal{A}$ cannot forge signatures of honest users. This follows directly from zero-knowldge of $\mathsf{NIZK}_{\mathsf{S}}$, the INDCPA security of PKE and the hiding property of PC: the adversary cannot obtain the y-coordinates of shares from the ciphertexts to reconstruct the secret polynomial f, nor can it extract them from the polynomial commitments.

Therefore, the adversary can only produce correctly formed signatures for identities it has registered with GM. Consequently, at least one identity must be used more than k times, which allows GM to trace it. □

Theorem 2 *Let $\Pi[k, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a k-AGS scheme as defined before. Assume the following building blocks:*

- $\mathsf{NIZK}_{\mathsf{KG}}$ *is a subversion-zero-knowledge NIZK scheme,*
- PC *is a hiding polynomial commitment scheme,*
- BS *is a secure blind signature scheme.*

Then $\Pi[k, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ is a k-anonymous k-AGS scheme.

Proof. (Sketch) Since the k-Anonymity experiment is defined with the group manager as the adversary, we immediately consider a modification of Π: instead of including an encryption of the y-coordinate of a share in a signature, the signature directly contains the y-coordinate, as well as the polynomial commitment ω_{PC} in place of π_{S}. In other words, for k-Anonymity we do not rely on any properties of PKE or $\mathsf{NIZK}_{\mathsf{S}}$.

The proof is straightforward. First, we use the subversion-zero-knowledge of $\mathsf{NIZK}_{\mathsf{KG}}$ and the hiding property of PC to replace identities sent by the adversary in the KeyGen protocol with fresh, independently chosen ones. This makes the shares and commitments contained in signatures independent of the identities chosen by $\mathcal{A}$.

The only remaining way for the adversary to link signatures and registration sessions is through the blind signature σ_{BS} over commitments contained in pseudo-identities. However, $\mathcal{A}$ cannot achieve this due to the blindness property of BS. □

Discussion. We choose subversion-resistant NIZKs [BFS16, ALSZ21] as building blocks because they allow us to delegate the entire burden of generating the common reference string to the group manager. This contrasts with, for example, using the common random string model and making the user verifiably generate a pseudo-random crs for each signature. In our setting, the group manager performs this step only once during master key generation.

This choice benefits us in two ways: it simplifies both the presentation of the construction and the security proofs, and it allows us to hard-code crs after setting up the group manager. We note that if the crs setup is performed in a trusted manner, the subversion-resistance requirement can be relaxed.

The requirement of simulation-extractability for $\mathsf{NIZK}_{\mathsf{S}}$ is fairly conventional for NIZKs used in signatures of knowledge [CL06]. We leave as an open question to what extent our requirements on NIZKs can be relaxed.

From the KeyGen protocol, we also identify another interesting—albeit slightly tangential—open question: designing an efficient NIZK-blind signature pair that allows the prover to demonstrate to the server that the

message being signed satisfies a fixed NP relation. For instantiating our construction, we consider pairing-based primitives along with NIZKs tailored for proving relations over pairings to be the most straightforward approach.

4 k-Anonymous Set Pre-constrained Group Signatures

4.1 Definition and Security Notions

We now define k-ASPCGS along with its correctness and security notions.

Definition 8 *A k-Anonymous Set Pre-constrained group signature (k-ASPCGS) scheme $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$ with a message space $\mathcal{M}$, an identity space $\mathcal{ID}$, and a set size n, is a tuple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{OpenShare}, \mathsf{Trace})$ *of PPT algorithms along with an interactive protocol* KeyGen *defined as follows:*

$\underline{(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{D}, k) :}$ *The master key generation algorithm takes as input a security parameter 1^λ, a set $\mathcal{D} \subseteq \mathcal{M}$ of size at most n, and a threshold parameter k, and outputs a master public key* mpk *and a master secret key* msk*.*

$\underline{\langle \mathsf{id}, (\mathsf{id}, \mathsf{sk}) \rangle \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}), \mathsf{C} \rangle :}$ *The interactive key generation protocol is run by a group manager* GM *with an input* msk*, and by a client* C*. It delivers an identity* $\mathsf{id} \in \mathcal{ID}$ *to both* GM *and* C*, and an identity signing key* sk *to* C*.*

$\underline{\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg) :}$ *The signing algorithm takes as input a master public key* mpk*, an identity signing key* sk*, and a message $msg \in \mathcal{M}$, and outputs a signature σ.*

$\underline{\{1, 0\} \leftarrow \mathsf{Verify}(\mathsf{mpk}, msg, \sigma) :}$ *The signature verification algorithm takes as input a master public key* mpk*, a message $msg \in \mathcal{M}$, and a signature σ, and outputs either 1 or 0, indicating accept or reject.*

$\underline{\{\mathsf{id}_1, \ldots, \mathsf{id}_m\} \leftarrow \mathsf{Trace}(\mathsf{msk}, k, \{\sigma_i\}_{i \in \mathcal{I}}) :}$ *The identity reconstruction algorithm takes as input a master secret key* msk*, the scheme anonymity parameter k, a set of signatures $\{\sigma_i\}_{i \in \mathcal{I}}$, and outputs a potentially empty set of corresponding identities $\{\mathsf{id}_1, \ldots, \mathsf{id}_m\}, \mathsf{id}_i \in \mathcal{ID}$.*

Remark 3 As in Definition 7, we define the key generation protocol as a tuple of PPT algorithms

$$(\mathsf{GM}_1, \mathsf{C}_1, \mathsf{GM}_2, \mathsf{C}_2, \ldots, \mathsf{GM}_i, \mathsf{C}_i),$$

subject to the same conditions on their inputs and outputs.

Correctness: *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, for every $\mathcal{D} \subset \mathcal{M}$, and for every $m \in \mathcal{M}$, the following holds:*

$$\Pr\left[\mathsf{Verify}(\mathsf{mpk}, msg, \sigma) = 1 \;:\; \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{D}, k) \\ \langle \mathsf{id}, (\mathsf{id}, \mathsf{sk}) \rangle \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}, k), \mathsf{C} \rangle \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg) \end{array}\right] = 1$$

Security. We now regard the security notions of a k-ASPCGS scheme. Analogous to k-AGS, these notions are traceability and k-anonymity. For k-anonymity, we only consider a simulation-based definition, as this is the conventional approach when defining security for PSI-based constructions against a dishonest server. Our simulation-based definition extends the functionality of k-AGS by incorporating the functionality of SPCE [BGJP22].

Traceability: *For a PPT adversary $\mathcal{A}$, let its advantage against traceability of a k-ASPCGS scheme Π be defined as follows:*

$$\mathsf{Adv}^{\mathrm{Trace}}_{\Pi,\mathcal{D}}(\mathcal{A},\lambda) = \Pr[\mathbf{Exp}^{\mathrm{Trace}}_{\Pi,\mathcal{D}}(1^\lambda,\mathcal{A}) \to 1],$$

where the traceability experiment $\mathbf{Exp}^{\Pi}_{\mathrm{Trace}}$ for k-ASPCGS is defined in Figure 9.[7]

A k-ASPCGS scheme Π is traceable if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, $\mathcal{D} \subseteq \mathcal{M}, |\mathcal{D}| \leq n$, and for every PPT adversary $\mathcal{A}$, the following holds:

$$\mathsf{Adv}^{\mathrm{Trace}}_{\Pi,\mathcal{D}}(\mathcal{A},\lambda) \leqslant \mathsf{negl}(\lambda).$$

k-Anonymity (Simulation-Based): To define k-anonymity of k-ASPCGS, we adopt a simulation-based approach, as is standard in the PSI literature when analyzing security against a malicious server. Concretely, with minor modifications, we follow the formalization of [BGJP22], extending it with a k-anonymity functionality.

The ideal functionality $\mathcal{F}^{\mathrm{kAnon}}_{kASPCGS}$ involves three parties: the group manager GM, the client C, and an authority Auth. The role of Auth is to verify, using the functionality, that the set $\mathcal{D}$ provided by GM satisfies one of the following conditions:

- In the case of *bounded-set* security: $|\mathcal{D}| \leq n$.
- In the case of *authenticated-set* security: $\mathcal{D} \subseteq \mathcal{D}^*$, where $\mathcal{D}^*$ is a secret reference set held by Auth.

At the beginning of the execution, Auth provides the functionality with a pair $(\mathsf{pp}, \mathsf{sp})$, where pp equals n, the maximum allowed set size, and sp is either empty (bounded-set case) or contains $\mathcal{D}^*$ (authenticated-set case). The functionality then evaluates a predicate $\mathcal{P}$ to check the corresponding condition. Then, the functionality proceeds as the k-Anonymity ideal functionality of kAGS defined in Secion 3.1. The formal description of $\mathcal{F}^{\mathrm{kAnon}}_{kASPCGS}$ is given in Figure 14.

Functionality $\mathcal{F}^{\mathrm{kAnon}}_{kASPCGS}$

$L_{\mathsf{fid}} \leftarrow \emptyset$

On input $(\mathsf{pp}, \mathsf{sp})$ from Auth:

Send pp to C and $(\mathsf{pp}, \mathsf{sp})$ to GM

On input $(\mathcal{D}, (\mathsf{id}_1, \ldots, \mathsf{id}_n))$ from GM:

if $\mathcal{P}(\mathsf{pp}, \mathsf{sp}, \mathcal{D}) = 0$:
  Send $\perp$ to Auth, GM, C and halt
else :
  Send $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$ to C

On input (id, m) from C:

if $m \in \mathcal{D}$:
  if $L_{\mathsf{fid}}[\mathsf{id}] = \emptyset$:
    $\mathsf{fid} \xleftarrow{\$} \mathcal{ID}$
    $L_{\mathsf{fid}} \leftarrow L_{\mathsf{fid}} \cup (\mathsf{id}, \mathsf{fid})$
  else :
    $\mathsf{fid} \leftarrow L_{\mathsf{fid}}[\mathsf{id}]$
  Send $(\mathsf{fid}, m, 1)$ to GM and (fid, m) to Auth
else :
  Send $(\mathsf{fid}, m, 0)$ to GM and (fid, m) to Auth

Figure 14: k-anonymity functionality $\mathcal{F}^{\mathrm{kAnon}}_{kASPCGS}$

Both the ideal and real protocol executions take place in the presence of an environment $\mathcal{Z}$, which provides inputs to all parties and receives their outputs. We restrict $\mathcal{Z}$ in the following way:

- It may prompt the client to sign at most k messages from $\mathcal{D}$ under the same identity.
- It may only use identities that were received from the functionality.

[7] We defer the pseudocode of the experiment to Appendix A, as it is almost identical to the traceability experiment for k-AGS.

We consider security against adversaries that can corrupt at most one party. If the client is corrupted, we restrict the adversary to be semi-honest: it follows the protocol specification but attempts to learn additional information about $\mathcal{D}$.

4.2 Construction

In this section we briefly discuss a simple modification of our k-AGS construction, that achieves Set Pre-Constrained definitions. Our k-ASPCGS construction $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$ uses the same building blocks as our k-AGS construction (see Section 3.2), except for:

- a set pre-constrained encryption scheme SPCE over a universe $\mathcal{M}$, with the message $\mathbb{Z}_p$ is used instead of a public-key encryption scheme PKE;
- a subversion-resistant non-interactive zero-knowledge argument system $\mathsf{NIZK}_{\mathsf{S}}$ is defined for a different relation generator $\mathsf{RelGen}_{\mathsf{S}}(1^\lambda)$. The generator outputs relations $\mathcal{R}^{\mathsf{S}}_{k,\mathcal{D},\mathsf{ck},\mathsf{pk}_{\mathsf{PKE}}}$ corresponding to languages of the form[8]:

$$\mathcal{L} = \left\{ (x, ct, com) \;\middle|\; \exists y \in \mathbb{Z}_p,\, \omega_{\mathsf{PC}},\, R\colon \begin{array}{c} ct = \mathsf{SPCE.Enc}(\mathsf{pk}_{\mathsf{SPCE}}, msg, y; R) \\ \mathsf{PC.Verify}(\mathsf{ck}, com, \omega_{\mathsf{PC}}, x, y) = 1 \end{array} \right\}.$$

The modification is, thereof, straightforward:

- in the master key generation algorithm Gen, instead $\mathsf{PKE.KeyGen}(1^\lambda)$ the SPCE key generation algorithm $\mathsf{SPCE.KeyGen}(1^\lambda, \mathcal{D})$ with the set $\mathcal{D}$ as input;
- in the signing algorithm Sign, the y-coordinate of the share, i.e. $f(x)$, is SPC encrypted with respect to message msg being signed;
- in the tracing algorithm Trace, SPCE decryption is used instead of PKE.

Note that Gen may become a protocol between the group manager and an authority that authenticates (or owns) the set $\mathcal{D}$, if the underlying SPCE provides such functionality. We provide full description of the scheme in Appendix C. The correctness of Π follows straightforwardly from the correctness of the underlying primitives.

Security. Next, we discuss traceability and k-anonymity of our k-ASPCGS construction $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$.

Theorem 3 *Let $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a k-ASPCGS scheme as defined above. Assume the following building blocks:*

- $\mathsf{NIZK}_{\mathsf{KG}}$ *is a knowledge-sound NIZK scheme,* $\mathsf{NIZK}_{\mathsf{S}}$ *is simulation-extractable, and zero-knowledge,*
- BS *is a one-more unforgeable blind signature scheme,*
- SPCE *is a perfectly correct and* oINDCPA*-secure set pre-constrained encryption scheme,*
- $\mathcal{HF}$ *is a collision-resistant hash function family,*
- PC *is a binding and hiding polynomial commitment scheme.*

Then $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ is a traceable k-ASPCGS scheme.

Proof. (Sketch) The proof follows the same structure as the proof of Theorem 1, with the following exceptions:

- The perfect correctness of SPCE is used in place of the perfect correctness of PKE. This ensures that for each signature of a "bad" message $msg \in \mathcal{D}$, decrypting the ct part of the signature yields the correct y-coordinate when Trace is executed.

[8] Formally, $\mathsf{RelGen}(1^\lambda)$ samples some set $\mathcal{D}, |\mathcal{D}| = n$, runs $\mathsf{PC.Setup}(1^\lambda, k)$ and $\mathsf{SPCE.KeyGen}(1^\lambda, \mathcal{D})$ to generate a relation.

- The outsider INDCPA security of SPCE is used instead of the INDCPA security of PKE, in exactly the same way. □

Theorem 4 *Let $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a k-ASPCGS scheme as defined above. Assume the following building blocks:*

- $\mathsf{NIZK}_{\mathsf{KG}}$ *and* $\mathsf{NIZK}_{\mathsf{S}}$ *are subversion-zero-knowledge NIZK schemes,*
- SPCE *realizes the ideal functionality* $\mathcal{F}_{\mathsf{SPCE}}$*,*
- PC *is a hiding polynomial commitment scheme,*
- BS *is a secure blind signature scheme.*

Then $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ realizes the ideal functionality $\mathcal{F}^{\text{kAnon}}_{kASPCGS}$.

Proof. (Sketch) We prove the theorem by gradually modifying the real protocol execution and building a simulator, until the execution matches the ideal functionality $\mathcal{F}^{\text{kAnon}}_{kASPCGS}$.

In contrast to the k-anonymity proof of our k-AGS scheme (Section B), security of our k-AGS construction crucially relies on the security properties of SPCE and $\mathsf{NIZK}_{\mathsf{S}}$. In particular, SPCE must hide the y-coordinate of a share from GM whenever the message is not "bad," while $\mathsf{NIZK}_{\mathsf{S}}$ must hide the witness used for encryption, which includes this coordinate.

Thereof, we begin by modifying signing in the real protocol execution. Concretely, we insert an intermediate functionality that simply forwards all messages between GM and C (including the KeyGen protocol messages).

Step 1: Introducing a dummy simulator. We place a dummy simulator $\mathcal{S}$ between the forwarding functionality and GM (controlled by $\mathcal{A}$). At this stage, $\mathcal{S}$ does nothing beyond relaying messages.

Step 2: Simulating $\mathsf{NIZK}_{\mathsf{S}}$ *proofs.* When the client signs a message, it no longer includes a $\mathsf{NIZK}_{\mathsf{S}}$ proof in the signature. Instead, upon receiving the signature from the functionality, the simulator computes a simulated proof

$$\mathsf{NIZK}_{\mathsf{S}}.\mathsf{Sim}(\mathsf{crs}_{\mathsf{S}}, \mathsf{td}, (x, ct, com)),$$

using a trapdoor td extracted beforehand, and inserts it into the signature, that is forwarded to $\mathcal{A}$. This change is indistinguishable from the real execution by the subversion zero-knowledge property of $\mathsf{NIZK}_{\mathsf{S}}$.

Step 3: Replacing SPCE *by* $\mathcal{F}_{\mathsf{SPCE}}$*.* We replace the use of SPCE with its ideal functionality. The client no longer includes the ciphertext in the signature of a message msg, but instead sends the share $(x, f(x))$ to the intermediate functionality. The functionality forwards $f(x)$ to $\mathcal{S}$ if and only if $msg \in \mathcal{D}$. The simulator then uses the SPCE simulator to produce a ciphertext ct for the adversary, and also invokes the SPCE simulator when GM generates keying material. Indistinguishability follows since SPCE UC-realizes $\mathcal{F}_{\mathsf{SPCE}}$.

At this point, the simulator learns the full share $(x, f(x))$ only when $msg \in \mathcal{D}$; otherwise, it learns only the x-coordinate (which it can compute itself) together with the pseudo-identity pid.

Step 4: Adapting KeyGen*.* The remainder of the proof proceeds similarly to the k-anonymity proof of our k-AGS scheme, with adaptations for the ideal functionality formalism:

- During the execution of KeyGen, the client does not include $\mathsf{NIZK}_{\mathsf{KG}}$ proofs. The simulator instead generates simulated proofs using $\mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Sim}$.
- The client stops its active participation in KeyGen. When GM initiates the protocol with an identity id, the simulator chooses a fresh identity id^{new}, computes $(f, com, umsg_1)$, and the corresponding simulated proof and reply for $\mathcal{A}$. It records the mapping between id^{new} (and its secret key) and id. The simulator delivers id to the functionality, which passes it to C.

  When C signs a message, it simply sends (id, msg) to the functionality, which forwards (id, msg) together with a bit indicating whether $msg \in \mathcal{D}$ to $\mathcal{S}$. The simulator computes the remainder of the signature. Since $\mathcal{S}$ never "encrypts" more than k evaluations of f, indistinguishability holds by the hiding property of PC.

- The functionality replaces each client identity id with a fresh random identity fid (using the same fid consistently for a given id). The simulator then maps these fresh identities to the secret keys and pseudo-identities it generated earlier one-by-one. This step is indistinguishable by the (multi-user) blindness property of BS.

It is now straightforward to verify that the resulting functionality coincides exactly with the ideal functionality $\mathcal{F}^{\text{kAnon}}_{kASPCGS}$. □

4.3 Generic Transform

In this section, we introduce our generic construction $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$ of a k-ASPCGS for an arbitrary message space $\mathcal{M}$ and an identity space $\mathcal{ID} = \mathbb{Z}_p$. We make use of the following primitives in a black-box manner:

- a kAGS for a threshold k, message space $\mathcal{M}$, and identity space $\mathcal{ID}$;
- a set pre-constrained encryption scheme SPCE over a universe $\mathcal{M}$, with the message space being the signature space of kAGS;
- a subversion-resistant non-interactive zero-knowledge argument system NIZK for a relation generator $\mathsf{RelGen}(1^\lambda)$, where the generator outputs relations $\mathcal{R}^{kASPCGS}_{k,\mathsf{pk}_{\mathsf{SPCE}},\mathsf{mpk}_{\mathsf{kAGS}}}$ corresponding to languages of the form[9]:

$$\mathcal{L} = \left\{ (msg, ct) \;\middle|\; \exists \sigma,\, R\colon \begin{array}{c} 1 \leftarrow \mathsf{kAGS.Verify}(\mathsf{mpk}_{\mathsf{kAGS}}, (msg, \sigma)), \\ ct \leftarrow \mathsf{SPCE.Enc}(\mathsf{pk}_{\mathsf{SPCE}}, msg, \sigma; R) \end{array} \right\}.$$

The algorithms and the key generation protocol of Π are defined in Figure 15. Note that the key generation protocol KeyGen is identical to the key generation protocol of kAGS. Furthermore, Gen may also be a protocol between the group manager and an authority that authenticates (or owns) the set $\mathcal{D}$, if the underlying SPCE provides such functionality.

The correctness of Π follows straightforwardly from the correctness of the underlying primitives.

Security. Next, we discuss the security of the generic k-ASPCGS transform $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$. We provide a brief proof sketch for the traceability theorem, with the full proof deferred to Appendix D. The k-anonymity theorem is straightforward.

Theorem 5 *Let $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a generic k-ASPCGS transform as defined above. Assume NIZK is a knowledge-sound NIZK scheme, SPCE is a perfectly correct set pre-constrained encryption scheme, and kAGS is a traceable k-anonymous group signature scheme. Then $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ is a traceable k-ASPCGS scheme.*

Proof. (Sketch) The proof is straightforward. First, we use the knowledge soundness of NIZK and the correctness of SPCE to show that the SPCE ciphertext in each signature decrypts to a valid k-AGS signature whenever the corresponding message is "bad." Then, we construct a direct reduction to the traceability of the underlying k-AGS. □

Theorem 6 *Let $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ be a k-ASPCGS scheme as defined above. Assume NIZK is a subversion-zero-knowledge NIZK scheme, SPCE realizes the ideal functionality $\mathcal{F}_{\mathsf{SPCE}}$, and kAGS realizes the ideal functionality $\mathcal{F}^{\text{kAnon}}_{kAGS}$. Then $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$ realizes the ideal functionality $\mathcal{F}^{\text{kAnon}}_{kASPCGS}$.*

Proof. The statement of the theorem follows immediately from the subversion-zero-knowledge property of NIZK and from the fact that kAGS and SPCE realize their respective ideal functionalities. □

[9] Formally, $\mathsf{RelGen}(1^\lambda)$ samples some set $\mathcal{D}, |\mathcal{D}| = n$, runs $\mathsf{kAGS.Gen}(1^\lambda, k)$ and $\mathsf{SPCE.Gen}(1^\lambda, \mathcal{D})$ to generate such a relation.

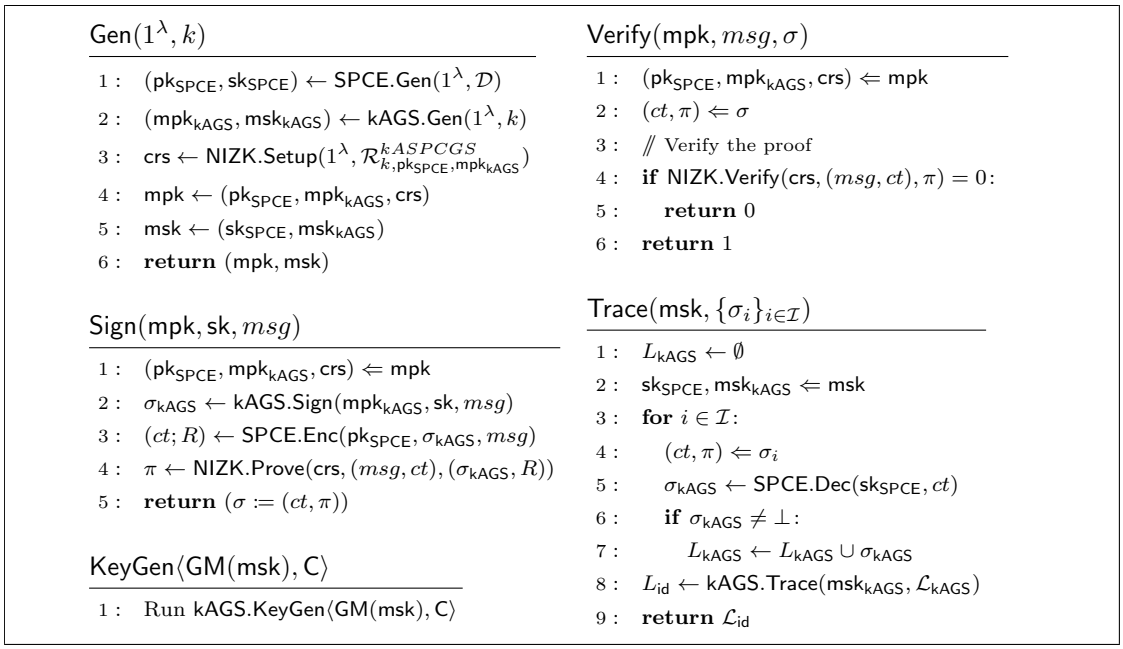

Figure 15: k-ASPCGS construction $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$

References

[AJ22] Prabhanjan Ananth and Abhishek Jain. Pre-constrained encryption. In *13th Innovations in Theoretical Computer Science Conference*, 2022.

[ALSZ21] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michał Zajac. On subversion-resistant snarks. *Journal of Cryptology*, 34(3):17, 2021.

[ASY06] Man Ho Au, Willy Susilo, and Siu-Ming Yiu. Event-oriented k-times revocable-iff-linked group signatures. In *Australasian Conference on Information Security and Privacy*, pages 223–234. Springer, 2006.

[BBM+21] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The Apple PSI system. 2021.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted crs: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, volume 10032 of *Lecture Notes in Computer Science*, page 777–804, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BGH+25] Pedro Branco, Matthew Green, Aditya Hegde, Abhishek Jain, and Gabriel Kaptchuk. How to trace viral content in end-to-end encrypted messaging. *Cryptology ePrint Archive*, 2025.

[BGJP22] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. End-to-end secure messaging with traceability only for illegal content. 2022.

[BGJP25] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. Laconic PSI on authenticated inputs and applications. 2025.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In

Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, page 614–629, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. 2000.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *International conference on security and cryptography for networks*, pages 141–155. Springer, 2006.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, page 78–96, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[CLXZ25] Wonseok Choi, Xiangyu Liu, Lirong Xia, and Vassilis Zikas. K-linkable ring signatures and applications in generalized voting. *Cryptology ePrint Archive*, 2025.

[CoEU25] Council of European Union. Proposal for a regulation of the European Parliament and of the Council laying down rules to prevent and combat child sexual abuse. https://cdn.netzpolitik.org/wp-upload/2025/07/2025-07-24_Council_Presidency_LEWP_CSA-R_Compromise-texts_11596.pdf, 2025.

[EBG+24] Harry Eldridge, Gabrielle Beck, Matthew Green, Nadia Heninger, and Abhishek Jain. Abuse-resistant location tracking: Balancing privacy and safety in the offline finding ecosystem. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5431–5448, 2024.

[EU21] European Union. Regulation (EU) 2021/784 of the European Parliament and of the Council. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A32021R0784, 2021.

[fJ21] Brennan Center for Justice. Facebook's content moderation rules are a mess. https://www.brennancenter.org/our-work/analysis-opinion/facebooks-content-moderation-rules-are-mess, 2021.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge snarks. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, volume 10769 of *Lecture Notes in Computer Science*, page 315–347, Cham, 2018. Springer International Publishing.

[GKVL21] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 553–583. Springer, 2021.

[Gua25] The Guardian. Reading the post-riot posts: how we traced far-right radicalisation across 51,000 facebook messages. https://www.theguardian.com/world/2025/sep/28/reading-the-post-riot-posts-how-we-traced-far-right-radicalisation-across-51000-facebook-messages?, 2025.

[HLZW24] Zhengan Huang, Junzuo Lai, Gongxian Zeng, and Jian Weng. Mild asymmetric message franking: Illegal-messages-only and retrospective content moderation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 266–295. Springer, 2024.

[IAV22] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. Hecate: Abuse reporting in secure messengers with sealed sender. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2335–2352, 2022.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Let25] Open Letter. Open letter on the position of scientists and researchers on the eu's proposed child sexual abuse regulation. https://csa-scientist-open-letter.org/Sep2025, 2025.

[LRTY21] Linsheng Liu, Daniel S Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (facts). *arXiv preprint arXiv:2109.04559*, 2021.

[LZH+23] Junzuo Lai, Gongxian Zeng, Zhengan Huang, Siu Ming Yiu, Xin Mu, and Jian Weng. Asymmetric group message franking: Definitions and constructions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 67–97. Springer, 2023.

[May19] Jonathan Mayer. Content moderation for end-to-end encrypted messaging. *Princeton University*, 2019.

[New25] The News. Google flags child's groin images taken for doctor as sexual abuse material. https://www.thenews.com.pk/latest/985350-google-flags-childs-groin-images-taken-for-doctor-as-sexual-abuse-material, 2025.

[NSN05] Lan Nguyen and Rei Safavi-Naini. Dynamic k-times anonymous authentication. In *International Conference on Applied Cryptography and Network Security*, pages 318–333. Springer, 2005.

[Pre20] Majority Press. U.S. senate committee on the judiciary. https://www.judiciary.senate.gov/press/rep/releases/graham-blumenthal-hawley-feinstein-introduce-earn-it-act-to-encourage-tech-industry-to-take-online-child-sexual-exploitation-seriously, 2020.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. K-times anonymous authentication. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 308–322. Springer, 2004.

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

[TNL+25] Nam Tran, Khoa Nguyen, Dongxi Liu, Josef Pieprzyk, and Willy Susilo. Many-time linkable ring signatures. *Cryptology ePrint Archive*, 2025.

[TS06] Isamu Teranishi and Kazue Sako. K-times anonymous authentication with a constant proving cost. In *International Workshop on Public Key Cryptography*, pages 525–542. Springer, 2006.

SUPPLEMENTAL MATERIALS

A Traceability Experiment for k-ASPCGS

$\mathbf{Exp}^{\text{Trace}}_{\Pi,\mathcal{D}}(1^\lambda, \mathcal{A})$

$L_{\mathsf{id}}, L_{\mathsf{st}}, L_{\mathsf{sk}}, L_{msg} \leftarrow \emptyset$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, k)$
$Dmsg \leftarrow 0$
$(s, \{(msg_i, \sigma_i)\}_{i=1,\ldots,s\}}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}, \cdots}(1^\lambda, \mathsf{mpk})$
if $\exists (i,j)$: $msg_i = msg_j \vee msg_i \in L_{msg}$:
  return 0
for $i = 1, \ldots, s$:
  if $\mathsf{Verify}(\mathsf{mpk}, msg_i, \sigma_i) = 0$:
    return 0
  if $msg_i \in \mathcal{D}$: $Dmsg \leftarrow Dmsg + 1$
// Check if the adversary abused the system
if $Dmsg \leqslant k \cdot |L_{\mathsf{id}}|$:
  return 0
// Trace identities
$\{\mathsf{id}_1, \ldots, \mathsf{id}_m\} \leftarrow \mathsf{Trace}(\mathsf{msk}, k, \{\sigma_1, \ldots, \sigma_s\})$
// If there is at least one id that belongs
// to the adversary, they lose
if $\{\mathsf{id}_1, \ldots, \mathsf{id}_m\} \cap L_{\mathsf{id}} \neq \emptyset$:
  return 0
// Else they succeeded in hiding identity
return 1

Oracle $\mathcal{O}^{\mathsf{HonestKG}}()$

// Oracle for $\mathcal{A}$ to register honest users
$(\mathsf{id}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}\langle \mathsf{GM}(\mathsf{msk}), \mathsf{C} \rangle$
$\mathcal{L}_{\mathsf{sk}} \leftarrow (\mathsf{id}, \mathsf{sk})$
return id

Oracle $\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{sid}, msg_{in})$

// An interface for $\mathcal{A}$ to run
// the KeyGen protocol with GM
if $L_{\mathsf{st}}[\mathsf{sid}] = \emptyset$:
  $(step, \mathsf{st}) \leftarrow (1, (\mathsf{msk}, \mathsf{mpk}))$
else :
  $(step, \mathsf{st}) \leftarrow L_{\mathsf{st}}[\mathsf{sid}]$
$(\mathsf{st}, msg_{out}) \leftarrow \mathsf{KeyGen.GM}_{step}(\mathsf{st}, msg_{in})$
$L_{\mathsf{st}}[\mathsf{sid}] \leftarrow (step + 1, \mathsf{st})$
if accept $\in \mathsf{st}$:
  $L_{\mathsf{id}} \leftarrow L_{\mathsf{id}} \cup \mathsf{st.id}$
return msg_{out}

Oracle $\mathcal{O}^{\mathsf{Sign}}(\mathsf{id}, msg)$

// An oracle for $\mathcal{A}$ to get
// signatures from honest users
$L_{msg} \leftarrow L_{msg} \cup msg$
if $\notin L_{\mathsf{sk}}[\mathsf{id}] = \emptyset$:
  return 0
$\mathsf{sk} \leftarrow L_{\mathsf{sk}}[\mathsf{id}]$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg)$
return σ

Figure 16: Traceability experiment $\mathbf{Exp}^{\text{Trace}}_{\Pi}$

B Security Proofs of k-AGS

B.1 Traceability

Theorem 7 *Let $\mathcal{A}$ be an adversary against the traceability of the k-AGS scheme $\Pi[\mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$. Let $\mathsf{NIZK}_{\mathsf{KG}}$ be knowledge-sound NIZK scheme and $\mathsf{NIZK}_{\mathsf{S}}$ be simulation-extractable, and zero-knowledge, let BS be a one-more-unforgeable blind signature, let PKE be a correct* INDCPA *secure public-key encryption scheme, let $\mathcal{HF}$ be a collision-resistant hash function family, and let PC be a binding and hiding polynomial commitment scheme. Then there exist negligible functions $\mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda)$, $\mathsf{negl}^{\mathrm{SE}}_{\mathsf{NIZK}_{\mathsf{S}}}(\lambda)$, $\mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{S}}}(\lambda)$ $\mathsf{negl}^{\mathrm{omUnforg}}_{\mathsf{BS}}(\lambda)$, $\mathsf{negl}^{\mathrm{corr}}_{\mathsf{PKE}}(\lambda)$, $\mathsf{negl}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(\lambda)$, $\mathsf{negl}^{\mathrm{cr}}_{\mathsf{H2F}}$, $\mathsf{negl}^{\mathrm{Bind}}_{\mathsf{PC}}(\lambda)$, and $\mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda)$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{Trace}}_{\Pi}(\mathcal{A},\lambda) \leqslant\ & \mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda) + \mathsf{negl}^{\mathrm{omUnforg}}_{\mathsf{BS}}(\lambda) + q_{\mathsf{Sign}}\mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{S}}}(\lambda)\\ & + \mathsf{negl}^{\mathrm{SE}}_{\mathsf{NIZK}_{\mathsf{S}}}(\lambda) + \mathsf{negl}^{\mathrm{corr}}_{\mathsf{PKE}}(\lambda) + \mathsf{negl}^{\mathrm{cr}}_{\mathcal{HF}}(\lambda)\\ & + \mathsf{negl}^{\mathrm{Bind}}_{\mathsf{PC}}(\lambda) + \mathsf{negl}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(\lambda) + \mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda),\end{aligned}$$

where q_{Sign} is a maximum number of queries $\mathcal{A}$ makes to $\mathcal{O}^{\mathsf{Sign}}$.

Proof. We construct a sequence of indistinguishable games $G_0, \ldots, G_9$, where G_0 corresponds exactly to the experiment $\mathbf{Exp}^{\mathrm{Trace}}_{\Pi}(1^\lambda, \mathcal{A})$, and G_9 is a game in which the adversary cannot win.

G_1: By the knowledge-soundness property of $\mathsf{NIZK}_{\mathsf{KG}}$, for the adversary $\mathcal{A}$ there exists a knowledge extractor $\mathcal{E}_{\mathcal{A}}$. In G_1, using the extractor $\mathcal{E}_{\mathcal{A}}$, we modify the KeyGen oracle as follows.

If an adversarial query to $\mathcal{O}^{\mathsf{KeyGen}}$ results in the state accept, this means that in that query the adversary provided a message $umsg_1$ and a proof π_{KG} such that $\mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Verify}(\mathsf{crs}, (umsg_1, \mathsf{id}), \pi_{\mathsf{KG}}) = 1$, where id is an identity generated in a previous adversarial query to KeyGen corresponding to that key-generation session. The challenger then uses the extractor $\mathcal{E}_{\mathcal{A}}$ to obtain a corresponding witness w and checks whether $\mathcal{R}_{k,\mathsf{ck},\mathsf{pk}_{BS}}((umsg_1, \mathsf{id}), w) = 1$. If this check fails, the challenger aborts the game (outputting 0). We call this a failure event and denote it by fail_{01}. Note that if the game does not abort, it holds that $f(0) = \mathsf{id} \in L_{\mathsf{id}}$.

We now construct an adversary $\mathcal{B}$ against the knowledge-soundness of $\mathsf{NIZK}_{\mathsf{KG}}$ as follows. $\mathcal{B}$ perfectly simulates G_1 for $\mathcal{A}$, using the $\mathsf{NIZK}_{\mathsf{KG}}$ parameters received from its own challenger. If the failure event occurs, $\mathcal{B}$ outputs $((umsg_1, \mathsf{id}), \pi_{\mathsf{KG}})$ as its solution in the knowledge-soundness game and wins. If the failure event does not occur, $\mathcal{B}$ outputs random data. Hence, the probability of $\mathcal{B}$ winning the knowledge-soundness game is at least the probability of the failure event in G_1.

Since G_0 and G_1 behave identically unless the failure event occurs, we have:

$$\begin{aligned}\Pr[G_0(\mathcal{A}, 1^\lambda) \to 1] &\leqslant \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] + \Pr[\mathsf{fail}_{01} \textit{ in } G_1(\mathcal{A}, 1^\lambda)]\\ &\leqslant \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda). \quad (1)\end{aligned}$$

G_2: In this game, for each adversarial query to $\mathcal{O}^{\mathsf{KeyGen}}$ that results in the state accept and does not cause the game to abort, the challenger parses the witness w extracted using $\mathcal{E}_{\mathcal{A}}$ as (f, R), computes a commitment $com \leftarrow \mathsf{PC}.\mathsf{Commit}(\mathsf{ck}, f)$, and then obtains a blind signature σ_{BS} for com as follows:

$$(umsg_1, \mathsf{st}) \leftarrow \mathsf{BS}.\mathsf{U}_1(\mathsf{pk}_{\mathsf{BS}}, com) \quad (2)$$

$$\sigma_{\mathsf{BS}} \leftarrow \mathsf{BS}.\mathsf{U}_2(\mathsf{st}, smsg_1), \quad (3)$$

where $smsg_1$ is the server-side BS message computed as part of the oracle's reply to the query.

At the start of the game, the challenger initializes two empty lists L_{BS} and L_{com}. During execution, each pair (f, com) is stored in L_{com}, and each corresponding commitment–signature pair $(com, \sigma_{\mathsf{BS}})$ is stored in L_{BS}. In addition to pairs computed in the interactive $\mathcal{O}^{\mathsf{KeyGen}}$ oracle, the challenger *includes values computed during honest executions* of the KeyGen protocol when $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{HonestKG}}$.

When the adversary $\mathcal{A}$ outputs a tuple $(m_1, \sigma_1), \ldots, (m_s, \sigma_s)$ of $kAGS$ signatures, the challenger—after verifying their validity but before running the Trace algorithm—parses each signature to compile a list $\mathcal{L}_{\mathsf{pid}} = \{\mathsf{pid}_i \mid (\cdot, \mathsf{pid}_i) \Leftarrow \sigma_i, i = 1, \ldots, s\}$ of all public identities used by the adversary. Recall that each public identity pid is a pair consisting of a commitment and a blind signature over it. The challenger checks whether each $\mathsf{pid} \in \mathcal{L}_{\mathsf{pid}}$ appears in the list $\mathcal{L}_{\mathsf{BS}}$, and aborts the game (outputting 0) if this check fails. We denote this failure event by fail_{12}.

We now construct an adversary $\mathcal{C}$ against the strong one-more unforgeability of BS as follows. $\mathcal{C}$ simulates G_2 for $\mathcal{A}$, using the BS parameters received from its own challenger. When replying to $\mathcal{A}$'s $\mathcal{O}^{\mathsf{KeyGen}}$ or $\mathcal{O}^{\mathsf{HonestKG}}$ queries, $\mathcal{C}$ uses its own omUnforg server-side signing oracle to compute $smsg_1$. It then constructs the list $\mathcal{L}_{\mathsf{BS}}$ exactly as in G_2.

When $\mathcal{A}$ outputs its list of $kAGS$ signatures, the list $\mathcal{L}_{\mathsf{BS}}$ contains exactly as many message–signature pairs as the number of $\mathcal{C}$'s queries to its signing oracle. If the failure event fail_{12} occurs, this means $\mathcal{A}$ has provided at least one $\mathsf{pid} = (com, \mathsf{Sign}_{\mathsf{BS}})$ where σ_{BS} is a valid blind signature over com, that is not in $\mathcal{L}_{\mathsf{BS}}$. In this case, $\mathcal{C}$ outputs $\mathcal{L}_{\mathsf{BS}} \cup \mathsf{pid}$ in the omUnforg experiment and wins. If the failure event does not occur, $\mathcal{C}$ outputs $\mathcal{L}_{\mathsf{BS}}$, losing its game. Hence, the probability that $\mathcal{C}$ wins is at least the probability of fail_{12} in G_2.

Since G_1 and G_2 are identical when the failure event does not occur, we have:

$$\begin{aligned}\Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] &\leqslant \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] + \Pr[\mathsf{fail}_{12} \textit{ in } G_2(\mathcal{A}, 1^\lambda)] \\ &\leqslant \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{negl}^{\mathrm{omUnforg}}_{\mathsf{BS}}(\lambda). \quad (4)\end{aligned}$$

G_3: In this game, we continue making the signing oracle independent of the honest users' secret information. Instead of honestly computing a proof π_{S} in the oracle $\mathcal{O}^{\mathsf{Sign}}$, the challenger uses the simulator algorithm (recall that the challenger knows the trapdoor td, since it generates the secret key). Concretely, the challenger computes

$$\pi_{\mathsf{S}} \leftarrow \mathsf{NIZK}_{\mathsf{S}}.\mathsf{Sim}(\mathsf{crs}_{\mathsf{S}}, \mathsf{td}, (x, ct, com)).$$

In addition, the challenger now skips the computation of the polynomial commitment opening ω_{PC}, since it is no longer required for producing a proof.

Since Games G_2 and G_3 differ only in the distribution of NIZK proofs in honestly generated signatures, we obtain[10]

$$\Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] \leqslant \Pr[G_3(\mathcal{A}, 1^\lambda) \to 1] + q_{\mathsf{Sign}} \cdot \mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{S}}}(\lambda), \quad (5)$$

where q_{Sign} is the number of $\mathcal{A}$'s queries to the signing oracle.

G_4: By the simulation-extractability of $\mathsf{NIZK}_{\mathsf{S}}$, for the adversary $\mathcal{A}$ there exists a knowledge extractor $\mathcal{E}_{\mathcal{A}}$. In G_3, the challenger uses $\mathcal{E}_{\mathcal{A}}$ to modify the experiment as follows.

After $\mathcal{A}$ outputs challenge signatures, for each valid message-signature pair $(msg, \sigma) \in \{(msg_i, \sigma_i), i = 1, \ldots, s\}$, i.e., such that $\mathsf{NIZK}_{\mathsf{S}}.\mathsf{Verify}(\mathsf{crs}_{\mathsf{S}}, (x, ct, com), \pi_{\mathsf{S}}) = 1$ where $\sigma = (((x, ct), \pi_{\mathsf{S}}), (com, \sigma_{\mathsf{BS}}))$, the challenger uses $\mathcal{E}_{\mathcal{A}}$ to obtain a corresponding witness $w = (y, \omega_{\mathsf{PC}}, R)$ and checks whether $\mathcal{R}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{BS}}}((x, ct, com), w) = 1$. If this check fails, the challenger aborts (outputting 0). We denote that failure event by fail_{34}.

If the game does not abort, it holds that ω_{PC} is a valid PC opening of com at (x, y), and ct is a correctly computed PKE encryption of y. The challenger records all extracted triples $(y, \omega_{\mathsf{PC}}, R)$ in a list.

We now construct an adversary $\mathcal{B}$ against the knowledge-soundness of $\mathsf{NIZK}_{\mathsf{S}}$. $\mathcal{B}$ perfectly simulates G_3 for $\mathcal{A}$, using the $\mathsf{NIZK}_{\mathsf{S}}$ parameters received from its own challenger, it queries its simulation oracle to compute honest signatures. If the failure event occurs, $\mathcal{B}$ outputs $((x, ct, com), \pi_{\mathsf{S}})$ as its solution in the knowledge-soundness game and wins. Otherwise, $\mathcal{B}$ outputs random data. Hence, the probability that $\mathcal{B}$ wins the knowledge-soundness game is at least the probability of the failure event in G_3.

[10] Although we define ZK in the presence of a subverter, here it is enough to assume that the subverter is honest.

Since G_3 and G_4 behave identically unless the failure event occurs and $\mathsf{NIZK_S}$ is simulation-extractable, we have

$$\begin{aligned}\Pr[G_3(\mathcal{A},1^\lambda)\to 1] &\leqslant \Pr[G_4(\mathcal{A},1^\lambda)\to 1] + \Pr[\mathsf{fail}_{34}\ in\ G_4(\mathcal{A},1^\lambda)]\\ &\leqslant \Pr[G_4(\mathcal{A},1^\lambda)\to 1] + \mathsf{Adv}^{\mathrm{SE}}_{\mathsf{NIZK_S}}(\mathcal{B},\lambda) \leqslant \Pr[G_4(\mathcal{A},1^\lambda)\to 1] + \mathsf{negl}^{\mathrm{SE}}_{\mathsf{NIZK_S}}(\lambda). \quad (6)\end{aligned}$$

G_5: In this game, the challenger for each valid signature $\sigma_i = (((x, ct), \pi_\mathsf{S}, \mathsf{pid})$ checks if it holds that a PKE decryption of ct is equal to an y-coordinate previously extracted from π_S. If this check fails, the challenger aborts (outputting 0). We denote that failure event by fail_{45}.

If the check doesn't fail, the challenger replaces ct with the extracted y coordinate, and π_S with the polynomial commitment opening ω_PC. It also now skips decryption in Trace.

Due to the changes introduced in G_4, every ciphertext ct in each signature σ_i is a correctly computed PKE encryption of the extracted y-coordinate with randomness R. Hence, the failure event fail_{45} corresponds exactly to a violation of PKE correctness. Since G_4 and G_5 behave identically unless the failure event occurs, we have

$$\Pr[G_4(\mathcal{A},1^\lambda)\to 1] \leqslant \Pr[G_5(\mathcal{A},1^\lambda)\to 1] + \Pr[\mathsf{fail}_{45}\text{ in } G_5(\mathcal{A},1^\lambda)] \leqslant \Pr[G_5(\mathcal{A},1^\lambda)\to 1] + \mathsf{negl}^{\mathrm{corr}}_{\mathsf{PKE}}(\lambda). \quad (7)$$

G_6: This game is almost identical to G_5. Before running the Trace algorithm on the signatures output by the adversary, and after performing the checks introduced in the previous games, the challenger checks whether there exist two message–signature pairs (msg_i, σ_i) and (msg_j, σ_j) such that:

$$\begin{aligned}\sigma_i.\mathsf{pid} &= \sigma_j.\mathsf{pid},\\ msg_i &\neq msg_j,\\ \sigma_i.\mathsf{sh}.x &= \sigma_j.\mathsf{sh}.x.\end{aligned}$$

If such a pair is found, the challenger aborts the game (outputting 0). We denote this failure event by fail_{56}.

Recall that the x-coordinate of a share sh in a k-AGS signature over a message msg is computed as $x = \mathsf{H2F}(\mathsf{pid}\|msg)$. It follows immediately that the probability of fail_6 occurring is equal to the probability of a collision in two outputs of $\mathsf{H2F}$.

Since G_5 and G_6 behave identically unless the failure event occurs, we have:

$$\Pr[G_5(\mathcal{A},1^\lambda)\to 1] \leqslant \Pr[G_6(\mathcal{A},1^\lambda)\to 1] + \Pr[\mathsf{fail}_{56}\text{ in } G_6(\mathcal{A},1^\lambda)] \leqslant \Pr[G_6(\mathcal{A},1^\lambda)\to 1] + \mathsf{negl}^{\mathrm{cr}}_{\mathcal{HF}}(\lambda). \quad (8)$$

G_7: In G_7, the challenger, before running Trace, the challenger for each valid signature (after changes of game 5) $\sigma_i = (((x, y), \omega_\mathsf{PC}, \mathsf{pid} = (com, \sigma_\mathsf{BS}))$ checks if holds that $y = f(x)$, where f is a polynomial corresponding to commitment com retrieved from L_{com}. The challenger aborts the game (outputting 0) if this check fails. We denote this failure event by fail_{67}.

We now construct an adversary $\mathcal{D}$ against the binding property of PC. $\mathcal{D}$ perfectly simulates G_7 for $\mathcal{A}$, using the commitment key ck received from its own challenger. If the failure event occurs, then one of the signatures satisfies $f(x) \neq y$, even though the extracted opening ω_PC is a valid opening for com at (x, y). In that case, $\mathcal{D}$ computes $\omega_{real} = \mathsf{PC.Open}(\mathsf{ck}, f, x, f(x))$, and outputs the tuple $(com, x, y, f(x), \omega_\mathsf{PC}, \omega_{real})$. Otherwise, it fails its own experiment. Hence, the probability that $\mathcal{D}$ wins is at least the probability of fail_{67} in G_7.

Since G_6 and G_7 are identical unless this failure event occurs, we have:

$$\Pr[G_6(\mathcal{A},1^\lambda)\to 1] \leqslant \Pr[G_7(\mathcal{A},1^\lambda)\to 1] + \Pr[\mathsf{fail}_{67}\text{ in } G_7(\mathcal{A},1^\lambda)] \leqslant \Pr[G_7(\mathcal{A},1^\lambda)\to 1] + \mathsf{negl}^{\mathrm{Bind}}_{\mathsf{PC}}(\lambda). \quad (9)$$

G_8: In this game, the challenger in $\mathcal{O}^{\mathsf{Sign}}$ no longer computes $y = f(x)$, but instead chooses y uniformly at random from $\mathbb{Z}_p$. It computes a simulated NIZK proof π_S for (x, ct) as before. Thus, the honest signing oracle is no longer dependent on any user's secret polynomial f.

We now construct an adversary $\mathcal{B}$ against the INDCPA security of PKE. The adversary $\mathcal{B}$ simulates Game G_7 for $\mathcal{A}$, using the PKE parameters received from its own challenger. Whenever $\mathcal{B}$ needs to compute an honest signature, it queries its encryption oracle with the pair $(f(x), y')$, where y' is chosen randomly. The resulting ciphertext is then included in the signature. The adversary makes at most q_{Sign} queries to its oracle. Note that the decryption oracle is never required, due to the modification introduced in Game G_5.

It is clear that if the bit b in $\mathcal{B}$'s game is 0, then $\mathcal{B}$ perfectly simulates Game G_7 for $\mathcal{A}$. Conversely, if $b = 1$, it simulates Game G_8. Hence, we obtain

$$\Pr[G_7(\mathcal{A}, 1^\lambda) \to 1] \leqslant \Pr[G_8(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{negl}^{\mathrm{INDCPA}}_{\mathsf{PKE}}(\lambda). \tag{10}$$

G_9: In this game, the challenger checks whether any valid signature output by $\mathcal{A}$ contains a pid corresponding to an honest user. (Note that, due to Game 2, pseudo-identities may belong only to adversarially controlled users or honest users.) Due to the changes introduced in previous games, it holds that in such signatures, the y-coordinate of a share is an evaluation of the user's secret polynomial f at some $x \neq 0$. The challenger aborts the game (outputting 0) if this check fails. We denote this failure event by fail_{89}.

We now construct an adversary $\mathcal{B}$ against the (computational version of) hiding of PC. The adversary $\mathcal{B}$ starts by perfectly simulating G_8 for $\mathcal{A}$. Before allowing $\mathcal{A}$ to make oracle queries, $\mathcal{B}$ chooses a random index i from 1 to q_{HonestKG}, where q_{HonestKG} is the maximum number of honest users that $\mathcal{A}$ is allowed to register. When $\mathcal{A}$ queries to register the i-th honest user, $\mathcal{B}$ sends the tuple $(\mathsf{ck}, k, (0, \mathsf{id}))$, along with its state, to its own experiment and receives a commitment *com* in response. It then uses *com* to form this user's pid. Note that $\mathcal{B}$ does not need to query its own oracle to compute honest signatures for that user, due to the changes introduced in Games 3 and 8.

If the failure event occurs, $\mathcal{B}$ checks whether the pid of the signature that triggered the event corresponds to the challenge pid it computed using its own challenge. If not, it explicitly fails its own experiment (it does so if the failure event does not occur as well). If it does match, $\mathcal{B}$ outputs the pair (x, y) and wins the game. Hence, the probability of $\mathcal{B}$'s success is exactly $\frac{1}{q_{\mathsf{HonestKG}}}$ times the probability of the failure event. We have:

$$\begin{aligned}\Pr[G_9(\mathcal{A}, 1^\lambda) \to 1] \leq \Pr[G_8(\mathcal{A}, 1^\lambda) \to 1] + \Pr[\mathsf{fail}_{89} \text{ in } G_9(\mathcal{A}, 1^\lambda)] \\ \leq \Pr[G_8(\mathcal{A}, 1^\lambda) \to 1] + q_{\mathsf{HonestKG}} \cdot \mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda).\end{aligned} \tag{11}$$

Analysis of G_9: Let L'_{BS} and L'_{com} be the subsets of L_{BS} and L_{com}, respectively, that contain only entries added as a result of adversarial KeyGen sessions. If all the additional checks introduced in the previous games are passed, the following holds:

1. The number of message–signature pairs in L'_{BS} and the number of pairs in L'_{com} are both equal to the number of adversarial KeyGen sessions that resulted in accept. Moreover, for each $(f, \cdot) \in L'_{com}$, it holds that $f(0) \in L'_{\mathsf{id}}$ (due to G_1).

2. The number of distinct public identities pid appearing in the signatures output by the adversary is at most the number of message–signature pairs in L'_{BS}. Moreover, each such pid is in L'_{BS}, and each commitment in a public identity has a corresponding entry in L'_{com} (due to G_2 and the abort event in G_9).

3. For each distinct pid, there are no two signatures that have the same x-coordinate in the share $\sigma.\mathsf{sh}$ (due to G_6).

4. For every signature with commitment $\mathsf{pid}.com$, it holds that $y = f(x)$, where f is the polynomial corresponding to *com* from L'_{com} (due to G_7).

5. The number of signatures output by the adversary is at least $k \cdot usr + 1$, where usr is the number of successful adversarial KeyGen sessions (due to the check in the original game G_0).

From these properties, it follows that there exists some public identity pid which is used in at least $k+1$ signatures, and each share sh in these signatures has a distinct x-coordinate. In G_9, the challenger, while running Trace, finds such a pid, parses the corresponding signatures to obtain distinct (due to item 3) shares $(x_1, y_1), \ldots, (x_{k+1}, y_{k+1})$, and interpolates a polynomial f' over these points. Clearly (due to item 4), f' is equal to the polynomial f retrieved from L'_{com} for commitment $\mathsf{pid}.com$, and $f(0) \in L_{\mathsf{id}}$ (by item 1). Hence, if the game has not already aborted, Trace will always output some $\mathsf{id} \in L_{\mathsf{id}}$, thereby causing the adversary to lose. We have:

$$\Pr[G_9(\mathcal{A}, 1^\lambda) \to 1] = 0.$$

Finally, by combining inequalities (1)–(11), we conclude the proof of the theorem. □

B.2 k-Anonymity

Theorem 8 *Let $\mathcal{A}$ be an adversary against the k-Anonymity of the k-AGS scheme $\Pi[\mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$. Let $\mathsf{NIZK}_{\mathsf{KG}}$ be a subversion-zero-knowledge NIZK scheme, PC a hiding polynomial commitment scheme, and BS a secure blind signature scheme. Then, there exist negligible functions $\mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}}(\lambda)$, $\mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda)$, and $\mathsf{negl}^{\mathrm{Blind}}_{\mathsf{BS}}(\lambda)$ such that*

$$\mathsf{Adv}^{\mathrm{kAnon}}_{\Pi}(\mathcal{A}, \lambda) \leqslant 4 \cdot \mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda) \;+\; 4 \cdot \mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda) \;+\; \mathsf{negl}^{\mathrm{Blind}}_{\mathsf{BS}}(\lambda).$$

Proof. Since the k-Anonymity experiment is defined with the group manager as the adversary, we immediately consider a modification of Π: instead of including an encryption of the y-coordinate of a share in a signature, the signature directly contains the y-coordinate, as well as the polynomial commitment ω_{PC} in place of π_{S}. In other words, for k-Anonymity we do not rely on any properties of PKE or $\mathsf{NIZK}_{\mathsf{S}}$.

We construct a sequence of indistinguishable games $G_0, \ldots, G_8$, where G_0 is the experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}(1^\lambda, \mathcal{A})$ with bit $b = 0$, and G_8 is the experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}(1^\lambda, \mathcal{A})$ with bit $b = 1$.

G_1: To define G_1, we first construct a subverter $\mathcal{S}$ for the NIZK scheme $\mathsf{NIZK}_{\mathsf{KG}}$ from $\mathcal{A}$. The subverter $\mathcal{S}$ starts $\mathcal{A}$ and uses the $\mathsf{crs}_{\mathsf{KG}}$ output by $\mathcal{A}$ (contained in the master public key mpk) as its own output. It includes all other outputs of $\mathcal{A}$ in its state st, together with $\mathcal{A}$'s internal state and outputs it along with the random tape used. By the definition of subversion zero-knowledge, there exists a negligible function $\mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda)$ and a PPT simulation trapdoor extractor $\mathcal{E}_{\mathcal{S}}$ such that, for every $\lambda \in \mathbb{N}$, every $\mathcal{R} \in \mathsf{RelGen}(1^\lambda)$, and every PPT adversary $\mathcal{B}$,

$$\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}((\mathcal{S}, \mathcal{B}), \lambda) \leqslant \mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda).$$

In G_1, right after starting $\mathcal{A}$, the challenger checks that the $\mathsf{crs}_{\mathsf{KG}}$ is well formed using $\mathsf{NIZK}_{\mathsf{KG}}.\mathsf{CVerify}$, and extracts a trapdoor td using $\mathcal{E}_{\mathcal{S}}$. The KeyGen oracle is then modified as follows:

In both KeyGen sessions, when computing the first client-side message in response to a corresponding oracle query, the challenger no longer generates the proof

$$\pi_{\mathsf{KG}} = \mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{KG}}, (umsg_1, \mathsf{id}), (f, R))$$

as in the original scheme Π. Instead, it uses the NIZK simulator:

$$\pi_{\mathsf{KG}} = \mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Sim}(\mathsf{crs}_{\mathsf{KG}}, \mathsf{td}, (umsg_1, \mathsf{id})).$$

For the convenience of establishing "indistinguishability" of the games G_0 and G_1, we introduce an intermediate game G'_1, in which the proof is simulated only in the first KeyGen session, while the second remains unchanged.

We now construct two adversaries, $\mathcal{B}_1$ and $\mathcal{B}_2$, and corresponding pairs $(\mathcal{S}, \mathcal{B}_1)$ and $(\mathcal{S}, \mathcal{B}_2)$ against the subversion zero-knowledge of $\mathsf{NIZK}_{\mathsf{KG}}$ and extractor $\mathcal{E}_{\mathcal{S}}$. In both $\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_1))$ and $\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_2))$, the subverter $\mathcal{S}$ first runs $\mathcal{A}$, outputs $\mathsf{crs}_{\mathsf{KG}}$, and stores the remaining outputs (including $\mathcal{A}$'s state $\mathsf{st}_{\mathcal{A}}$) as its own state $\mathsf{st}_{\mathcal{S}}$.

Each adversary then reconstructs the master public key from $\mathsf{st}_{\mathcal{S}}$, resumes $\mathcal{A}$ with state $\mathsf{st}_{\mathcal{A}}$, and continues the simulation. The distinction is as follows. $\mathcal{B}_1$ simulates G_0, except that in the first KeyGen session it forwards the statement–witness pair $((umsg_1, \mathsf{id}), (f, R))$ to its own challenger and uses the returned proof π as a reply to $\mathcal{A}$'s query. $\mathcal{B}_2$ behaves analogously, but does so in the second KeyGen session, while the first session is handled as in G_1 (with a simulated proof).

Finally, both adversaries output whatever $\mathcal{A}$ outputs.

It follows that $(\mathcal{S}, \mathcal{B}_1)$ perfectly simulates G_0 if the bit $b = 0$ in its ZK experiment, since the proof received form the challenger is a real one, and G_1' if $b = 1$, since the proof is simulated. Analogously, $(\mathcal{S}, \mathcal{B}_2)$ perfectly simulates G_1' if $b = 0$, and G_1 if $b = 1$. Hence,

$$
\begin{aligned}
&\Pr[G_0(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] \\
&= \Pr[G_0(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_1'(\mathcal{A}, 1^\lambda) \to 1] + \Pr[G_1'(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] \\
&= \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_1)) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_1)) \to 1 \mid b = 1] \\
&\quad + \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_2)) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}(1^\lambda, (\mathcal{S}, \mathcal{B}_2)) \to 1 \mid b = 1] \\
&= \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}((\mathcal{S}, \mathcal{B}_1), \lambda) + \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}, \mathcal{E}_{\mathcal{S}}}((\mathcal{S}, \mathcal{B}_2), \lambda) \leqslant 2 \cdot \mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda). \qquad (12)
\end{aligned}
$$

G_2: In this game, in both KeyGen sessions initiated by adversarial queries to the oracle, the challenger modifies the way the first client-side message is computed. Whenever the adversary submits a query that contains a user identity id, the challenger instead chooses a fresh identity id^{new}, and uses it to generate a secret polynomial f. It then computes a polynomial commitment com to f, and the first message $umsg_1$ of the blind signature scheme. At the same time, the challenger still uses the adversarially chosen id to compute a simulated proof. The resulting user secret key consists of the polynomial f associated with the fresh identity id^{new}, and a public identity pid formed by the commitment com and a blind signature σ_{BS} on com. These values, being part of the secret key, are also used to answer signing queries through the Sign oracle. In other words, instead of running $\mathsf{KeyGen}.\mathsf{C}_1(\mathsf{id})$, the challenger executes the modified algorithm $\tilde{\mathsf{C}}_1$ given below.

$\tilde{\mathsf{C}}_1(\mathsf{id})$

1 : $\mathsf{id}^{new} \leftarrow \mathbb{Z}_p$
2 : $\mathsf{ck}, \mathsf{pk}_{\mathsf{BS}}, \mathsf{crs}_{\mathsf{KG}} \Leftarrow \mathsf{mpk}$
3 : $f \xleftarrow{\$} \{f \in \mathbb{Z}_p[x] : \deg(f) = k \wedge f(0) = \mathsf{id}^{new}\}$
4 : $com \leftarrow \mathsf{PC.Commit}(\mathsf{ck}, f)$
5 : $(umsg_1, \mathsf{st}_{\mathsf{BS}}; R) \leftarrow \mathsf{BS.U}_1(\mathsf{pk}_{\mathsf{BS}}, com)$
6 : $\pi_{\mathsf{KG}} \leftarrow \mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Sim}(\mathsf{crs}_{\mathsf{KG}}, \mathsf{td}, (umsg_1, \mathsf{id}))$
7 : $\mathsf{st} \leftarrow (\mathsf{id}^{new}, f, \mathsf{st}_{\mathsf{BS}})$
8 : **return** $\mathsf{st}, umsg_1, \pi_{\mathsf{KG}}$

We also define an intermediate game G_2', where a fresh identity id^{new} is used only in the first KeyGen session, while in the second session the challenger proceeds with the adversarially chosen identity id.

We now construct two adversaries, $\mathcal{C}_1$ and $\mathcal{C}_2$, against the hiding property of the polynomial commitment scheme PC.

Adversary $\mathcal{C}_1$ simulates the game G_1 for $\mathcal{A}$. It begins by parsing the master public key mpk, output by $\mathcal{A}$, in order to extract the commitment key ck. When $\mathcal{A}$ initiates the first KeyGen session with an identity id, $\mathcal{C}_1$ outputs $(\mathsf{ck}, k, (0, \mathsf{id}))$ and its own internal state as input to its hiding experiment, and receives in return a commitment com. Using this com, $\mathcal{C}_1$ runs the blind-signature protocol to obtain a blind signature, computes the simulated zero-knowledge proof, and forms the corresponding public identity pid_1 for the first user. For the second KeyGen session, $\mathcal{C}_1$ proceeds exactly as in G_2, i.e., it executes the KeyGen protocol using the adversary's chosen identity id, but always replaces the zero-knowledge proof with a simulated one.

When $\mathcal{A}$ queries the signing oracle for the first user with message m, $\mathcal{C}_1$ needs to compute a signature share sh and opening ω_{PC}, which are derived from the polynomial f. To do this, it queries its own Open oracle at $\mathsf{H2F}(\mathsf{pid}_1 \| m)$. For the second user, $\mathcal{C}_1$ answers signing queries according to the scheme.

If in $\mathcal{C}_1$'s hiding experiment the challenge bit $b = 0$, then the commitment com and its openings correspond to a polynomial f such that $f(0) = \mathsf{id}$. In this case, $\mathcal{C}_1$ provides a perfect simulation of G_1. If $b = 1$, then the commitment corresponds to a polynomial f with $f(0)$ equal to a random value independent of id, which yields a perfect simulation of G_2'. In either case, $\mathcal{C}_1$ outputs the same result as $\mathcal{A}$.

The adversary $\mathcal{C}_2$ is defined in a completely analogous way, but it instead relates G_2' and G_2. Specifically, $\mathcal{C}_2$ simulates G_2' for $\mathcal{A}$, meaning that in the first KeyGen session it uses a fresh identity id^{new}, while in the second KeyGen session it follows the protocol with the adversary's chosen identity. However, $\mathcal{C}_2$ is also given a polynomial commitment by its challenger, and uses it to form the corresponding public identity and to answer signing queries (through its Open oracle), just as $\mathcal{C}_1$ does for the first user. If $b = 0$, $\mathcal{C}_2$'s simulation is indistinguishable from G_2'; if $b = 1$, it simulates G_2. Again, $\mathcal{C}_2$ outputs whatever $\mathcal{A}$ outputs.

Putting these together, we obtain:

$$\begin{aligned}
&\Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] \\
&= \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_2'(\mathcal{A}, 1^\lambda) \to 1] + \Pr[G_2'(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] \\
&= \Pr[\mathbf{Exp}_{\mathsf{PC}}^{\text{Hiding}}(1^\lambda, \mathcal{C}_1) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}_{\mathsf{PC}}^{\text{Hiding}}(1^\lambda, \mathcal{C}_1) \to 1 \mid b = 1] \\
&\quad + \Pr[\mathbf{Exp}_{\mathsf{PC}}^{\text{Hiding}}(1^\lambda, \mathcal{C}_2) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}_{\mathsf{PC}}^{\text{Hiding}}(1^\lambda, \mathcal{C}_2) \to 1 \mid b = 1] \\
&= \mathsf{Adv}_{\mathsf{PC}}^{\text{Hiding}}(\mathcal{C}_1, \lambda) + \mathsf{Adv}_{\mathsf{PC}}^{\text{Hiding}}(\mathcal{C}_2, \lambda) \leqslant 2 \cdot \mathsf{negl}_{\mathsf{PC}}^{\text{Hiding}}(\lambda),
\end{aligned} \tag{13}$$

where the last inequality follows from the hiding property of PC.

G_3: In this game, the challenger slightly changes the way in which user identities and the associated polynomials are generated. After $\mathcal{A}$ is initialized, but before it is given access to the KeyGen oracle, the challenger samples two fresh identities id_0^{new} and id_1^{new}. For each of these identities it chooses a polynomial of degree k over $\mathbb{Z}_p$ that evaluates to the corresponding identity at 0. It then computes two commitments, com_0 and com_1, to these polynomials using the commitment key ck that $\mathcal{A}$ provided in the master public key. These two commitments are fixed for the rest of the game. Whenever the adversary queries the KeyGen oracle, the challenger no longer generates a polynomial commitment on the fly, but instead uses the precomputed one: for the first user it uses com_0 and for the second user it uses com_1. In both cases, the challenger engages in the blind-signature protocol with $\mathcal{A}$ to produce the corresponding blind signature and thus form the public identity pid. The same two precomputed polynomials are also used by the challenger to answer signing queries via the Sign oracle.

This modification is purely technical: it only changes the point in time when the two polynomials and their commitments are sampled, without altering the distribution of the resulting values or the view of the adversary. Therefore,

$$\Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] = \Pr[G_3(\mathcal{A}, 1^\lambda) \to 1].$$

G_4: In this game, the challenger sets the bit b to 1. In other words, the challenger still uses id_i^{new} together with the corresponding polynomial f_i and commitment com_i in the KeyGen session initiated by the adversary $\mathcal{A}$ for user $i \in \{0, 1\}$, obtaining the secret key sk_i as the outcome of that session. At the same time, the roles of the keys in the signing oracle are swapped: when $\mathcal{A}$ requests a signature for user $i = 0$, the challenger responds using sk_1, and when $\mathcal{A}$ requests a signature for user $i = 1$, it responds using sk_0.

We now construct an adversary $\mathcal{D}$ against the blindness of BS. The adversary $\mathcal{D}$ simulates the execution of G_3 for $\mathcal{A}$. After initializing $\mathcal{A}$ and generating the fresh identities together with the corresponding commitments com_0 and com_1, $\mathcal{D}$ outputs $(\mathsf{pk}_{\mathsf{BS}}, com_0, com_1)$ along with its internal state to its own experiment, where $\mathsf{pk}_{\mathsf{BS}}$ is parsed from the master public key generated by $\mathcal{A}$, and (com_0, com_1) serves as the challenge message pair in the blindness experiment.

Whenever $\mathcal{A}$ initiates a KeyGen session for user i, $\mathcal{D}$ queries its own oracles in order to compute the blind-signature messages. Since $\mathcal{D}$ does not receive the blind signatures (σ_0, σ_1) from its experiment until both KeyGen sessions are completed, it postpones the formation of the secret keys until that point. Once the signatures are obtained, $\mathcal{D}$ sets $\mathsf{sk}_i.\mathsf{pid} = (com_i, \sigma_i)$ and uses the resulting sk_i to respond to signing queries of $\mathcal{A}$ for user i.

If in $\mathcal{D}$'s experiment the bit b is 0, then in the first KeyGen session com_0 is blind signed, and the corresponding secret key sk_0 belongs to the first session, while sk_1 belongs to the second. In this case, $\mathcal{D}$ perfectly simulates the game G_3 for $\mathcal{A}$.

If instead the bit b in $\mathcal{D}$'s experiment is 1, then in the first KeyGen session com_1 is blind signed, and thus sk_1 belongs to the first session while sk_0 belongs to the second. At the same time, the use of the secret keys in the signing oracle is not swapped: when $\mathcal{A}$ queries signing for user i, the challenger still responds using sk_i. Consequently, the secret key generated in the first KeyGen session is used for signing on behalf of the second user, and vice versa. This matches exactly the behavior of the challenger in G_4. Hence, in this case $\mathcal{D}$ perfectly simulates G_4 for $\mathcal{A}$.

It follows that

$$\begin{aligned}
&\Pr[G_3(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_4(\mathcal{A}, 1^\lambda) \to 1] \\
&= \Pr[\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}(1^\lambda, \mathcal{D}) \to 1 \mid b = 0] - \Pr[\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS}}(1^\lambda, \mathcal{D}) \to 1 \mid b = 1] \\
&= \mathsf{Adv}^{\mathrm{Blind}}_{\mathsf{BS}}(\mathcal{D}, \lambda) \leqslant \mathsf{negl}^{\mathrm{Blind}}_{\mathsf{BS}}(\lambda).
\end{aligned} \tag{14}$$

G_5, G_6, G_7: In these games, the modifications introduced in games G_3, G_2, and G_1 are successively reverted. Thus, G_7 corresponds exactly to the experiment $\mathbf{Exp}^{\mathrm{kAnon}}_{\Pi}(1^\lambda, \mathcal{A})$ with the challenge bit fixed to $b = 1$. Therefore, we obtain

$$\Pr[G_4(\mathcal{A}, 1^\lambda) \to 1] - \Pr[G_7(\mathcal{A}, 1^\lambda) \to 1] \leqslant 2 \cdot \mathsf{negl}^{\mathrm{ZK}}_{\mathsf{NIZK}_{\mathsf{KG}}}(\lambda) + 2 \cdot \mathsf{negl}^{\mathrm{Hiding}}_{\mathsf{PC}}(\lambda). \tag{15}$$

Finally, by combining the bounds established in the transitions from G_0 through G_7, we conclude the proof of the theorem. □

C k-ASPCGS Construction

In this section, we introduce our construction $\Pi[k, n, \mathcal{M}, \mathcal{ID}]$ of a k-ASPCGS for an arbitrary message space $\mathcal{M}$ and an identity space $\mathcal{ID} = \mathbb{Z}_p$. We make use of the following primitives in a black-box manner:

- a polynomial commitment scheme PC for polynomials over $\mathbb{Z}_p$;
- a two-round blind signature scheme BS;
- a set pre-constrained encryption scheme SPCE over a universe $\mathcal{M}$, with the message space $\mathbb{Z}_p$
- a hash to field function $\mathcal{HF}$ from the message space $\mathcal{M}$ to $\mathbb{Z}_p$;
- a subversion-resistant non-interactive zero-knowledge argument system $\mathsf{NIZK}_{\mathsf{KG}}$ for a relation generator $\mathsf{RelGen}(1^\lambda)$, where the generator outputs relations $\mathcal{R}^{\mathsf{KG}}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{BS}}}$ corresponding to languages of the form[11]:

$$\mathcal{L} = \left\{ (msg, \mathsf{id}) \;\middle|\; \exists f \in \mathbb{Z}_p[x],\ R\colon \begin{array}{c} f(0) = \mathsf{id},\ \ com = \mathsf{PC.Commit}(\mathsf{ck}, f), \\ msg = \mathsf{BS.U}_1(\mathsf{pk}_{\mathsf{BS}}, com; R) \end{array} \right\}.$$

[11] Formally, $\mathsf{RelGen}(1^\lambda)$ runs $\mathsf{PC.Setup}(1^\lambda, k)$ and $\mathsf{BS.KeyGen}(1^\lambda)$ to generate such a relation.

- a subversion-resistant non-interactive zero-knowledge argument system $\mathsf{NIZK}_{\mathsf{S}}$ for a relation generator $\mathsf{RelGen}_{\mathsf{S}}(1^\lambda)$, where the generator outputs relations $\mathcal{R}^{\mathsf{S}}_{k,\mathcal{D},\mathsf{ck},\mathsf{pk}_{\mathsf{SPCE}}}$ corresponding to languages of the form[12]:

$$\mathcal{L} = \left\{ (x, ct, com) \;\middle|\; \exists y \in \mathbb{Z}_p,\ \omega_{\mathsf{PC}},\ R\colon \begin{array}{c} ct = \mathsf{SPCE.Enc}(\mathsf{pk}_{\mathsf{SPCE}}, msg, y; R) \\ \mathsf{PC.Verify}(\mathsf{ck}, com, \omega_{\mathsf{PC}}, x, y) = 1 \end{array} \right\}.$$

The algorithms and the key generation protocol of Π are defined in Figure 17 and Figure 18 correspondingly. The master key generation algorithm Gen selects a hash function $\mathsf{H2F}$ from $\mathcal{HF}$ and includes its description in the public key—this step is omitted from the pseudocode for brevity. In the Trace algorithm, $\mathsf{Interpolate}$ denotes any efficient polynomial interpolation procedure that, given $d+1$ points $(x_i, y_i) \in \mathbb{Z}_p^2$, outputs a polynomial $f \in \mathbb{Z}_p[x]$ of degree at most d.

The key generation protocol KeyGen is specified by four algorithms: $\{\mathsf{GM}_1, \mathsf{C}_1, \mathsf{GM}_2, \mathsf{C}_2\}$. For the sake of compactness, we omit in the pseudocode the details of how the output states of GM_1 and C_1 are constructed and later parsed by GM_2 and C_2. We simply assume that the states contain the intermediate values required for subsequent computations. The output messages of GM_1, C_1, and GM_2 are shown above the arrows.

$\mathsf{Gen}(1^\lambda, k, \mathcal{D})$

$\mathsf{ck} \leftarrow \mathsf{PC.Setup}(1^\lambda, k)$
$(\mathsf{pk}_{\mathsf{BS}}, \mathsf{sk}_{\mathsf{BS}}) \leftarrow \mathsf{BS.KeyGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{SPCE}}, \mathsf{sk}_{\mathsf{SPCE}}) \leftarrow \mathsf{SPCE.Gen}(1^\lambda, \mathcal{D})$
$\mathsf{crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^{\mathsf{KG}}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{BS}}})$
$\mathsf{crs}_{\mathsf{S}} \leftarrow \mathsf{NIZK}_{\mathsf{S}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^{\mathsf{S}}_{k,\mathsf{ck},\mathsf{pk}_{\mathsf{SPCE}}})$
$\mathsf{mpk} \leftarrow (\mathsf{ck}, \mathsf{pk}_{\mathsf{BS}}, \mathsf{pk}_{\mathsf{SPCE}}, \mathsf{crs}_{\mathsf{KG}}, \mathsf{crs}_{\mathsf{S}})$
$\mathsf{msk} \leftarrow \mathsf{sk}_{\mathsf{BS}}$
return $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{Trace}(\mathsf{msk}, k, \{\sigma_i\}_{i \in \mathcal{I}})$

$\mathsf{sk}_{\mathsf{SPCE}} \Leftarrow \mathsf{msk}$
$L_{\mathsf{sh}} \leftarrow \emptyset$
$L_{\mathsf{id}} \leftarrow \emptyset$
for $i \in \mathcal{I}$:
  $((x, ct), \pi_{\mathsf{KG}}), \mathsf{pid}) \Leftarrow \sigma_i$
  $y \leftarrow \mathsf{SPCE.Dec}(\mathsf{sk}_{\mathsf{SPCE}}, ct)$
  if $(x, y) \notin L_{\mathsf{sh}}[\mathsf{pid}]$:
    $\mathcal{L}_{\mathsf{sh}}[\mathsf{pid}] \leftarrow L_{\mathsf{sh}}[\mathsf{pid}] \cup (x, y)$
for $shares \in L_{\mathsf{sh}}$:
  if $|shares| \geq k + 1$:
    $f \leftarrow \mathsf{Interpolate}(shares)$
    $\mathsf{id} \leftarrow f(0)$
    $L_{\mathsf{id}} \leftarrow L_{\mathsf{id}} \cup \mathsf{id}$
return L_{id}

$\mathsf{Sign}(\mathsf{mpk}, \mathsf{sk}, msg)$

$(\mathsf{ck}, \mathsf{pk}_{\mathsf{SPCE}}, \mathsf{crs}_{\mathsf{S}}) \Leftarrow \mathsf{mpk}$
$(f, \mathsf{pid}) \Leftarrow \mathsf{sk}$
$x \leftarrow 0, i \leftarrow 0$
while $x = 0$:
  $x \leftarrow \mathsf{H2F}(\mathsf{pid} \| msg \| i)$
  $i \leftarrow i + 1$
$\omega_{\mathsf{PC}} \leftarrow \mathsf{PC.Open}(\mathsf{ck}, f, x, f(x))$
$(ct; R) \leftarrow \mathsf{SPCE.Enc}(\mathsf{pk}_{\mathsf{SPCE}}, msg, f(x))$
$\pi_S \leftarrow \mathsf{NIZK}_{\mathsf{S}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{S}}, (x, ct, com), (f(x), \omega_{\mathsf{PC}}, R))$
$\mathsf{sh} \leftarrow (x, ct)$
$\sigma \leftarrow ((\mathsf{sh}, \pi_{\mathsf{S}}), \mathsf{pid})$
return σ

$\mathsf{Verify}(\mathsf{mpk}, msg, \sigma)$

$(\mathsf{crs}_{\mathsf{KG}}, \mathsf{pk}_{\mathsf{BS}}) \Leftarrow \mathsf{mpk}$
$((\mathsf{sh}, \pi_{\mathsf{KG}}), \mathsf{pid}) \Leftarrow \sigma$
$(x, ct) \Leftarrow \mathsf{sh}$
$(com, \sigma_{\mathsf{BS}}) \Leftarrow \mathsf{pid}$
// Verify pseudoidentity
if $\mathsf{BS.Verify}(\mathsf{pk}_{\mathsf{BS}}, \sigma_{\mathsf{BS}}, com) = 0$:
  return 0
// Verify share computed correctly
if $x \neq \mathsf{H2F}(\mathsf{pid} \| msg)$:
  return 0
if $\mathsf{NIZK}_{\mathsf{KG}}.\mathsf{Verify}(\mathsf{crs}_{\mathsf{KG}}, (x, ct, com), \pi_{\mathsf{KG}}) = 0$:
  return 0
return 1

Figure 17: k-ASPCGS construction $\Pi[k, n, \mathcal{M}, \Pi]$

[12]Formally, $\mathsf{RelGen}(1^\lambda)$ samples some set $\mathcal{D}, |\mathcal{D}| = n$, runs $\mathsf{PC.Setup}(1^\lambda, k)$ and $\mathsf{SPCE.KeyGen}(1^\lambda, \mathcal{D})$ to generate such a relation.

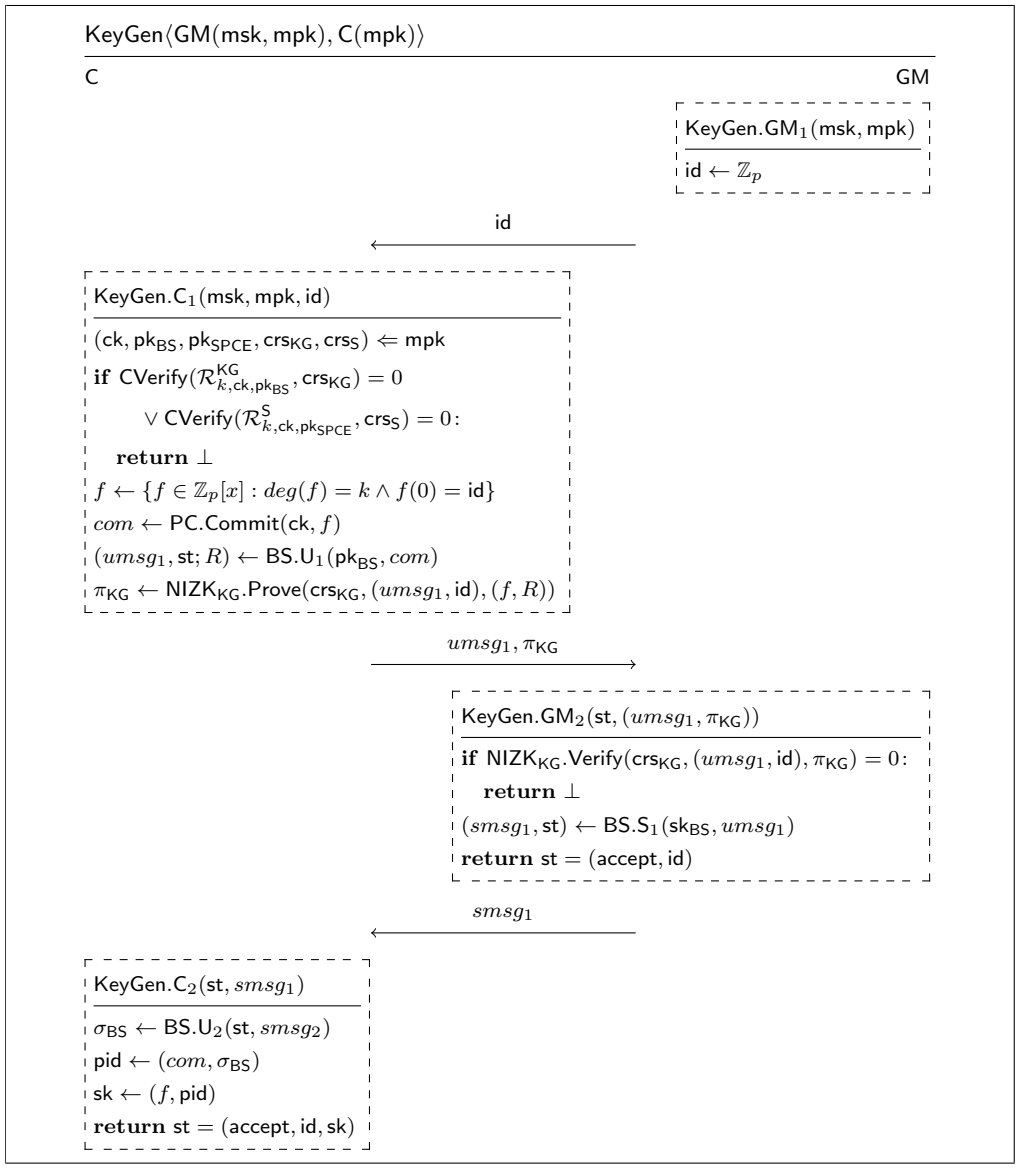

Figure 18: KeyGen protocol of k-ASPCGS $\Pi[k, n, \mathcal{M}, \Pi]$

D k-ASPCGS Generic Transform Security Proofs

Theorem 9 *Let $\mathcal{A}$ be an adversary against the traceability of a k-ASPCGS scheme $\Pi[k, n, \mathcal{M}, \mathcal{ID} = \mathbb{Z}_p]$. Let* NIZK *be a knowledge-sound NIZK scheme, let* SPCE *be a correct set pre-constrained encryption scheme, and let* kAGS *be a traceable k-anonymous group signature scheme. Then there exists negligible functions $\mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}}(\lambda)$, $\mathsf{negl}^{\mathrm{corr}}_{\mathsf{SPCE}}(\lambda)$, and $\mathsf{negl}^{\mathsf{Trace}}_{\mathsf{kAGS}}(\lambda)$ such that*

$$\mathsf{Adv}^{\mathrm{Trace}}_{\Pi}(\mathcal{A}, \lambda) \leqslant \mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}}(\lambda) + \mathsf{negl}^{\mathrm{corr}}_{\mathsf{SPCE}}(\lambda) + \mathsf{negl}^{\mathsf{Trace}}_{\mathsf{kAGS}}(\lambda).$$

Proof. (Sketch) We construct a sequence of indistinguishable games G_0, G_1, G_2, where G_0 corresponds exactly to the experiment $\mathbf{Exp}^{\mathrm{Trace}}_{\Pi}(1^\lambda, \mathcal{A})$, and G_2 is equivalent to the traceability game for the underlying kAGS scheme.

G_1: By the knowledge-soundness property of NIZK, for the adversary $\mathcal{A}$ there exists a knowledge extractor $\mathcal{E}_{\mathcal{A}}$. In G_1, the challenger uses $\mathcal{E}_{\mathcal{A}}$ to modify the experiment as follows.

After $\mathcal{A}$ outputs challenge signatures, for each valid message-signature pair $(msg, \sigma) \in \{(msg_i, \sigma_i), i = 1, \ldots, s\}$, i.e., such that $\mathsf{NIZK.Verify}(\mathsf{crs}, (msg, ct), \pi) = 1$ where $\sigma = (ct, \pi)$, the challenger uses $\mathcal{E}_{\mathcal{A}}$ to obtain a corresponding witness $w = (\sigma_{\mathsf{kAGS}}, R)$ and checks whether $\mathcal{R}_{k,\mathsf{ck},\mathsf{pk}_{BS}}((msg, ct), w) = 1$. If this check fails, the challenger aborts (outputting 0). We denote that failure event by fail_{01}.

If the game does not abort, it holds that σ_{kAGS} is a correct kAGS signature of msg, and ct is a correctly computed SPCE encryption of that signature with respect to msg. The challenger records all extracted signatures in a list.

We now construct an adversary $\mathcal{B}$ against the knowledge-soundness of NIZK. $\mathcal{B}$ perfectly simulates G_1 for $\mathcal{A}$, using the NIZK parameters received from its own challenger. If the failure event occurs, $\mathcal{B}$ outputs $((msg, ct), \pi)$ as its solution in the knowledge-soundness game and wins. Otherwise, $\mathcal{B}$ outputs random data. Hence, the probability that $\mathcal{B}$ wins the knowledge-soundness game is at least the probability of the failure event in G_1.

Since G_0 and G_1 behave identically unless the failure event occurs and NIZK is knowledge-sound, we have

$$\begin{aligned}\Pr[G_0(\mathcal{A}, 1^\lambda) \to 1] &\leqslant \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] + \Pr[\mathsf{fail}_{01} \textit{ in } G_1(\mathcal{A}, 1^\lambda)] \\ &\leqslant \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{Adv}^{\mathrm{knSound}}_{\mathsf{NIZK}}(\mathcal{B}, \lambda) \leqslant \Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{negl}^{\mathrm{knSound}}_{\mathsf{NIZK}}(\lambda). \quad (16)\end{aligned}$$

G_2: In this game, when running the Trace algorithm, the challenger checks each signature $\sigma_i = (ct, \pi)$ as follows:

- If the corresponding message msg_i is in D, the challenger checks whether an SPCE decryption of ct is equal to a previously extracted kAGS signature.
- If $msg_i \notin D$, the challenger checks whether an SPCE decryption of ct is equal to $\perp$ or to a previously extracted kAGS signature.

If this check fails, the challenger aborts (outputting 0). We denote that failure event by fail_{12}.

Note that when the Trace algorithm is called, it has already been checked that the number of signed messages in D exceeds $k \cdot |\mathcal{L}_{\mathsf{id}}|$. Thus, if G_2 does not abort on the checks introduced above, then during Trace, after decrypting ciphertexts contained in $\{\sigma_1, \ldots, \sigma_s\}$, the challenger obtains more than $k \cdot |\mathcal{L}_{\mathsf{id}}|$ correct kAGS signatures.

Due to the changes introduced in G_1, every ciphertext ct in each signature σ_i is a correctly computed SPCE encryption of an extracted kAGS signature with respect to msg. Hence, the failure event fail_{12} corresponds exactly to a violation of SPCE correctness.

Since G_1 and G_2 behave identically unless the failure event occurs, we have

$$\begin{aligned}\Pr[G_1(\mathcal{A}, 1^\lambda) \to 1] &\leqslant \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] + \Pr[\mathsf{fail}_{12} \text{ in } G_2(\mathcal{A}, 1^\lambda)] \\ &\leqslant \Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] + \mathsf{negl}^{\mathrm{corr}}_{\mathsf{SPCE}}(\lambda). \quad (17)\end{aligned}$$

To bound $\Pr[G_2(\mathcal{A}, 1^\lambda) \to 1]$, we construct an adversary $\mathcal{B}$ against the traceability of kAGS. $\mathcal{B}$ simulates G_2 for $\mathcal{A}$; to compute a $kASPCGS$ master public key mpk, it uses the kAGS public key received from its challenger, while computing everything else itself. $\mathcal{B}$ forwards $\mathcal{A}$'s $\mathcal{O}^{\mathsf{KeyGen}}$ and $\mathcal{O}^{\mathsf{HonestKG}}$ queries to its own oracle without modification, since the key-generation protocol of Π is exactly that of kAGS. Thus, the number of users $\mathcal{B}$ registers with its challenger is exactly the number $\mathcal{A}$ registers. To answer a query to $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}$ forwards it to its own oracle, and then SPCE encrypts the received signature and computes a corresponding proof.

When $\mathcal{A}$ outputs the challenge signatures, $\mathcal{B}$ runs the checks from G_1 (aborting if any check fails), then compiles a list of kAGS signatures by SPCE-decrypting the ciphertexts contained in $\mathcal{A}$'s signatures. It also runs the additional decryption checks from G_2. $\mathcal{B}$ outputs the resulting list of kAGS signatures to its challenger. As argued, all these signatures are valid, and there are at least $k \cdot |\mathcal{L}_{\mathsf{id}}|$ of them.

$\mathcal{A}$ wins G_2 if and only if $\mathsf{kAGS.Trace}$ does not output identity of any registered user when run on the decrypted signatures. Likewise, $\mathcal{B}$ wins its own game if and only if $\mathsf{kAGS.Trace}$ does not output any registered user when run on its output. Since $\mathcal{B}$'s output is exactly the decrypted signatures, $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins. Therefore, by the traceability of kAGS, we have

$$\Pr[G_2(\mathcal{A}, 1^\lambda) \to 1] = \mathbf{Exp}^{\mathrm{Trace}}_{\mathsf{kAGS}}(1^\lambda, \mathcal{B}) \leqslant \mathsf{negl}^{\mathsf{Trace}}_{\mathsf{kAGS}}(\lambda). \tag{18}$$

Finally, by combining inequalities (16)–(18), we conclude the proof. □