Mihir Bellare, Wei Dai, and Phillip Rogaway

Reimagining Secret Sharing: Creating a Safer and More Versatile Primitive by Adding Authenticity, Correcting Errors, and Reducing Randomness Requirements

Abstract:
Aiming to strengthen classical secret-sharing to make it a more directly useful primitive for human end-users, we develop definitions, theorems, and efficient constructions for what we call *adept* secret-sharing. Our primary concerns are the properties we call *privacy*, *authenticity*, and *error correction*. Privacy strengthens the classical requirement by ensuring maximal confidentiality even if the dealer does not employ fresh, uniformly random coins with each sharing. That might happen either intentionally—to enable *reproducible* secret-sharing—or unintentionally, when an entropy source fails. Authenticity is a shareholder's guarantee that a secret recovered using his or her share will coincide with the value the dealer committed to at the time the secret was shared. Error correction is the guarantee that recovery of a secret will succeed, also identifying the valid shares, exactly when there is a unique explanation as to which shares implicate what secret. These concerns arise organically from a desire to create general-purpose libraries and apps for secret sharing that can withstand both strong adversaries and routine operational errors.

Keywords: Adept secret-sharing, computational secret sharing, cryptographic definitions, secret sharing

DOI 10.2478/popets-2020-0082
Received 2020-02-29; revised 2020-06-15; accepted 2020-06-16.

1 Introduction

Overview. This paper strengthens classical secret-sharing [17, 40] to obtain a primitive we call *adept* secret-sharing (ADSS). Our initial reason for developing ADSS was to address use cases involving journalists and whistleblowers. We were motivated by a conversation with journalist Laurent Richard [22, 36], by the Snowden revelations [24], and by the development of Sunder [39]. We recognized that unadorned Shamir secret-sharing [40] wouldn't do; for example, garbage would be recovered if a share got accidentally corrupted, and a strong adversary could force recovery of whatever secret it wanted by adjusting a single share. We set out to develop a primitive that would guarantee more. It would need to be versatile, easy to understand, and support efficient and provably secure realizations.

Our approach is definitionally focused. Modern cryptography has taught that stronger definitions lead to conforming schemes that are easier to correctly use, so less prone to misuse. Our definitions are motivated by use cases, although no one use case fully motivates all of our demands. This is customary. By way of analogy, no application we know requires the full strength of IND-CCA public-key encryption [34], yet this has become the accepted definitional target because it implies other properties, such as nonmalleability [19], that are useful in numerous settings. We strive to create definitions that can play the same role for secret sharing that IND-CCA plays for public-key encryption.

Sample use case. To start to appreciate why new definitions are needed, let us consider a realistic but fictitious use case. German journalist D is visiting New York when a source hands him a thumb drive of shocking, classified files. D transfers the archive to his laptop, encrypts it with a strong passphrase, and destroys the thumb drive. D now wants to return to Berlin with these materials, but fears he will be detained, or worse, before he can publish. D mustn't have the sensitive plaintext on him at border crossings, where phone and laptop contents may be copied by authorities.

To ensure that the material gets out no matter what, D decides to give the encrypted archive and a share of its decryption key to colleagues A, B, and C. He intends that any two parties can reconstruct the archive. D decides it would be safest to meet A at the Newark airport, B at the Icelandair lounge where D will transit, and to send C her materials over Signal.

To begin, D needs to generate a share c of his passphrase for C. But the way secret-sharing schemes generate shares is probabilistic: fresh coins are chosen

Mihir Bellare: University of California, San Diego, USA. E-mail: mihir@eng.ucsd.edu
Wei Dai: University of California, San Diego, USA. E-mail: weidai@eng.ucsd.edu
Phillip Rogaway: University of California, Davis, USA. E-mail: rogaway@cs.ucdavis.edu

BY-NC-ND

Auth	**A share held by a user can recover, if anything, only the one secret committed to at the time of the sharing, regardless of what other shareholders contribute.**
Errx	**Recovery will reconstruct the secret and identify the valid shares if and only if there's a unique plausible explanation for what shares implicate what secret.**
Priv	**Unauthorized sets of shares reveal the least possible amount of information given the combined entropy of the secret and the provided coins.**

Fig. 1. Properties of ADSS. When uniform coins are used for sharing, the Priv notion captures the complexity-theoretic formalization of the classical secret-sharing goal; otherwise, it asks for more. Authenticity and error correction concern attacks on the reconstruction of secrets—attacks that get participants to reconstruct the wrong secret, or no secret at all.

with each sharing. So it would seem that D will need to retain A's share a until he meets A in Newark, and must retain B's share b all the way to Iceland. But this is no good, for keeping a and b on the laptop along with the encrypted achieve is equivalent to keeping the archive as plaintext. A better choice might be to retain the *coins* that generated the shares, using them, and the passphrase, to regenerate a or b only when they are needed. But it is unclear what security properties secret sharing will have if an attacker learns retained coins. With Shamir secret-sharing, acquiring them (e.g., by confiscating the device) along with any *one* share (say c) enables reconstruction of the secret. In any case, D needs to use off-the-shelf tools, which, quite correctly, do not support the retention of coins used for share generation.

The scenario motivates *reproducible* secret-sharing: the ability to recompute a share, or a vector of shares, as long as you still have the secret.

Continuing our example, we must report that, soon after his arrival, D mysteriously vanished in Berlin. Meanwhile, A fell ill with COVID-19. Parties B and C nervously converge in Iceland. Unfortunately, C's smartphone had already been hacked by a state intelligence agency, her share c quietly replaced by $\tilde{c}$. When B and C reconstruct the passphrase and use it to decrypt, the plaintext *looks* fine—parties B and C don't know that anything is wrong—but the archive is less important than they anticipated. It is not the original one. This is possible, at least in principle, because, with classical secret-sharing, if someone can control a single share, they may be able to control the secret that is recovered, even without knowing other shares. Nothing in the classical secret-sharing definition excludes this. This possibility motivates another non-standard aim: *authenticity.* It guarantees that recovery using a share either *fails* or recovers the secret originally associated to it. Schemes like Shamir's achieve nothing like this.

Finally, as an alternative continuation of our story, party A, now recovered, meets up with B and C in Iceland. Party C's share is still wrong. When A, B, and C contribute their shares $a, b, \tilde{c}$ for recovery, a classical secret-sharing scheme (like Shamir's) will recover *something*—but something wrong. This time, the recovered archive looks like random bits. The shareholders know that something went wrong, but they don't know what. If they had the insight to try recovery again *without* using C's share $\tilde{c}$ they would recover the correct secret. But they don't know to do this. How much nicer it would be if the recovery algorithm *itself* would have said: "look, share $\tilde{c}$ was bad, but shares a and b were fine, and implicate the following passphrase." A scheme like that enjoys *error correction.* Our formalization strengthens *robustness* [15, 31], which would actually be sufficient for this example (but not, say, for 2-of-4 secret sharing).

We use the labels Auth (authenticity), Errx (error correction), and Priv (privacy) for our main goals (the last of these encompassing reproducibility). Fig. 1 provides a single-sentence description of each. Fig. 2 summarizes definitional choices and their rationale more broadly. As that figure makes clear, we have taken clues from multiple directions—not just use cases—as to what characteristics an ADSS scheme should enjoy.

Enhanced syntax. ADSS begins with an enriched syntax, over which the security notions above can be defined. Let us start by taking a look at the new syntax.

Unlike a classical secret-sharing scheme, the sharing algorithm of an ADSS scheme is *deterministic*, surfacing an input R that captures the provided *coins.* This enables *reproducibility* (described above) and *hedging* (described later). The sharing algorithm also takes in a description of an *access structure*—the specification of which sets of shareholders are *authorized*—rather than being specific to one. This enables runtime selection of the access structure and for the access structure itself to be authenticated being crucial for security. Finally, the sharing algorithm now takes in a string of *associated data* (AD), analogous to that seen in schemes for authenticated encryption. Moving on, the recovery algorithm of an ADSS scheme no longer operates on *vectors* of shares, but on *sets* of shares [2]. This better models the coming together of human participants who have only their shares. And the algorithm not only returns the original secret, but also identifies which shares were deemed to be valid. This allows a shareholder to reject

Characteristic	*Reasons*
The sharing algorithm is deterministic but surfaces an input R via which the caller can provide "coins." (In contrast, classical secret-sharing is probabilistic.)	**Some settings require *reproducibility*: the ability of a dealer to recompute a share. ▶ Eg, a dealer may distribute shares of a passphrase to different shareholders at multiple points in time. ▶ Or she may need to *replace* the share of a shareholder who has lost access to it. ▶ The analogous move from internal coins to coins provided across the interface was pivotal for authenticated encryption (AE). ▶ Without surfacing R one can't investigate the impact of different choices for it.**
The coins R provided to the sharing algorithm might not be uniform. They might be fixed. They might depend on persistent *state.*	**▶ Failures in random-number generators are common. They arise from implementation errors or the inaccessibility of good randomness. ▶ Install-time randomness or maintaining state may be more feasible than per-sharing randomness. ▶ Hedged and deterministic encryption have proven to be useful. ▶ Deterministic signature schemes avoid security vulnerabilities that probabilistic signatures schemes are susceptible to.**
A string of *associated data* (AD) can be bound to a sharing.	**▶ The AD might encode information like: time of deal or conditions under which recovery may take place. ▶ The inclusion of AD in AE has been very useful for applications.**
An encoding of the intended access structure is provided as an input to the sharing algorithm. It is authenticated.	**▶ General-purpose libraries and user-facing tools need to support a variety of access structures. A caller might not know which it needs until runtime. ▶ Without authenticating the access structure itself simple attacks are possible.**
Recovery operates on set-valued inputs, not the vector-valued inputs of classical secret-sharing.	**▶ When a group of human shareholder get together for a reconstruction ceremony there may be no side-information to order them. ▶ Without side information there is no way to know even the number of shareholders needed to reconstruct.**
Definitions envision shares being arbitrarily changed or created.	**▶ Real-world adversaries aren't restricted to crossing-out shares from known shareholders, but can modify shares and create shares for new, alleged shareholders.**
An incorrect secret should never be returned: either one should get back the original message or an indication of failure.	**▶ Honest parties deserve to know if recovery was impossible. ▶ Parties may be unable at reconstruction time to assess if a recovered message "makes sense". And "making sense" is not evidence of authenticity, anyway (a common misunderstanding in encryption). ▶ Authenticity implies nonmalleability. Malleable schemes would allow an adversary to adjust an unknown secret to one that better suits it.**
Shares can have a designated non-secret ("public") portion.	**▶ Secrets can be extremely long, which implies that some shares will be. Having to store less privately can reduce the burden of custody. ▶ It is desirable to be able to store shares (at least the private part) in an HSM (hardware security module).**
If shares get corrupted then the recovery process must fix the problem if there's an unambiguous explanation as to what went wrong.	**▶ Shares can get corrupted for inconsequential reasons, like the accidental mixup of shares from different sharings. ▶ *Robust* secret-sharing is already recognized as useful, but was formalized in a way that neither demands recovery from recoverable errors nor forbids the recovery of junk when there is no authorized set of shareholders.**
On recovery, particular shares can be marked as trusted, or a known access structure can be provided.	**▶ If a shareholder reconstructs, she likely trust her *own* share. ▶ If some shares were *signed* by a trusted dealer, we can insist on using them. ▶ The recovering party might have side information on what access structure was used. ▶ An adversary can try to thwart recovery by adding a single share asserting a 1-of-1 access structure.**

Fig. 2. Some ADSS definitional choices and their rationale. Considerations shaping our definitions include use cases, philosophical arguments, and reasoning by analogy. Simplicity and strength were key desiderata.

a recovered secret if she has confidence in her own share but it was not deemed valid. As for the shares themselves, the access structure and AD must be encoded in each, to ensure that no side information needs be known by the recovering party. And shares can have separate secret and non-secret parts, so that shareholders need only keep the first in private storage. This enables the sharing of arbitrarily long secrets even when shareholders are only able to privately store a limited amount. The syntactic changes just sketched may seem low-level but are fundamental in enabling the capabilities we seek.

Enhanced security. A journalist could certainly share her documents using Shamir secret-sharing [40]. This provides privacy. But as our extended example illustrates, the adversary may have other goals in mind, like disrupting recovery, either by making it fail or by making shareholders recover something other than that which the dealer shared. The adversary can attempt to achieve this goal by infiltrating a shareholder's system (something nation-state adversaries are good at) and changing her share. It can create entirely new shareholders that show-up for reconstruction. We want to defend against such attacks to the maximal extent possible. To do so, we develop Auth and Errx.

Authenticity (Auth), which we alternatively term *binding*, ensures that when a user is given a share S, there is at most one secret M for which it might be a share. The share is effectively a *commitment* to that secret. A shareholder can thus regard her share as a locked box containing some well-defined secret that she does not yet know. (We do allow that if the dealer was dishonest then *nothing* might be in that box.) In short, authenticity concretizes the basic intuition that a share is associated to some one particular *thing*.

Error correction (Errx) ensures that if some shares from a sharing of a secret get corrupted, or new shares are added, but there remains a single nontrivial explanation as to what the secret must have been before the shares got messed up, then the recovery process must identify that one correct secret. It must also indicate which shares implicate it. Recovery must fail if there is no authorized subset of shares, or if there are two or more explanations for what got corrupted. In short, an Errx-secure scheme must fix what is fixable, and must indicate if shares irreparably messed up.

Auth and Errx are different from *verifiable secret-sharing* (VSS) [18]. VSS requires a reliable broadcast channel, which may not be available; Auth and Errx do not. Errx is related to but different from *robustness* [15, 31], which aims to guarantee message recovery despite the presence of some bad shares shuffled in among an authorized collection of good ones. Errx formalizes different intuition: that recovery does the best job possible with whatever is presented. For this reason, Errx is achievable for any access structure, while robustness is achievable for threshold schemes with honest majority, but little more. Errx demands that *nothing* be recovered when there is *no* authorized subset of shares, will robustness requires nothing. All of that said, Errx security *does* imply robustness whenever the latter can be achieved. Auth and Errx have little in common with *repairable threshold schemes* [25, 32], which allow a party to reconstruct a missing share by interacting with fellow shareholders. In scenarios we care about, shareholders may not have the ability to interact with one another prior to recovery. See Section A for fuller comparisons with VSS, robustness, and repairability.

Enhanced usability. Our ADSS schemes employ *hedging* [5], using the anticipated unpredictability of the secret itself, together with the entropy in the provided coins, to provide as much privacy as possible. This means that privacy is maintained, to the extent possible, in settings where high-quality randomness is unavailable or was inadequately harvested. At the same time, the approach provides for *reproducibility*, enabling the dealer, if it so arranges, to re-share a secret M and get the same shares as before. Other ADSS elements that enhance usability include: the ability to handle AD, having the recover algorithm operate on sets instead of vectors of shares; and the capacity to deal with enormous secrets, as we now discuss.

Protocols and proofs. We construct simple, efficient, and provably secure ADSS schemes for arbitrary access structures. Our schemes can be used for splitting anything from a PIN code to a huge archive of files (the Panama papers were 2.6 terabytes [26]). In sharing out a large archive one needn't encrypt it and then share out the key; the user can regard the archive itself as the secret, which is conceptually and operationally simpler.

Our constructions begin with a scheme $\mathbb{S}1$ that satisfies our ADSS syntax but only achieves Priv-security with uniformly random coins—what we call *classical* privacy, or Priv\$. The scheme only works for threshold access structures. It is basically just Shamir's scheme [40], but adapted to our syntax. Scheme $\mathbb{S}2$ still achieves only Priv\$ privacy, but can handle any access structure, now presented as a circuit of threshold gates. Despite the classical aim, we could find no full exposition or proof on how to carry out secret sharing for arbitrary access structures; our paper fills this gap.

Our main construction is the transformation AX that converts a secret-sharing scheme $\mathbb{S}$ that achieves only classical (Priv\$) privacy to a scheme $\mathbb{SS} = \mathrm{AX}[\mathbb{S}]$ that achieves Priv and Auth security. This can then be composed with a further transformation, EX, to achieve Errx security. Rather roughly, AX starts by symmetrically encrypting the secret M to the ciphertext C, which is put in the public portion of each share. The key K for this encryption is determined by applying a hash function to all inputs the sharing algorithm gets. The hash function is here used as a randomness extractor [33]. The lower-level secret-sharing scheme $\mathbb{S}$ shares out K using randomness that is again extracted from the inputs to $\mathbb{SS}$. When speculative values are recovered, a correctness check is done to see if re-sharing M with the recovered randomness R gives rise to a superset of the shares received. Schemes $\mathbb{SS}1$ and $\mathbb{SS}2$ are the result of applying AX and then EX to $\mathbb{S}1$ and $\mathbb{S}2$. They are the concrete ADSS schemes that we propose.

Discussion. Kacsmar, Komlo, Kerschbaum, and Goldberg [29] also address gaps between the formulation and use of secret sharing. Their motivations are similar to ours: closing the theory/practice gulf in this domain. They employ the idea of *ceremonies* [20] and design a proactive VSS scheme [27] to achieve goals that they call *integrity* and *availability*. Our work is more formal, and, in carrying it out, we have insisted on retaining the fundamental elements of the classical model: ADSS abides no broadcast channels, no interaction among shareholders, no preprocessing, no PKI (public-key infrastructure), and no algorithms but Share and Recover.

Secret sharing can be viewed as a flavor of encryption [15]: sharing corresponds to encryption; recovery corresponds to decryption. From this vantage, the move from classical to adept secret-sharing mirrors the move from semantically secure to authenticated encryption [11, 13, 30], as well as the move from probabilistic to deterministic [4, 8] and hedged [5] public-key encryption. We shift the focus from eavesdropping to interference, and from perfect to possibly absent or deficient randomness. We trade internal randomness for an externally supplied input [38]. We add in AD [37]. Of course there are spots where the analogy breaks down: secret sharing involves no keys, while access structures were outside the ambit of encryption prior to ABE [23]. Still, the analogy explains many aspects of our work.

Reproducibility comes at the price of diminished privacy for low-entropy secrets. But we never mandate reproducibility; we merely enable it. If the dealer uses random coins shes gets classical privacy; if she wires in a constant, she gets best possible privacy for a deterministic scheme. Similar tradeoffs are present for deterministic and searchable public-key encryption [4] and for format-preserving encryption [12]. In addition, the strengthening of secret sharing that begins by surfacing the coins includes other benefits, like hedging.

The value of new definitions is always somewhat speculative. Our definitional choices have been guided by uses cases, by analogies, and by conversations with developers, journalists, whistleblowers, and cryptographers. But only time will tell if we have identified the secret-sharing aims that can precipitate a flourishing.

2 Preliminaries

Notation. Fig. 3 summarizes the most frequently used notation in this paper. The table may serve as a reference or overview of things to come.

$\{0,1\}^*$	Set of all strings over $\{0,1\}$
$\{0,1\}^{**}$	Set of all vectors (= lists) of strings
$\langle \cdots \rangle$	A string encoding of what's in the brackets
$\bot$	Indicates invalidity: no secret can be recovered
$[1..n]$	Integers between 1 and n inclusive
$a \twoheadleftarrow X$	Sample, then assign. Uniform dist if X a set
A	A string that names an access structure
$\mathcal{A}$	An access structure: a set of subsets of $[1..\mathrm{n}(\mathcal{A})]$
$\mathbb{A}$	An adversary
Acc	Access-structure naming function
Access	Set of strings that name access structures
$\mathbf{Adv}_{\mathbb{S}}^{\mathrm{auth}}(\mathbb{A})$	$\mathbb{A}$'s advantage breaking Auth-security of $\mathbb{S}$
$\mathbf{Adv}_{\mathbb{S}}^{\mathrm{errx}}(\mathbb{A})$	$\mathbb{A}$'s advantage breaking Errx-security of $\mathbb{S}$
$\mathbf{Adv}_{\mathbb{S}}^{\mathrm{priv}}(\mathbb{A})$	$\mathbb{A}$'s advantage breaking Priv-security of $\mathbb{S}$
$\boldsymbol{\mathcal{AS}}$	The set of all possible access structures
Auth	Main authenticity property we formalize
AX	Transform to get Auth + Priv from Priv$
ε	The empty string
Errx	The error-correction property we formalize
EX	Transform to get Errx security
Known	Set of things the recovering party might be sure of
H	Hash function, modeled as a random oracle
M	A secret shared out in a secret-sharing scheme
$\lvert M \rvert$	Length of the string M (in bits)
Msg	The set of possible secrets (messages)
$\mathrm{n}(\mathcal{A})$	Number of parties in the access structure $\mathcal{A}$
$\mathcal{P}(X)$	Set of all finite subsets of the set X
Priv	The new privacy property we formalize
Priv$	Classical privacy property. Weaker than Priv
R	Randomness / coins given to Share
Rand	All possible coins (randomness)
S	A share. String-valued and have several parts
$\boldsymbol{S}$	A vector of shares, $\boldsymbol{S} = (S_1, \ldots, S_n)$
$\boldsymbol{S}[i]$	The i-th entry of vector $\boldsymbol{S} = (\boldsymbol{S}[1], \ldots \boldsymbol{S}[\lvert \boldsymbol{S} \rvert])$
$\boldsymbol{\mathcal{S}}$	A set of shares, $\mathcal{S} = \{S_1, \ldots, S_t\}$
$\mathbb{S}$	Generic ADSS scheme $\mathbb{S} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$
$S.\mathrm{as}$	Access structure associated to share S
$S.\mathrm{id}$	Identity of party associated to share S
$S.\mathrm{pub}$	Non-secret part of share S
$S.\mathrm{sec}$	Secret part of share S
$S.\mathrm{tag}$	Tag (AD) part of share S
$\mathbb{S}1$	Shamir-like SS scheme. Achieves Priv$
$\mathbb{S}2$	Yao-like SS scheme. Achieves Priv$
Recover	Algorithm that recovers a secret
Share	Algorithm that shares a secret
Share	All possible shares (which are strings)
Share*	All possible *vectors* of shares
Shares	All possible *sets* of shares
T	Tag (associated data) (a string)
Tag	The set of all possible tags (AD values)
$\mathcal{V}$	Maximal set of valid shares, $\mathcal{V} \subseteq \mathcal{S}$

Fig. 3. Frequently used notation. Note that font styles for a given letter are routinely differentiated.

Access structures. We need a way to specify which parties are authorized to reconstruct the secret. Number parties $1, 2, \ldots, n$. We then define an *access structure* $\mathcal{A}$ as a set of sets of positive numbers. It must be finite, nonempty, and exclude the empty set. Define $\mathrm{n}(\mathcal{A})$, the number of parties in $\mathcal{A}$, as the least n such that $U \subseteq [1..n]$ for all $U \in \mathcal{A}$. We say that $U \subseteq [1..\mathrm{n}(\mathcal{A})]$ is *authorized* if $U \in \mathcal{A}$ and *unauthorized* if $U \notin \mathcal{A}$. We require access structures be *monotone*, which means that an authorized set stays authorized if you add in parties: if $U \in \mathcal{A}$ and $U \subseteq V \subseteq [1..\mathrm{n}(\mathcal{A})]$ then $V \in \mathcal{A}$.

An example access structures is the 2-out-of-3 one $\mathcal{A}_{2,3} = \{\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$. More generally, for $1 \leq t \leq n$ the *threshold* access-structure $\mathcal{A}_{n,t}$ is $\{U \subseteq [1..n] : |U| \geq t\}$. A secret-sharing scheme for such access structures is called a *threshold scheme*. A simple *non*-threshold access-structure is $\mathcal{A}_{12\vee 13} = \{\{1,2\},\{1,3\},\{1,2,3\}\}$: party 1 and either party 2 or 3.

Classical secret-sharing. Let us briefly review the *classical* notion of a secret-sharing scheme—what a scheme like Shamir's targets [17, 40]. One can formalize a classical secret-sharing scheme as a pair of algorithms $\mathbb{S} = (\mathit{Share}, \mathit{Recover})$ along with an associated access structure $\mathcal{A}$, as follows:

Share The sharing algorithm *Share* probabilistically maps a message $M \in \mathsf{Msg}$ to a vector (or *list*) of $n = \mathrm{n}(\mathcal{A})$ *shares*, each of them a string: $\boldsymbol{S} \twoheadleftarrow \mathit{Share}(M)$.

Recover The recovery algorithm *Recover* is a deterministic algorithm that takes in an n-vector of values, $n = \mathrm{n}(\mathcal{A})$, each being either a string or the special symbol $\Diamond$, which is used to indicate that the share is *missing*. It returns a string $M \leftarrow \mathit{Recover}(\boldsymbol{S})$.

If $\boldsymbol{S} = (S_1, \ldots, S_n)$ is an n-vector of strings and $U \subseteq [1..n]$, let $\boldsymbol{S}_U$ be the n-vector with ith component S_i if $i \in U$, and $\Diamond$ otherwise. So $\boldsymbol{S}_U$ is $\boldsymbol{S}$ with $\Diamond$-symbols shuffled in at positions outside of U. Then we require the following: if $\boldsymbol{S} \twoheadleftarrow \mathit{Share}(M)$ and $U \in \mathcal{A}$ then $\mathit{Recover}(\boldsymbol{S}_U) = M$. In words: you can recover the secret from an authorized subvector of shares.

For any $M, M' \in \mathsf{Msg}$ and any $U \notin \mathcal{A}$, we can regard $(\mathit{Share}(M))_U$ as a distribution on vectors of shares, the underlying randomness that of *Share*. The security notion for a classical secret-sharing scheme can then be formalized by asking that distributions $(\mathit{Share}(M))_U$ and $(\mathit{Share}(M'))_U$ be identical. In words: unauthorized subvectors of shares reveal nothing about the secret. If desired, this condition can be relaxed to computational indistinguishability or formalized in other ways [15, 31].

3 Syntax

Changes. We enrich the syntax of a classical secret-sharing scheme in multiple directions. First, the access structure $\mathcal{A}$ won't be fixed, but, instead, the party who shares a secret, the *dealer*, will be able to specify the access structure it wants. A string A will denote the desired access structure, a function Acc specifying its interpretation. For example, the string "`2,3`" might denote the threshold access-structure $\mathcal{A}_{2,3}$. Second, our sharing algorithm Share will have still more inputs. Beyond the access structure and the secret, the dealer will provide *coins* and *associated data*. With coins now an explicit input, the algorithm will be deterministic. Finally, the recovery algorithm Recover will output more: not only will it return the recovered secret, but also the shares that were used. Alternatively, it may recover nothing, outputting a special I-can't-recover-anything symbol. We now make all of this precise.

Formal definition. We define a scheme for *adept secret-sharing* (ADSS) as a triple of deterministic functions $\mathbb{S} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$, as follows.

Acc: The access-structure naming function

$$\mathrm{Acc}\colon \mathsf{Access} \to \mathcal{AS}$$

associates an access structure $\mathcal{A} = \mathrm{Acc}(A)$ with each string $A \in \mathsf{Access}$. Here $\mathsf{Access} = \{0,1\}^*$ (Kleene star) is the set of all binary strings, while $\mathcal{AS}$ is the set of all possible access structures. Note that there may be multiple ways to name an access structure under Acc: distinct strings A and A' such that $\mathrm{Acc}(A) = \mathrm{Acc}(A')$. Also, some access structures might be impossible to name using Acc: for some $\mathcal{A} \in \mathcal{AS}$ there might be no $A \in \mathsf{Access}$ with $\mathrm{Acc}(A) = \mathcal{A}$. For example, a secret-sharing scheme designed for threshold access structures won't be able to request $\{\{1,2\},\{1,3\},\{1,2,3\}\}$ (i.e., "1 and (2 or 3)").

Share: The sharing algorithm

$$\mathrm{Share}\colon \mathsf{Access} \times \mathsf{Msg} \times \mathsf{Rand} \times \mathsf{Tag} \to \mathsf{Share}^*$$

takes in a description $A \in \mathsf{Access}$ of an access structure, a *message* (or *secret*) $M \in \mathsf{Msg}$, some *coins* $R \in \mathsf{Rand}$, and a *tag* (or *associated data*) $T \in \mathsf{Tag}$. It outputs a vector of *shares*. Here Msg, Access, Rand, Tag, $\mathsf{Share} \subseteq \{0,1\}^*$ are binary strings. We require that $M \in \mathsf{Msg}$ implies $\{0,1\}^{|M|} \subseteq \mathsf{Msg}$ and $R \in \mathsf{Rand}$ implies $\{0,1\}^{|R|} \subseteq \mathsf{Rand}$. The sharing algorithm must generate the appropriate number of shares for the specified access structure: $|\mathrm{Share}(A,M,R,T)| = \mathrm{n}(\mathrm{Acc}(A))$ for all $A \in \mathsf{Access}$, $M \in \mathsf{Msg}$, $R \in \mathsf{Rand}$, and $T \in \mathsf{Tag}$: By $|\cdot|$ we denote the length, cardinality, or number of components for string, set, or vector.

Recover: The message-recovery algorithm

$$\mathrm{Recover}\colon \mathsf{Shares} \to \mathsf{Msg} \times \mathsf{Shares} \cup \{\bot\}$$

maps a set of shares $\mathcal{S} \in \mathsf{Shares} = \mathcal{P}(\mathsf{Share})$ to a message $M \in \mathsf{Msg}$ and a set of *valid shares* $\mathcal{V} \subseteq \mathcal{S}$. Alternatively, the algorithm can decline to produce such a pair and return $\perp$ instead. By $\mathcal{P}(X)$ we mean the finite power set of X, the set of all finite subsets of X. (The traditional power set of an infinite set such as $\mathsf{Share} = \{0,1\}^*$ includes infinite subsets. We don't want that, as one provide Recover, like any algorithm, a *finite* set of strings.) Note that Share returns a *list* of shares while Recover takes in a *set* of shares.

Parts of a share. We establish the convention that each share $S \in \mathsf{Share}$ is actually the encoding of five strings, $S = \langle S.\mathrm{id}, S.\mathrm{as}, S.\mathrm{sec}, S.\mathrm{pub}, S.\mathrm{tag}\rangle$. We call the parts of a share its *identity*, *access structure*, *secret portion*, *public portion*, and *AD*. We extend the .sec and .pub operators to vectors, defining $\boldsymbol{S}.\mathrm{sec} = (S_1.\mathrm{sec}, \ldots, S_n.\mathrm{sec})$ and $\boldsymbol{S}.\mathrm{pub} = (S_1.\mathrm{pub}, \ldots, S_n.\mathrm{pub})$ when $\boldsymbol{S} = (S_1, \ldots, S_n)$. We extend the .id operator to sets, defining $\mathcal{S}.\mathrm{id} = \{S.\mathrm{id} : S \in \mathcal{S}\}$. We extend the .as operator to sets, so that $\mathcal{S}.\mathrm{as} = A$ if $S.\mathrm{as} = A$ for all $S \in \mathcal{S}$, and $\mathcal{S}.\mathrm{as} = \perp$ otherwise. We insist that $\boldsymbol{S} = \mathrm{Share}(A, M, R, T) = (S_1, \ldots, S_n)$ implies that $S_i.\mathrm{tag} = T$ and $S_i.\mathrm{as} = A$ and $S_i.\mathrm{id} = i$ for all i. Whenever Recover($\mathcal{S}$) returns a pair $(M, \mathcal{V})$ we demand that all shares in $S \in \mathcal{V}$ share the same .as component A, the same .tag component T, and that $\mathcal{V}.\mathrm{id} \in \mathrm{Acc}(A)$, meaning the set of shareholders underlying $\mathcal{V}$ is authorized.

Random oracles. We allow the Share and Recover algorithms of an ADSS scheme may call an oracle Hash that realizes a function $H \in \Omega$, with Ω, the set of all functions $H\colon \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$ such that $|H(\ell, \boldsymbol{X})| = \ell$. By $\{0,1\}^{**} = (\{0,1\}^*)^*$ we denotes the space of vectors of strings. We can explicitly indicate the presence of the oracle or hash function that Share and Recover may access by writing it as a superscript.

Basic correctness. An ADSS scheme $\mathbb{S} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ enjoys *basic correctness* if for all $A \in \mathsf{Access}$, $M \in \mathsf{Msg}$, $R \in \mathsf{Rand}$, $T \in \mathsf{Tag}$, $H \in \Omega$, $\boldsymbol{S} \leftarrow \mathrm{Share}^H(A, M, R, T)$, and $U \subseteq [1..\mathrm{n}(\mathrm{Acc}(A))]$,

- if $U \in \mathrm{Acc}(A)$ then $\mathrm{Recover}^H(\boldsymbol{S}[U]) = (M, \boldsymbol{S}[U])$,
- while if $U \notin \mathrm{Acc}(A)$ then $\mathrm{Recover}^H(\boldsymbol{S}[U]) = \perp$.

Here $\boldsymbol{S}[U] = \{\boldsymbol{S}[i]\colon i \in U\}$ is the set of shares from parties U. In words: applying Recover to a subset of shares obtained by sharing out M gives M if the subset is authorized and $\perp$ if it is not. We henceforth require that all ADSS schemes satisfy basic correctness.

Notation. We write $\mathbb{S}$.Acc, $\mathbb{S}$.Share, and $\mathbb{S}$.Recover for the components of $\mathbb{S}$. In the same way, we write $\mathbb{S}.\mathsf{Access}$, $\mathbb{S}.\mathsf{Msg}$, $\mathbb{S}.\mathsf{Rand}$, and $\mathbb{S}.\mathsf{Tag}$.

Discussion. Once the decision has been reached to provide the access structure to Share it is tempting to just say that it's encoded as a string $\langle \mathcal{A}\rangle$ and leave it at that. But more care needs to be taken because *what* access structures can be named, and *how compactly*, are central concerns of secret sharing. This is what motivates making Acc a first-class component of an ADSS scheme.

Let us give some examples of access-structure naming functions Acc. For threshold schemes, a string $\langle n, t\rangle$ encoding numbers n and t could name $\mathcal{A}_{n,t}$. It would be equally permissible, but less compact and convenient, to have Acc expect a string listing authorized sets, like "`{{1,2},{1,3},{2,3},{1,2,3}}`". For a representation that is compact *and* expressive, the string A could encode a Boolean circuit of threshold gates having a single output wire and input wires $1, \ldots, n$. We'd say that $U \in \mathrm{Acc}(A)$ if the circuit named by A evaluates to true when its n inputs indicate if a party is present (that bit is 1) or absent (it is 0).

Having Recover take in a set instead of a vector relieves shareholders of having to know their "position" in line. It also opens the door to authenticity notions where multiple parties can impersonate some shareholder.

While the AD of an ADSS scheme is analogous to that of an authenticated-encryption (AE) scheme, there are important differences. Our AD values are not assumed to be known by the party recovering a secret; an AE scheme's AD value is. This follows the philosophical view that for secret sharing one should not require the receiver to know anything beyond what is in the shares.

The "public" portion of a share need not be public; we only mean that it is not a secret. The secret/public distinction matters most when the message being shared is long. We anticipate that most or all of the public portion of shares would be the same across all shares. When this is true, the public potion of shares might be kept in some highly available repository, rather than stored with each share.

Extensions to the syntax of Recover are described in Section 6, where we allow *a priori* information $K \in \mathsf{Known}$ to be input to Recover, and allow coins $R \in \mathsf{Rand}$ to be output by Recover.

A paper on VSS by Bai, Damgård, Orlandi, and Xia [2] employed some related syntactic choice. In particular, their Share algorithm takes in an access structure, assumed to be described by a circuit. But it is not authenticated, returned during recovery, or guaranteed to be dropped into shares. The "public share" S_0 that their Share algorithm outputs resembles the public portion of a share from our own treatment. But the former was ac-

```
Game $\mathrm{G}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A})$

procedure MAIN
  $c \twoheadleftarrow \{0,1\}$;  $H \twoheadleftarrow \Omega$
  $q \leftarrow 0$;  $(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{I}^{\mathrm{DEAL}}$
  if $(\exists\, j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j]))$ then return false
  $c' \twoheadleftarrow \mathbb{A}^{H}(St, \boldsymbol{S}_1[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}_q[\boldsymbol{B}[q]], \boldsymbol{P})$
  return $(c = c')$

procedure DEAL$(A, M_0, M_1, R, T)$
  $q \leftarrow q + 1$;  $\boldsymbol{A}[q] \leftarrow A$
  $\boldsymbol{S}_q \leftarrow \mathbb{S}.\mathrm{Share}^{H}(A, M_c, R, T)$;  $\boldsymbol{P}[q] \leftarrow \boldsymbol{S}_q.\mathrm{pub}$
  return
```

```
Game $\mathrm{G}^{\mathrm{pred}}_{\mathbb{I}}(\mathbb{P})$

procedure MAIN
  $q \leftarrow 0$;  $(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{I}^{\mathrm{DEAL}}$
  $(M, R) \twoheadleftarrow \mathbb{P}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{T}, \boldsymbol{L}, St)$
  return $((M, R) \in D)$

procedure DEAL$(A, M_0, M_1, R, T)$
  $D \leftarrow D \cup \{(M_0, R), (M_1, R)\}$;  $q \leftarrow q + 1$
  $\boldsymbol{A}[q] \leftarrow A$;  $\boldsymbol{R}[q] \leftarrow R$;  $\boldsymbol{T}[q] \leftarrow T$;  $\boldsymbol{L}[q] \leftarrow |M_0|$
  return
```

Fig. 4. Defining privacy. Top: Game used for measuring Priv security of an ADSS scheme $\mathbb{S}$ relative to an input generator $\mathbb{I}$ and an adversary $\mathbb{A}$. Bottom: Game used for measuring the predictability of inputs selected by the input generator $\mathbb{I}$.

tually used to formalize a broadcast channel, which is not present in our model.

4 Privacy

THE IDEA. One way to formalize privacy for a classical secret-sharing scheme captures this idea: if an adversary obtains an unauthorized set of shares, this will tell it nothing about the message beyond that which it already knows [15]. Achieving this requires fresh, high-entropy coins with each sharing. In their absence, all bets are off. Our formulation generalizes this idea, following the idea of *hedging* [5], so that the guarantee above continues to hold, to the maximum extent possible, even when the coins are not good. We ask that if an adversary obtains an unauthorized set of shares, this will tell it nothing about the message M as long as the (M, R) pair was drawn from a set too large for the adversary to exhaust. Intuitively, this is the best possible, because if the adversary could exhaust this set then it could violate privacy by running the sharing algorithm on each candidate (M, R) to see which results are consistent with the shares it has seen.

What is the benefit of all of this? First, it enables reproducible secret-sharing with meaningful privacy guarantees. For example, the random input R might be chosen at software-installation time, then supplemented by a counter with each sharing. Share regeneration is now possible, but even if the adversary does get hold of the device containing R, privacy will be preserved as long as M itself is unpredictable and the adversary obtains only an unauthorized set of shares. For a classical secret-sharing scheme like Shamir's that wouldn't be true. A related benefit is for the sharing algorithm to work as well as possible in the presence of imperfect randomness. A cryptographic technique becomes safer to use when you can prove that it does not catastrophically fail when the randomness isn't perfect.

DEFINITION. Fix an adept secret-sharing scheme $\mathbb{S}$, an algorithm $\mathbb{I}$ called the *input-selector*, and an adversary $\mathbb{A}$ called the *privacy adversary*. Consider the $\mathrm{G}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A})$ game of Fig. 4. The Priv advantage of $\mathbb{A}$ relative to $\mathbb{I}$ is defined by

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}) = 2\Pr[\mathrm{G}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A})] - 1 .$$

We first explain the broad elements of the game, and then its fine points. The game picks a challenge bit c at random. The input-selector represents the dealer. It has a DEAL oracle via which it provides a pair of message $M_0, M_1 \in \mathbb{S}.\mathsf{Msg}$ that are required to be of the same length. It also provides an access-structure description $A \in \mathbb{S}.\mathsf{Access}$ and a tag $T \in \mathbb{S}.\mathsf{Tag}$. More unusually, it provides a string $R \in \mathbb{S}.\mathsf{Rand}$ that will be the randomness used by $\mathbb{S}$.Share. The randomness is chosen by the input-selector, not the game. In response to a query (A, M_0, M_1, T, R), oracle DEAL creates a vector $\boldsymbol{S}_i$ of shares by running $\mathbb{S}$.Share, the message being either M_0 or M_1 depending on the challenge bit c. The access structure and AD, and also the randomness, are taken from the query. The oracle may be called as often as the input-selector likes, but with the following *non-repetition condition*: if $(A_1, M_{1,0}, M_{1,1}, R_1, T_1), \ldots, (A_q, M_{q,0}, M_{q,1}, R_q, T_q)$ are the queries made, then the tuples $(A_1, M_{1,0}, R_1, T_1), \ldots, (A_q, M_{q,0}, R_q, T_q)$ are all distinct, and also the tuples $(A_1, M_{1,1}, R_1, T_1), \ldots, (A_q, M_{q,1}, R_q, T_q)$ are all distinct. So for both $c = 0$ and $c = 1$, the inputs provided to Share will be distinct. This is necessary because, otherwise, Share being deterministic, an adversary could trivially discover the challenge bit c.

The number of calls made is recorded in the variable q. As per line 29, nothing is returned to the adversary in response to a Deal query. This ensures that the inputs to Deal are chosen non-adaptively, a choice we will discuss later. The output of $\mathbb{I}$ consists of state information St, to be passed to its accomplice $\mathbb{A}$, and a q-vector $\boldsymbol{B}$ whose j-th component $\boldsymbol{B}[j] \subseteq [1..\mathrm{n}(\mathrm{Acc}(A[j]))]$, for each $j \in [1..q]$, is a set of parties that the input-selector is corrupting, meaning $\mathbb{I}$ is requesting the corresponding set of shares $\boldsymbol{S}_j[\boldsymbol{B}[j]]$ be provided to $\mathbb{A}$. If a set $\boldsymbol{B}[j]$ returned by $\mathbb{I}$ is authorized, the game immediately returns false. Otherwise, the privacy adversary is executed on input of the state information St from $\mathbb{I}$ and the sets of shares $\boldsymbol{S}_1[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}_q[\boldsymbol{B}[q]]$ of the corrupted parties, as well as the public parts of all shares dealt. It also gets the random oracle H, which was denied to $\mathbb{I}$. The privacy adversary returns its guess c' for the challenge bit c and wins (the game returns true) if this guess is correct.

Priv security is achievable only when the (M_0, R), (M_1, R) pairs in the Deal queries of $\mathbb{I}$ are unpredictable, as we now formalize, following [4, 5, 8]. Game $\mathrm{G}^{\mathrm{pred}}_{\mathbb{I}}(\mathbb{P})$ of Fig. 4 measures the predictability of an input-selector $\mathbb{I}$ via another adversary $\mathbb{P}$ called a *predictor*. The input-selector $\mathbb{I}$ is executed with its Deal oracle, the latter now simply recording the adversary queries: no secret sharing is done, and nothing is returned to the adversary. The predictor wins if it can predict (output) some secret-randomness pair that was returned by the adversary. Its input is that which we allow secret sharing to leak to the second stage: the access structures, tags, message lengths, which parties are corrupted, and the state returned by $\mathbb{A}$ in its first stage. The privacy-adversary $\mathbb{A}$ is not relevant here; unpredictability is a metric on the input-selector alone. We let

$$\mathbf{Adv}^{\mathrm{pred}}_{\mathbb{I}}(\mathbb{P}) = \Pr[\mathrm{G}^{\mathrm{pred}}_{\mathbb{I}}(\mathbb{P})] \quad \text{and}$$

$$\mathbf{pred}(\mathbb{I}) = \max_{\mathbb{P}} \left\{ \mathbf{Adv}^{\mathrm{pred}}_{\mathbb{I}}(\mathbb{P}) \right\} .$$

The notation reflects that $\mathbb{I}$ is the object whose security (in the sense of unpredictability) is being measured and $\mathbb{P}$ is the adversary. The max is over all predictor adversaries $\mathbb{P}$, regardless of their running time or the number of H queries they make. Thus $\mathbf{pred}(\mathbb{I})$ measures the information-theoretic guessing probability. The min-entropy of $\mathbb{I}$ could be defined as the negative log of this probability, but we do not need this.

Recovering classical privacy. Classical privacy corresponds to Priv security restricted to a class of input-selectors denoted $\mathbb{I}^{\mathrm{priv}\$}$. An input-selector $\mathbb{I}$ is in this class if there is an input-selector $\mathbb{I}_1$ and an integer r such that $\mathbb{I}$ is defined as follows: it lets $(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{I}_1^{\mathrm{Deal}^*}$ and returns $(St, \boldsymbol{B})$. Here Deal* is a subroutine defined by $\mathbb{I}$ as follows: On input a query (A, M_0, M_1, R, T) made by $\mathbb{I}_1$, input-selector $\mathbb{I}$ picks $R^* \twoheadleftarrow \{0,1\}^r$, queries its own Deal oracle with (A, M_0, M_1, R^*, T), and returns. For such an input-selector, we drop the non-repeating requirement; we expect that r is large, in which case non-repetition holds with high probability. For emphasis, we can in this case write $\mathbf{Adv}^{\mathrm{priv}\$}_{\mathbb{S},\mathbb{I}}(\mathbb{A})$ in place of $\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A})$. Note that $\mathbf{pred}(\mathbb{I}) \leq q \cdot 2^{-r}$ where q is the number of Deal queries of $\mathbb{I}$.

Discussion. In an asymptotic-security setting, where all advantages are functions of a security parameter, we would say that $\mathbb{I}$ is *unpredictable* if $\mathbf{pred}(\mathbb{I})$ is negligible. Then we would say that $\mathbb{S}$ is Priv-*secure* if $\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{S},\mathbb{I}}(\mathbb{A})$ is negligible for every polynomial-time, unpredictable $\mathbb{I}$ and every polynomial-time $\mathbb{A}$. In our concrete-security setting, we will informally use the terms in italics above with the understanding that polynomial-time means "efficient" and negligible means "small." Results will make this precise via concrete bounds on advantage. For example, Theorem 1 upper-bounds $\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{SS},\mathbb{III}}(\mathbb{AA})$ as a function of $\mathbf{pred}(\mathbb{III})$, so that if the latter is small ($\mathbb{III}$ is unpredictable) then the former is too ($\mathbb{SS}$ is Priv-secure). Unpredictability is necessary for Priv security in the same way that it is necessary for the privacy of deterministic public-key encryption [4].

Denying $\mathbb{I}$ access to the hash function H is necessary to achieve Priv for the same reason that messages in deterministic public-key encryption cannot depend on the public key [4] and in message-locked encryption cannot depend on the parameters [10]. From a usage perspective, this models dealers (users) picking the inputs to $\mathbb{S}$.Share independently of H, which is what we expect real users to do. This is analogous to the argument that users of deterministic public-key encryption will not usually encrypt messages that depend on the public key of the recipient [4]. For both deterministic public-key encryption and message-locked encryption, allowing messages to depend on the public key or parameters (respectively) has been considered [1, 7, 35]. Doing the same here is an open question.

Our formalizations capture non-adaptive privacy, meaning that secrets (as well as the access structure, set of corrupted parties and the randomness) are not chosen as a function of the shares the adversary sees of previously shared secrets. This is in general necessary for Priv. In the case that the randomness is true and independent across sharing, stronger privacy (adaptive and with $\mathbb{I}$ allowed access to Hash) is possible and in fact achieved by our schemes. But we prefer the simplicity of a single definition to pursuing this because in usage, inputs to $\mathbb{S}$.Share are chosen by users who are unlikely to pick them adaptively or in a way depending on H.

Usage scenarios and their privacy. Different choices of randomness, made by a combination of user and scheme choices, are captured by different classes of input-selectors. We discuss a few.

The $\mathbb{S}$.Share interface of an implementation could give the caller options with regard to R, effectively asking: *do you want to pick the coins, or do you want the implementation to?* If the user selects the latter, the implementation could pick R uniformly random from a large space for each invocation of Share. This would be captured as the class of input-selectors that pick R in this way, and, for that class, achieving the definition confers the standard indistinguishability-style privacy. However, it precludes reproducibility. A user desiring the latter could select the option of itself providing R, and then has various choices of how to do so. It may omit it altogether, setting R to the empty string, corresponding to an $\mathbb{I}$ that does the same, so that privacy is provided as long as the message alone is unpredictable, as is possible if it is a good passphrase. To strengthen privacy in the case the message may lack entropy, the user could maintain a separate, long-term, high quality password, always using this in the role of R. Finally, the input-selector could choose such a long-lived R, but then append a counter. All of these possibilities are modeled as different choices of $\mathbb{I}$.

5 Authenticity

Authenticity captures the immutability of what is shared: if a dealer shares out M, then nothing *else* can be recovered, even if some shares are changed. One could call the desired aim a *binding* property—one of the goals of a commitment scheme.

We give two notions of authenticity, Auth0 and Auth. The former assumes an honest dealer. For the use cases we have considered, it is sufficient. The Auth notion is simpler and implies Auth0. It does not assume an honest dealer. We take Auth as our main definitional target, but retain Auth0 to clarify the key security aim that Auth ensures.

The Auth0 goal. Our first notion for authenticity, Auth0, says that if you receive a share from an honest dealer, contribute it to Recover, and a secret is recovered using your share, then that secret must be what the dealer originally shared. In brief, a valid share is a commitment to the secret that was shared at that time. This holds no matter what other parties do.

```
Game $\mathrm{G}^{\mathrm{auth0}}_{\mathbb{S}}(\mathbb{A})$

40 $H \twoheadleftarrow \Omega$
41 $(A, M, T, St) \twoheadleftarrow \mathbb{A}^{H}$;  $R \twoheadleftarrow \mathbb{S}.\mathsf{Rand}$
42 $\boldsymbol{S} \leftarrow \mathbb{S}.\mathrm{Share}^{H}(A, M, R, T)$
43 $\mathcal{S} \leftarrow \{\boldsymbol{S}[i]\colon\ i \in [1..|\boldsymbol{S}|]\}$
44 $\mathcal{S}' \twoheadleftarrow \mathbb{A}^{H}(St, \boldsymbol{S})$
45 $(M', \mathcal{V}') \leftarrow \mathbb{S}.\mathrm{Recover}^{H}(\mathcal{S}')$
46 return $\mathcal{S} \cap \mathcal{V}' \neq \emptyset$ and $M \neq M'$

Game $\mathrm{G}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A})$

50 $H \twoheadleftarrow \Omega$
51 $(\mathcal{S}, \mathcal{S}') \twoheadleftarrow \mathbb{A}^{H}$
52 $(M, \mathcal{V}) \leftarrow \mathbb{S}.\mathrm{Recover}^{H}(\mathcal{S})$
53 $(M', \mathcal{V}') \leftarrow \mathbb{S}.\mathrm{Recover}^{H}(\mathcal{S}')$
54 return $\mathcal{V} \cap \mathcal{V}' \neq \emptyset$ and $M \neq M'$
```

Fig. 5. Defining authenticity. Games Auth0 and Auth capture security of $\mathbb{S}$ against adversary $\mathbb{A} = \mathbb{A}^{H}$. If $\perp$ is parsed into components at lines 52 or 53, the the first is $\perp$ and the second is $\emptyset$.

The definition of Auth0 employs the game $\mathrm{G}^{\mathrm{auth0}}_{\mathbb{S}}(\mathbb{A})$ defined in Fig. 5. An adversary $\mathbb{A}$ attacking Auth0 security runs a first stage to pick (A, M, T). The sharing algorithm is then run on these values, along with uniformly random coins R, to produce a vector of shares $\boldsymbol{S}$. The adversary, given $\boldsymbol{S}$ (and whatever state she wants to retain from her first stage of execution, St), must now find a set of shares $\mathcal{S}'$ that has some share S in common with those in $\boldsymbol{S}$. It wins if recovering a secret from $\mathcal{S}'$ results in some message M' distinct from M and employing the share S. Formally, we define $\mathbf{Adv}^{\mathrm{auth0}}_{\mathbb{S}}(\mathbb{A}) = \Pr[\mathrm{G}^{\mathrm{auth0}}_{\mathbb{S}}(\mathbb{A})]$ as the probability that the specified game returns true. Note that the game depends on the selection of a random oracle H, which we let $\mathbb{A}$ query. Note that the common share S of our English exposition is not explicit in the game, but is an arbitrary element of $\mathcal{V} \cap \mathcal{V}'$. Recall that the second argument $(M, \mathcal{V})$ returned by a call to Recover is the set of shares deemed valid.

The Auth goal. There is a natural way to strengthen and simplify Auth0. A game that does so is again defined in Fig. 5. Rather than insisting that shares arise from honestly sharing out a secret, we let the adversary name two sets of shares, $\mathcal{S}$ and $\mathcal{S}'$, in whatever way it likes. Recovery is then performed on both sets of shares. The adversary wins if the two sets of shares have at least one share S in common, that share is used in recovery operations, but the messages recovered differ. Note that strings are recovered if the adversary wins (that is, $M \neq \bot \neq M'$), because $\mathcal{V} \neq \bot \neq \mathcal{V}'$. We let $\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A}) = \Pr[\mathrm{G}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A})]$ be the probability that the game returns true.

Auth implies is stronger than Auth0, as the adversary certainly has the *option* of creating $\mathcal{S}$ by sharing out some (A, M, T) of its choice.

We can summarize the difference between Auth0 and Auth by saying that, in the former, a good share commits the dealer to at most one M, while in the latter, any share, good or bogus, commits to at most one M. Auth0 speaks to what a party can believe if it gets a share from an honest dealer; Auth speaks to what can be believed if the share is of unknown provenance.

We prefer Auth to Auth0 because it is simpler and stronger. For applications the extra strength would usually be irrelevant: in our motivating use cases, legitimate shareholders are assumed to hold valid shares.

We comment that having Recover identify which shares are good is important to making the authenticity guarantee meaningful. In particular, a party holding a share S it believes to be valid and who recovers a message M should only accept M as the underlying secret if her share S was identified as one of the good shares.

6 Error Correction

Informal description. Basic correctness demands that Recover($\mathcal{S}$) return $(M, \mathcal{S})$ when $\mathcal{S}$ is an authorized subset of some sharing of M. But what should Recover do when $\mathcal{S}$ is *not* an authorized subset of any sharing of M? One possibility is to have it return $\bot$, thereby signaling that *something* is wrong. One might call such a scheme *error-detecting.*

Error-detection comes with a liability: it enables the adversary to thwart message recovery by changing a single share. There is no attempt to fix any problem. *Error correction* (Errx) goes to the opposite extreme: we seek to recover from errors whenever we can.

Error correction can be regarded as an exercise in *explanation seeking.* The Recover algorithm is presented with a set of shares. If there is a *unique* explanation for how they arose, we demand that Recover find this explanation. Given shares $\mathcal{S}$, the explanation will say: "Here is the message M that was previously shared out to give a subset $\mathcal{V} \neq \emptyset$ of the shares $\mathcal{S}$. The remaining shares from $\mathcal{S}$ are all bad." If there is no unique explanation like this, then an Errx-secure scheme must say so. Note that we disregard the *degenerate* explanation in which *all* shares are bad. That explanation is *always* a possibility, so an uninteresting one.

In this section we formalize Errx security. In Section 7 we show how to achieve Errx security, while in Appendix A.2 we compare it to *robustness* [15, 31].

Enriching the syntax. For ADSS schemes that target error correction we enrich the syntax for the Recover algorithm in two directions.

First, we allow *known information* to be identified in Recover's input. This information $K \in \mathsf{Known} = \mathsf{Access} \cup \mathsf{Shares}$ is either an access structure $K = A \in \mathsf{Access}$ that the recovering party somehow knows to be the operative one, or it is the subset $K \in \mathsf{Shares}$ of the shares $\mathcal{S}$ given to Recover know to be valid.

Second, we demand that the recovery process identify not only the message M and the valid shares $\mathcal{V} \subseteq \mathcal{S}$ but also the randomness R that was used in the sharing.

Formally, we will say that an *enriched* ADSS scheme $\Pi = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ has Acc and Share as before but the message-recovery algorithm Recover: $\mathsf{Known} \times \mathsf{Shares} \to \mathsf{Msg} \times \mathsf{Rand} \times \mathsf{Shares} \ \cup \ \{\bot\}$ gets the indicated domain and range. We demand that Recover respects the known information: if $A \in \mathsf{Access} \cap \mathsf{Known}$ and $\mathrm{Recover}(A, \mathcal{S}) = (M, R, \mathcal{V})$ then $\mathcal{V}.\mathrm{as} = A$; and if $\mathcal{G} \in \mathsf{Shares} \cap \mathsf{Known}$ and $\mathrm{Recover}(\mathcal{G}, \mathcal{S}) = (M, R, \mathcal{V})$ then $\mathcal{G} \subseteq \mathcal{V}$. If the recovering party has no *a priori* information, select $K = \mathcal{G} = \emptyset$.

As before, if $\mathrm{Recover}(K, \mathcal{S}) = (M, R, \mathcal{V})$ then $\mathcal{V} \subseteq \mathcal{S}$ and all shares from $\mathcal{V}$ have the same .as component and the same .tag component. As for the return value R, we now describe our expectations.

Full correctness. For $\Pi = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ an enriched ADSS scheme, we adjust basic correctness to demand that the identified coins are right: for all $A \in \mathsf{Access}$, $H \in \Omega$, $I \subseteq [1..\mathrm{n}(\mathrm{Acc}(A))]$, $M \in \mathsf{Msg}$, $R \in \mathsf{Rand}$, $T \in \mathsf{Tag}$, $\boldsymbol{S} \leftarrow \mathrm{Share}^H(A, M, R, T)$, $\mathcal{S} = \boldsymbol{S}[I]$, and $K \in \{A\} \cup \mathcal{P}(\boldsymbol{S})$ (where $\mathcal{P}(\boldsymbol{S})$ is all subsets of the components of $\boldsymbol{S}$): (1) if $I \in \mathrm{Acc}(A)$ then $\mathrm{Recover}^H(K, \mathcal{S}) = (M, R, \mathcal{S})$, and (2) if $I \notin \mathrm{Acc}(A)$ then $\mathrm{Recover}^H(K, \mathcal{S}) = \bot$. If all you are worried about is vanishing shares then Recover returns the right thing.

The following *validity* requirement for an enriched ADSS scheme $\Pi = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ can be considered a converse to basic correctness: when $\mathrm{Recover}(K, \mathcal{S}) = (M, R, \mathcal{V})$ and $\boldsymbol{S} = \mathsf{Share}(\mathcal{V}.\mathrm{as}, M, R, \mathcal{V}.\mathrm{tag})$ then $\mathcal{V}$ is an authorized subset of $\boldsymbol{S}$, meaning that $\mathcal{V} = \boldsymbol{S}[G]$ for some $G \in \mathrm{Acc}(A)$,

```
Game $\mathrm{G}^{\mathrm{errx}}_{\mathbb{S}}(\mathbb{A})$

70  $H \twoheadleftarrow \Omega$ ; $(K, \mathcal{S}) \twoheadleftarrow \mathbb{A}^H$
71  return $\mathbb{S}.\mathrm{Recover}^H(\mathcal{K}, \mathcal{S}) \neq$ UniqueExplanation$^H(K, \mathcal{S})$

72  procedure UniqueExplanation$^H(K, \mathcal{S})$
73  if $\exists(A, M, R, \mathcal{V}) \in$ Explanations$^H(K, \mathcal{S})$ such that
74      $(\hat{A}, \hat{M}, \hat{R}, \hat{\mathcal{V}}) \in$ Explanations$^H(K, \mathcal{S}) \Rightarrow$
75      $(A = \hat{A} \wedge M = \hat{M} \wedge R = \hat{R} \wedge \mathcal{V} \supseteq \hat{\mathcal{V}})$
76  then return this (necessarily unique) $(M, R, \mathcal{V})$
77  else return $\perp$

78  procedure Explanations$^H(K, \mathcal{S})$
79  return $\{(\mathcal{V}.\mathrm{as}, M, R, \mathcal{V}):$
7A      $\hat{\mathcal{S}} \subseteq \mathcal{S}, (M, R, \mathcal{V}) = \mathbb{S}.\mathrm{Recover}^H(K, \hat{\mathcal{S}})\}$
```

Fig. 6. Defining error correction. We define an adversary $\mathbb{A}$'s errx-advantage for $\mathbb{S} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ as the probability it wins the specified game.

$A = \mathcal{V}.\mathrm{as}$. Informally, Recover($\mathcal{S}$) does not lie by identifying an $(M, R, \mathcal{V})$ that doesn't work. Such lying would be pointless, as the party recovering can verify that $\mathcal{V}$ is an authorized subset of Share($\mathcal{V}.\mathrm{as}, M, R, \mathcal{V}.\mathrm{tag}$). An enriched ADSS scheme is *fully correct* if it satisfies basic correctness and validity. When we speak of an ADSS scheme achieving Errx security, we always assume it is fully correct.

Adjusting Auth. Enriching ADSS syntax is irrelevant for Priv security because that notion does not depend on the Recover algorithm. On the other hand, the Auth security notion, previously defined by the game of Fig. 5, needs a slight adjustment. The code of that game is replaced with:

```
50   $H \twoheadleftarrow \Omega$
51′  $(K, \mathcal{S}, K', \mathcal{S}') \twoheadleftarrow \mathbb{A}^H$
52′  $(M, R, \mathcal{V}) \leftarrow \mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S})$
53′  $(M', R', \mathcal{V}') \leftarrow \mathbb{S}.\mathrm{Recover}^H(K', \mathcal{S}')$
54   return $\mathcal{V} \cap \mathcal{V}' \neq \emptyset$ and $M \neq M'$
```

The above continues to capture that a share commits to single underlying secret. By changing the "$M \neq M'$ (line 54) to "$(M, R) \neq (M', R')$" we would capture the idea that a share commits to a secret and coins. Our main construction achieves this stronger variant as well.

Errx security. We now define the Errx security of an ADSS scheme Π. See Fig. 6. We then define $\mathbf{Adv}^{\mathrm{errx}}_{\mathbb{S}}(\mathbb{A}) = \Pr[\mathrm{G}^{\mathrm{errx}}_{\mathbb{S}}(\mathbb{A})]$ as the probability that the adversary wins the error-correction game. An ADSS scheme $\mathbb{S}$ has perfect error correction if it never fails to correct a correctable error: $\mathbf{Adv}^{\mathrm{errx}}_{\mathbb{S}}(\mathbb{A}) = 0$ for any $\mathbb{A}$.

The error-correction game is structured in a way to directly reflect the intended intuition. The adversary wins if it forces Recover to recover something *wrong*—something other than the unique explanation, when there is one, for the provided shares. We carry out the thought experiment of looking at all plausible explanations for the shares, and see if one is unique. At the lowest level, at lines 78–7A, the plausible explanations are indicated by the Recover procedure itself.

Alternative Errx definition. There is an equivalent way to define Errx security: one defines the plausible explanations for a set of shares according to the Share procedure instead of the Recover procedures. Specifically, lines 78–7A of Fig. 6 can be replaced by the following:

```
78′  procedure Explanations$^H(K, \mathcal{S})$
79′  if $K \in$ Access then return $\{(K, M, R, \mathcal{V}):$
7A′      $G \in \mathrm{Acc}(K), R \in$ Rand$, T \in$ Tag$,$
7B′      $\boldsymbol{S} = \mathrm{Share}^H(K, M, R, T), \mathcal{V} = \boldsymbol{S}[G] \subseteq \mathcal{S}\}$
7C′  else return $\{(A, M, R, \mathcal{V}):\ A \in$ Access$,$
7D′      $G \in \mathrm{Acc}(A), R \in$ Rand$, T \in$ Tag$,$
7E′      $\boldsymbol{S} = \mathrm{Share}^H(A, M, R, T), \mathcal{V} = \boldsymbol{S}[G] \subseteq \mathcal{S}, K \subseteq \mathcal{V}\}$
```

No other changes are made. We justify the equivalence of the definitions in Appendix C.2.

Discussion. We defined ADSS in such a way that the reconstructing party is not required to know the operative access structure; rather, it recovers this from the valid shares. This choice interacts badly with use of an expressive access-structure naming function. Suppose, for example, that Acc supports the 1-out-of-1 threshold scheme. Then an adversary can replace a single share S from a deal $\mathcal{S}$ with a share S_1 for a message M_1, the share asserting the 1-out-of-1 access structure. Message recovery will either be thwarted by the presence of this one bogus share (when $\mathcal{S} \setminus \{S\}$ is qualified), or $(M_1, \{S_1\})$ will be recovered (when $\mathcal{S} \setminus \{S\}$ is not qualified). Neither outcome is good.

The underlying problem is a failure to distinguish between the access structures that a secret-sharing scheme can handle and those that a reconstructing party might regard as reasonable. Once that distinction is drawn, one can consider it an important but out-of-model step that the recovering party discards any share asserting an access structure it deems unreasonable. A simple special is when the receiver knows what the right access structure is. It can provide this side information to Recover.

A well-known variant of secret sharing [41] envisages that a shareholder who trusts her own share S is performing recovery. In such a case, explanations from Recover that do not include this share should be regarded as implausible. More generally, we have enriched Recover so that any subset of shares can be designated as trusted. Only explanations that include all trusted shares are considered valid. Note that marking any share as trusted establishes a known access-structure, too. Both make it harder for an adversary to obstruct message recovery.

One way for the receiving party to obtain assurance that a given share is trusted is for the share to be digitally signed by the dealer and for the reconstructing party to know the dealer's public key. Such a model for secret sharing meaningfully disadvantages the adversary, but takes us outside our basic model.

Coin recovery. Our enriched syntax demands that Recover, when presented the set of shares $\mathcal{S}$, return not only M and $\mathcal{V}$ but also the coins R that were used in the sharing of M and gave rise to $\mathcal{V} \subseteq \mathcal{S}$. Why?

The returned coins serve as a certificate that the the valid shares really could arise from a legal sharing of the message M. Beyond this, a unique explanation $(M, R, \mathcal{V})$ for the set of shares $\mathcal{S}$ becomes a demonstration that, for an honest dealer, it was a single sharing of M that gave rise to shares $\mathcal{V}$. In effect, returning R and absorbing it into the Errx definition makes the definition stronger, ensuring that it was *one* sharing from which we are seeing shares. It eliminates degeneracies about what Recover should do when, for example, two shares of a 2-out-of-4 secret-sharing are combined with two shares from a different 2-out-of-4 secret-sharing for the same message. Such possibilities returning $\perp$ (as we think they should) would thwart claims that Errx security imply robustness; they would make it untrue. We find that to be undesirable: error-correction intuitively *should* imply robustness (once side conditions are added so that robustness becomes achievable), but with other definitional choices we explored that do not surface R, such a claim is untrue.

7 Constructions

This section provides schemes and transformations for achieving ADSS. We start with a version of Shamir's secret-sharing scheme, $\mathbb{S}1$. It achieves classical privacy, Priv\$, and works for threshold access structures. We then provide the main construction of this paper: the transformation AX. It turns an ADSS scheme $\mathbb{S}$ achieving only Priv\$ security into an ADSS scheme $\mathbb{SS} = \mathrm{AX}[\mathbb{S}]$ that achieves Priv and Auth security. Finally, transformation EX adds in Errx security. Proofs for our constructions are in Appendix C.

```
procedure $\mathbb{S}1.\mathrm{Share}(A, M, R, T)$
100 $\langle k, n\rangle \leftarrow A$
101 $M_1 \| \cdots \| M_m \leftarrow M$ where $|M_1| = \cdots = |M_m| = \beta$
102 for $(i, j) \in [1..(k-1)] \times [1..m]$ do $R_{j,i} \leftarrow f_R^{\beta}(i, j)$
103 for $i \in [1..n]$ do
104   for $j \in [1..m]$ do
105     $B_{i,j} \leftarrow M_j + R_{j,1} \cdot i + R_{j,2} \cdot i^2 + \cdots + R_{j,k-1} \cdot i^{k-1}$
106   $S_i \leftarrow \langle i, \langle k, n\rangle, B_{i,1} \cdots B_{i,m}, \varepsilon, \varepsilon\rangle$
107 return $(S_1, \ldots, S_n)$

procedure $\mathbb{S}1.\mathrm{Recover}(\mathcal{S})$
110 $t \leftarrow |\mathcal{S}|$;  $\{S_1, \ldots, S_t\} \leftarrow \mathcal{S}$
111 for $i \in [1..t]$ do $\langle \iota_i, \langle k_i, n_i\rangle, B_{i,1} \cdots B_{i,m_i}, \varepsilon, \varepsilon\rangle \leftarrow S_i$
112 $(k, n, m) \leftarrow (k_1, n_1, m_1)$
113 if $t < k$ then return $\perp$
114 for $j \in [1..m]$ do
115   $\varphi_j(x) \leftarrow \mathrm{Interpolate}_\beta(\{(\iota_1, B_{1,j}), \ldots, (\iota_k, B_{k,j})\})$
116   $M_j \leftarrow \varphi_j(0)$
117 return $(M_1 \cdots M_m,\ \mathcal{S})$
```

Fig. 7. Secret-sharing scheme $\mathbb{S}1$. Scheme $\mathbb{S}1 = \mathbb{S}1[\beta, f]$ depends on the number β and a PRF $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \rightarrow \{0,1\}^*$ satisfying $|f_R^\ell(\cdot)| = \ell$. The message space is $\mathsf{Msg} = (\{0,1\}^\beta)^*$, the entropy space is $\mathsf{Rand} = \{0,1\}^\kappa$, the AD space is $\mathsf{Tag} = \{\varepsilon\}$. Lines 105, 115, 116 do arithmetic in the filed $\mathbb{F}$ with 2^β points. $\mathrm{Interpolate}_\beta$ takes a set of points in $\mathbb{F}^2$ and returns the unique minimal-degree polynomial over $\mathbb{F}$ that passes through them. If a value cannot be parsed as indicated, the routine returns $\perp$.

Transformations AX and EX leave unchanged the access structure of the scheme they are applied to, so $\mathbb{SS}1 = \mathrm{EX} \circ \mathrm{AX} \circ \mathbb{S}1$ is the the concrete ADSS scheme we put forward for threshold access structures. To handle arbitrary access structures all that is needed is to start with a base-level scheme that works for arbitrary access structures. We give such a scheme, $\mathbb{S}2$, in Appendix B. The access structures is presented as a circuit of threshold gates. Scheme $\mathbb{SS}2 = \mathrm{EX} \circ \mathrm{AX} \circ \mathbb{S}2$ is our suggestion for an ADSS scheme on arbitrary access structures.

We discuss the efficiency of our schemes at the end of this section.

Base-level scheme S1. We begin by describing Shamir secret-sharing [40], but with a few minor twists: scheme $\mathbb{S}1$ operates over the field $\mathbb{F}$ with 2^β points and is extended blockwise; the polynomial coefficients are determined by a pseudorandom generator (PRG) based on a pseudorandom function (PRF); and, in keeping with our syntax, a description of the (threshold) access structure is an input to Share. Concretely, Fig. 7 defines secret-sharing scheme $\mathbb{S}1 = \mathbb{S}1[\beta, f]$ where

(1) $\beta \geq 2$ is the *blocklength.* In practice, one would likely select $\beta = 8$, corresponding to the partitioning of a plaintext into bytes; and

(2) $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$ formalizes how the entropy source $R \in \{0,1\}^\kappa$ is used to create the internal randomness. We require $|f_R^\ell(\boldsymbol{x})| = \ell$ (the first two arguments of f written as a subscript then superscript).

The set $\mathbb{S}1.\mathsf{Access}$ contains all $\langle k, n\rangle$ (a string that encodes k and n) where $1 \leq k \leq n < 2^\beta$. The access-structure naming function $\mathbb{S}1.\mathrm{Acc}$ maps each $\langle k, n\rangle \in \mathbb{S}1.\mathsf{Access}$ to the set $\mathcal{A}_{k,n} = \{U \in [1..n] : |U| \geq k\}$. The message space of $\mathbb{S}1$ is $\mathsf{Msg} = \mathsf{B}^*$ where $\mathsf{B} = \{0,1\}^\beta$. The randomness space is $\mathsf{Rand} = \{0,1\}^\kappa$. The scheme uses the finite field $\mathbb{F}$ having 2^β points, which must be more than the maximum number of parties. We fix some canonical representation of field points as β-bit strings. We interchangeably regard β-bit strings, numbers in $[0..2^\beta - 1]$, and points in $\mathbb{F}$. For lines 106 and 111 recall that the fourth and fifth components of a share S_i represent the public portion $S_i.\mathrm{pub}$ and the tag $S_i.\mathrm{tag}$. Both are ε since scheme $\mathbb{S}1$ doesn't support tags and doesn't mark any portion of a share as public.

The security of $\mathbb{S}1$ relies on the PRF security of f, which is defined in Appendix C.1. We give the following proposition, which states that if f is a secure PRF, then $\mathbb{S}1[\beta, f]$ is Priv\$ secure. Recall the latter is Priv restricted to input-selectors in the class $\mathbb{I}^{\mathrm{priv\$}}$, namely those who pick the coins in their Deal queries uniformly and independently of anything else.

Proposition 1. *Let* $\mathbb{S}1 = \mathbb{S}1[\beta, f]$ *with* $\beta \geq 2$ *and* $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$. *Then* $\mathbb{S}1$ *satisfies* Priv\$. *Concretely, given input-selector* $\mathbb{I} \in \mathbb{I}^{\mathrm{priv\$}}$ *and given Priv-adversary* $\mathbb{A}$ *we build a PRF-adversary* $\mathbb{B}$ *such that* $\mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S}1,\mathbb{I}}(\mathbb{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B})$. *Adversary* $\mathbb{B}$ *is efficient when* $\mathbb{I}$ *and* $\mathbb{A}$ *are.*

Main construction AX. The AX transformation turns a Priv\$-secure secret-sharing scheme $\mathbb{S}$ into a secret-sharing scheme $\mathbb{SS}$ that augments this with Priv- and Auth-security. $\mathbb{SS}$ uses the enriched ADSS syntax but does not yet target error correction; that will come next. The AX transformation also expands the message space—scheme $\mathbb{SS}$ can share messages of any length, while $\mathbb{S}$ might only be able to share short ones. AX also handles associated-data, which scheme Shares is not required to support. The access structures that can be handled by $\mathbb{SS}$ are exactly those that can be handled by Share. Besides the secret-sharing scheme $\mathbb{S}$ the transformation will use PRF and a random-oracle-modeled hash-function. The former can be built from the latter, but we leave them separate because we anticipate, for example, an AES-based construction for the PRF and a SHA256-based construction for the hash.

The AX transformation is given in Fig. 8. It specifies $\mathbb{SS}$.Share and $\mathbb{SS}$.Recover for $\mathbb{SS} = \mathrm{AX}[\mathbb{S}, f]$. Access-structure naming function $\mathbb{SS}$.Acc is $\mathbb{S}$.Acc.

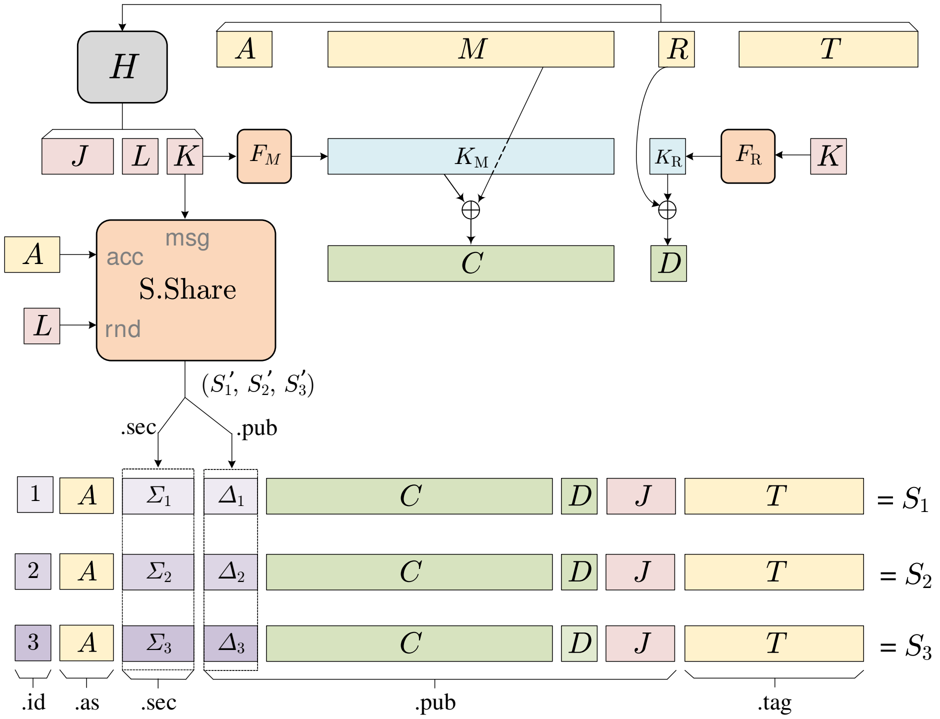
Theorem 1. *Let* $\mathbb{SS} = \mathrm{AX}[\mathbb{S}, f]$ *where* $\mathbb{S}$ *is an ADSS scheme with message space* $\mathbb{S}.\mathsf{Msg} \supseteq \{0,1\}^\kappa$*, tag space* $\mathbb{S}.\mathsf{Tag} = \{\varepsilon\}$*, and entropy space* $\mathbb{S}.\mathsf{Rand} = \{0,1\}^\kappa$*, and where* $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$. *Then:*

1. *If* $\mathbb{S}$ *is* Priv\$*-secure then* $\mathbb{SS}$ *is* Priv*-secure. Given an input-selector* $\mathbb{II}$ *making* q_{D} *calls to* Deal *and adversary* $\mathbb{AA}$ *(attacking the* Priv *security of* $\mathbb{SS}$*) making* q *queries to* Hash*, we build input-selector* $\mathbb{I} \in \mathbb{I}^{\mathrm{priv\$}}$ *and adversaries* $\mathbb{A}$ *and* $\mathbb{B}$ *s.t.*
$$\begin{aligned}\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{SS},\mathbb{II}}(\mathbb{AA}) &\leq 2(q_{\mathrm{D}} + q)\mathbf{pred}(\mathbb{II}) + \\ &\quad 4\,\mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}) + 4\,\mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B})\ .\end{aligned} \tag{1}$$
Adversaries $\mathbb{A}, \mathbb{B}$ *are about as efficient as* $\mathbb{AA}$.
2. $\mathbb{SS}$ *is* Auth*-secure. For any* $\mathbb{A}$ *making* q_{H} *queries to* Hash*, we have* $\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{SS}}(\mathbb{A}) \leq (q_{\mathrm{H}} + 1)(q_{\mathrm{H}} + 2) \cdot 2^{-(2\kappa+1)}$.

A more explicit theorem statement would quantify the resources of the constructed adversaries. In this case, adversary $\mathbb{A}$ makes q_{D} queries to Deal and q queries to Hash, while $\mathbb{B}$ makes q_{D} queries to New and 2 queries per instance to Fn. The running times of $\mathbb{A}$ and $\mathbb{B}$ are about the same as that of $\mathbb{AA}$. For brevity we omit such details in this and other result statements. They can be gleaned from the proofs.

Error correction with EX. Fig. 9 defines a transformation EX that turns an enriched ADSS scheme $\mathbb{S}$ into an error-correcting scheme $\mathbb{SS}$. EX avoids making any changes to Share, putting all the work in Recover.

For line 80, a set of shares $\mathcal{S}'$ is said to be K*-plausible* if all the shares in $\mathcal{S}'$ name the same access-structure encoding A, and $A = K$ if K names an access structure; the shares name distinct identities in $[1..\mathrm{n}(\mathrm{Acc}(A))]$; these parties are authorized according to $\mathrm{Acc}(A)$; the shares all have the same tag T; and the shares include all those in K if K is a set of shares. Further scheme-dependent criteria can be added without harming the



```
procedure $\mathbb{SS}.\text{Share}^{\text{Hash}}(A, M, R, T)$
$J \parallel K \parallel L \leftarrow H^{4\kappa}(A, M, R, T)$ where $|J| = 2\kappa$ and $|K| = |L| = \kappa$
$C \leftarrow M \oplus f_K^{|M|}(\varepsilon)$;  $D \leftarrow R \oplus f_K^{\kappa}(0)$;  $n \leftarrow \text{n}(\mathbb{S}.\text{Acc}(A))$
$(S'_1, \ldots, S'_n) \leftarrow \mathbb{S}.\text{Share}^h(A, K, L, \varepsilon)$
for $i \in [1..n]$ do
    $\Sigma_i \leftarrow S'_i.\text{sec}$;  $\Delta_i \leftarrow S'_i.\text{pub}$
for $i \in [1..n]$ do
    $P_i \leftarrow \langle \Delta_i, C, D, J \rangle$
    $S_i \leftarrow \langle i, A, \Sigma_i, P_i, T \rangle$
return $(S_1, \ldots, S_n)$

procedure $h^{\ell}(\boldsymbol{x})$
return $\text{Hash}(\ell, 0 \parallel \boldsymbol{x})$
procedure $H^{\ell}(\boldsymbol{x})$
return $\text{Hash}(\ell, 1 \parallel \boldsymbol{x})$
```

```
procedure $\mathbb{SS}.\text{Recover}^{\text{Hash}}(K, \mathcal{S})$
$t \leftarrow |\mathcal{S}|$;  $\{S_1, \ldots, S_t\} \leftarrow \mathcal{S}$
for $i \in [1..t]$ do
    $\langle j_i, A_i, \Sigma_i, P_i, T_i \rangle \leftarrow S_i$
    $\langle \Delta_i, C_i, D_i, J_i \rangle \leftarrow P_i$
    $S'_i \leftarrow \langle j_i, A_i, \Sigma_i, \Delta_i, \varepsilon \rangle$
$\mathcal{S}' \leftarrow \{S'_i \colon i \in [1..t]\}$
$(K, \mathcal{G}) \leftarrow \mathbb{S}.\text{Recover}^h(\mathcal{S}')$
$\langle \cdot, A, \cdot, \langle \cdot, C, D, J \rangle, T \rangle \leftarrow S_1$
$M \leftarrow C \oplus f_K^{|C|}(\varepsilon)$;  $R \leftarrow D \oplus f_K^{\kappa}(0)$
if $H^{4\kappa}(A,M,R,T)[1..3\kappa] = J \| K$
    and $\mathcal{S}.\text{id} \in \mathbb{S}.\text{Acc}(A)$
    and $\mathcal{S} \subseteq \mathbb{SS}.\text{Share}^{\text{Hash}}(A, M, R, T)$
then return $(M, R, \mathcal{S})$ else return $\perp$
```

Fig. 8. The AX transform $\mathbb{SS} = \text{AX}[\mathbb{S}, f]$. The construction depends on a Priv\$-secure $\mathbb{S}$ and a PRF $f \colon \{0,1\}^{\kappa} \times \mathbb{N} \to \{0,1\}^{**} \to \{0,1\}^{*}$. **Top:** Illustration of sharing. PRGs $F_M(K) = f_K^{|M|}(\varepsilon)$ and $F_{\text{R}}(K) = f_K^{\kappa}(0)$ are defined from f. The message K is shared by $\mathbb{S}$ using access structure A and coins L. Hash function H is defined from the random oracle Hash. **Bottom:** Definition of the scheme. It is Priv and Auth secure, in the random-oracle model, when $\mathbb{S}$ is Priv\$-secure and f is a PRF.

correctness of EX as long as one omits no element of $\mathcal{P}(\mathcal{S})$ that could arise in a valid sharing that respects the known information K. As an example, if Share happens to place the same word J in each share of a deal then the definition of K-plausible sets can further include the constraint that all shares have the same J-value. Lines 82–83 look for a *first* explanation $\mathcal{S}_i$ for $\mathcal{S}$ such that the identified set of valid shares $\mathcal{V}$ is equal to $\mathcal{S}_i$. If we fail to find one, we fail at line 84. Otherwise we seek at lines 86–88 a *second* explanation for $\mathcal{S}$. If we find one, we again fail, but now because there are *two* explanations for $\mathcal{S}$. At line 85 we prune the plausible second explanations to only include those that are not a subset of the first.

Theorem 2. *Let $\mathbb{S}$ be any enriched ADSS scheme with full correctness, and let $\mathbb{SS} = \text{EX}[\mathbb{S}]$. Then:*

```
procedure $\mathbb{SS}$.Recover($K, \mathcal{S}$)
80 let $\mathcal{S}_1, \ldots, \mathcal{S}_w \in \mathcal{P}(\mathcal{S})$ include all $K$-plausible sets
81    of shares, arranged so that $\mathcal{S}_i \supseteq \mathcal{S}_j \Rightarrow i \leq j$
82 for $i \leftarrow 1$ to $w$ do
83    if $(M, R, \mathcal{V}) \leftarrow \mathbb{S}$.Recover($K, \mathcal{S}_i$) and $\mathcal{V} = \mathcal{S}_i$ then goto 85
84 return $\bot$
85 $\{\mathcal{S}'_1, \ldots, \mathcal{S}'_u\} \leftarrow \{\mathcal{S}_{i+1}, \ldots, \mathcal{S}_w\} \setminus \mathcal{P}(\mathcal{V})$
86 for $i \leftarrow 1$ to $u$ do
87    if $(M', R', \mathcal{V}') \leftarrow \mathbb{S}$.Recover($K, \mathcal{S}'_i$) and $\mathcal{V}' \not\subseteq \mathcal{V}$ then
88       return $\bot$
89 return $(M, R, \mathcal{V})$
```

Fig. 9. The EX **construction.** The method turns an enriched ADSS scheme $\mathbb{S}$ into an enriched ADSS scheme $\mathbb{SS} = \mathrm{EX}[\mathbb{S}]$ with Errx security. We let $\mathbb{SS}.\mathrm{Acc} = \mathbb{S}.\mathrm{Acc}$ and $\mathbb{SS}.\mathrm{Share} = \mathbb{S}.\mathrm{Share}$. For a set of shares $\mathcal{S}_i \subseteq \mathcal{S}$ to be "plausibly valid" all the shares of $\mathcal{S}_i$ must have the same access structure and tag, and they must have distinct identities in from that access structures.

1. *If* $\mathbb{S}$ *is* Auth*-secure then so is* $\mathbb{SS}$*. Concretely, given adversary* $\mathbb{AA}$ *(for attacking the* Auth *security of* $\mathbb{SS}$*), we construct adversary* $\mathbb{A}$ *with complexity similar to* $\mathbb{AA}$ *(for attacking the* Auth *security of* $\mathbb{S}$*) such that* $\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{SS}}(\mathbb{AA}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A})$.
2. $\mathbb{SS}$ *is perfectly* Errx*-secure. Concretely, for any adversary* $\mathbb{A}$*,* $\mathbf{Adv}^{\mathrm{errx}}_{\mathbb{SS}}(\mathbb{A}) = 0$.

Efficiency of the constructions. We apply AX and then EX to $\mathbb{S}1$ to obtain a threshold scheme $\mathbb{SS}1$, or AX and then EX to $\mathbb{S}2$ to obtain a scheme $\mathbb{SS}2$ for any access structure. These schemes are highly efficient: sharing an m-byte message M will take $O(m)$ time and, more concretely, about the amount of time to symmetrically encrypt and hash M. This assumes a fixed number of shareholders n, a fixed access-structure encoding A, a fixed tag T, and fixed scheme parameters. Concretely, to share M one will need to apply a hash function like SHA-256 to a string that's $|A| + |T| + \kappa$ bits longer than M (likely $\kappa \in \{128, 256\}$); run a blockcipher like AES in counter mode to generate a pad κ bits longer than $|M|$; and run a sharing under $\mathbb{S}1/\mathbb{S}2$. That last part is fast because the message being shared is just the κ-bit string K. For $\mathbb{S}2$ one needs time linear in the number of threshold gates in the circuit described by A. Practical access structures will have no more than a few gates.

Message recovery for $\mathrm{AX} \circ \mathbb{S}1$ or $\mathrm{AX} \circ \mathbb{S}2$ takes about the same time as sharing, but once EX is added the recovery process can be slow: exponential in the number of shares n presented to Recover. In the worst case, the recovery algorithm, given $\mathcal{S} = \{S_1, \ldots, S_n\}$, might inspect as many as 2^n subsets of $\mathcal{S}$. Still, in practical contexts the number n is likely to be so small that 2^n is still small. Beyond this, we have designed EX so that exponential-time recovery can only arise when there are adversarial edits to shares, not just omissions. In the setting where shares are either valid or absent, Recover will run in essentially the same time as Share.

8 Conclusions & Open Problems

Classical secret-sharing envisages an adversary that does no more than erase some users' shares. Its only aim is to learn what it shouldn't know. Real adversaries aren't so restrained. In response, one can reduce expectations or increase guarantees. We've done the latter.

An unresolved technical problem is how to achieve Errx security with efficient worst-case message-recovery time. Many constructions are plausible. For example, one could add n hash values to each of the n shares, a check-value for each share on each share, using these to partition shares into plausible subsets. Or one could add to each share a dealer-generated digital signature. We suspect that techniques like these can work, and can also make for a simpler Recover than that of EX.

We have not implemented our ADSS schemes. We hope to soon see implementations by others, both as a callable library and as an end-user tool. Ultimately, ADSS implementations should conform to a standards document, such as an RFC. In this way, techniques for adept secret-sharing may become as fixed as those for, say, authenticated encryption.

Our formulation of ADSS has shares leak metadata such as the share number and the operative access structure. Definitions and schemes for metadata-concealing ADSS should be possible.

Underlying our work has been a belief that secret sharing has been underutilized. Secret sharing is not just a tool for doing other things; it is also an aim directly tied to a human aspiration. Shamir wrote in 1979 [40] that "Threshold schemes are ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate." Such cooperation is needed now more than ever.

Acknowledgments

A meeting with J. Alex Halderman and Laurent Richard led to this project. A later meeting with staff at the Freedom of the Press Foundation helped clarify our goals, especially authenticity.

Thanks to John Chan, Jake Craige, Fred Jacobson, Romain Ruetschi, and Conor Schaefer for useful feedback. Thanks to the PoPETs referees for their excellent comments and suggestions.

Bellare and Dai were supported by NSF CNS 1717640, Rogaway by NSF CNS 1717542.

References

[1] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev. Message-locked encryption for lock-dependent messages. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 374–391. Springer, Heidelberg, Aug. 2013.

[2] G. Bai, I. Damgård, C. Orlandi, and Y. Xia. Non-interactive verifiable secret sharing for monotone circuits. In D. Pointcheval, A. Nitaj, and T. Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 225–244. Springer, Heidelberg, Apr. 2016.

[3] A. Beimel. Secret-sharing schemes: A survey. In Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, editors, *Coding and Cryptology*, pages 11–46, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Heidelberg, Aug. 2007.

[5] M. Bellare, Z. Brakerski, M. Naor, T. Ristenpart, G. Segev, H. Shacham, and S. Yilek. Hedged public-key encryption: How to protect against bad randomness. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 232–249. Springer, Heidelberg, Dec. 2009.

[6] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th FOCS*, pages 514–523. IEEE Computer Society Press, Oct. 1996.

[7] M. Bellare, W. Dai, and L. Li. The local forking lemma and its application to deterministic encryption. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 607–636. Springer, Heidelberg, Dec. 2019.

[8] M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 360–378. Springer, Heidelberg, Aug. 2008.

[9] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, Oct. 2012.

[10] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 296–312. Springer, Heidelberg, May 2013.

[11] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, Dec. 2000.

[12] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In M. J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini, editors, *SAC 2009*, volume 5867 of *LNCS*, pages 295–312. Springer, Heidelberg, Aug. 2009.

[13] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer, Heidelberg, Dec. 2000.

[14] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[15] M. Bellare and P. Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In P. Ning, S. De Capitani di Vimercati, and P. F. Syverson, editors, *ACM CCS 2007*, pages 172–184. ACM Press, Oct. 2007.

[16] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 27–35. Springer, Heidelberg, Aug. 1990.

[17] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[18] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th FOCS*, pages 383–395. IEEE Computer Society Press, Oct. 1985.

[19] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[20] C. Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, 2007. http://eprint.iacr.org/2007/399.

[21] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, Oct. 1987.

[22] Freedom Voices Network. Forbidden stories, webpage, visited 2019.09.19.

[23] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, Oct. / Nov. 2006. Available as Cryptology ePrint Archive Report 2006/309.

[24] G. Greenwald. *No Place to Hide*. Metropolitan Books, 2014.

[25] X. Guang, J. Lu, and F. Fu. Repairable threshold secret sharing schemes. *CoRR*, abs/1410.7190, 2014.

[26] L. Harding. What are the Panama papers? A guide to history's biggest data leak. *The Guardian*, 04 2016. https://goo.gl/rXUNdj.

[27] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage.

In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Heidelberg, Aug. 1995.

[28] Z. Jafargholi, C. Kamath, K. Klein, I. Komargodski, K. Pietrzak, and D. Wichs. Be adaptive, avoid overcommitting. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 133–163. Springer, Heidelberg, Aug. 2017.

[29] B. Kacsmar, C. Komlo, F. Kerschbaum, and I. Goldberg. Mind the gap: Ceremonies for applied secret sharing. *PoPETs*, 2020(2):497–415, 2020.

[30] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer, Heidelberg, Apr. 2001.

[31] H. Krawczyk. Secret sharing made short. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 136–146. Springer, Heidelberg, Aug. 1994.

[32] T. M. Laing and D. R. Stinson. A survey and refinement of repairable threshold schemes. *J. Mathematical Cryptology*, 12(1):57–81, 2018.

[33] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[34] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, Heidelberg, Aug. 1992.

[35] A. Raghunathan, G. Segev, and S. P. Vadhan. Deterministic public-key encryption for adaptively chosen plaintext distributions. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 93–110. Springer, Heidelberg, May 2013.

[36] L. Richard. A warning to the corrupt: if you kill a journalist, another will take their place. *The Guardian*, April 2016. https://goo.gl/U868Ye.

[37] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, Nov. 2002.

[38] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, Nov. 2001.

[39] C. Schaefer. Meet Sunder, a new way to share secrets, May 2018. webpage, visited 2019-02-09.

[40] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

[41] M. Tompa and H. Woll. How to share a secret with cheaters. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 261–265. Springer, Heidelberg, Aug. 1987.

[42] V. Vinod, A. Narayanan, K. Srinathan, C. P. Rangan, and K. Kim. On the power of computational secret sharing. In T. Johansson and S. Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 162–176. Springer, Heidelberg, Dec. 2003.

A Relations

Our Auth security goal for ADSS may seem similar to the verifiable secret-sharing (VSS) goal to the first formulated by Chor, Goldwasser, Micali, and Awerbuch (CGMA85) [18]. In this section we contrast these goals. Then we contrast Auth with robustness [15, 31] and, finally, with repairability [25, 32].

A.1 Comparison with VSS

VSS arose in the context of multiparty computation (MPC), where each party would, in a first phase, share out its secret, and later, in a second phase, compute on these shares. For the second phase to work, it was important that, at the end of the first phase, honest parties could be sure that, for each dealer, there existed a single value such that, if, at some later stage, an authorized subset of honest parties attempted recovery, they would recover this unique value.

The original VSS method of CGMA85 [18] involved interaction among the shareholders. Feldman was the first to describe a non-interactive scheme for VSS [21]. Each shareholder performs a local verification step, applying a verification algorithm specified by the scheme to its own share and some public quantity broadcast by the dealer. The result informs a party if its share is valid. This local validity was required to ensure the global unique recoverability property.

Thus two points of difference with ADSS that emerge from the above are: (1) the presence, in a VSS setting, of the verification algorithm; and (2) the presence, in the model for VSS, of a broadcast channel. ADSS does not have a verification algorithm and does not anticipate a broadcast model. Formally, for different definitions D of ours like Auth or Errx, questions like "does VSS imply D, or does D imply VSS" do not make formal sense, as the goals involve different syntax and models.

Rather than end the comparison on this unsatisfactory note, we treat VSS in the ADSS context. We extend the syntax and model to include a verification algorithm and a broadcast channel. With a formal definition of VSS in place, we show that a VSS-secure scheme *is* Auth-secure. That is, VSS is effectively a stronger demand than Auth.

However, achieving VSS instead of Auth seems to involve more work: known VSS schemes are substantially less efficient than the Auth-secure schemes we

```
Game $\mathrm{G}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A})$
80 $H \twoheadleftarrow \Omega$; $(A, A', \mathcal{S}, \mathcal{S}') \twoheadleftarrow \mathbb{A}^H$
81 if $(\exists\, S, S' \in \mathcal{S} \cup \mathcal{S}' : S.\mathrm{pub} \neq S'.\mathrm{pub})$ then return false
82 if $(\exists\, S \in \mathcal{S} \cup \mathcal{S}' : \mathbb{VS}.\mathrm{Verify}(S) = 0)$ then return false
83 if $(\mathcal{S}.\mathrm{id} \notin \mathbb{VS}.\mathrm{Acc}(A)$ or $\mathcal{S}'.\mathrm{id} \notin \mathbb{VS}.\mathrm{Acc}(A'))$
84     then return false
85 $Y \leftarrow \mathbb{VS}.\mathrm{Recover}^H(\mathcal{S})$; $Y' \leftarrow \mathbb{VS}.\mathrm{Recover}^H(\mathcal{S}')$
86 if $(Y = \perp$ or $Y' = \perp)$ then return true
87 $(M, \mathcal{V}) \leftarrow Y$; $(M', \mathcal{V}') \leftarrow Y'$
88 return $(M \neq M')$
```

Fig. 10. Game defining VSS security of a broadcast-model VSS scheme $\mathbb{VS} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}, \mathrm{Verify})$

propose. Worse, VSS needs what is, in our context, the untenable assumption of a broadcast channel: for our motivating use cases, this isn't present.

Formalizing VSS. To enable a formal comparison of Auth and VSS we formalize VSS using ADSS-like syntax. We say that a scheme for *verifiable-ADSS* is a four-tuple of deterministic algorithms $\mathbb{VS} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}, \mathrm{Verify})$. We require that $\mathbb{S} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover})$ is an ADSS scheme, called the ADSS scheme induced by $\mathbb{VS}$. The new algorithm

$$\mathrm{Verify}\colon \mathrm{Share} \to \{0, 1\}$$

will tell a shareholder whether or not its share S is valid. Basic correctness for $\mathbb{VS}$ is that of its induced ADSS scheme together with the requirement that $\boldsymbol{S} = \mathrm{Share}^H(A, M, R, T)$ implies that $\mathrm{Verify}^H(\boldsymbol{S}[i]) = 1$ for all i. We may speak of Priv, Auth security of $\mathbb{VS}$, by which we simply mean those of its induced ADSS scheme.

The new requirement is VSS-security. Consider the $\mathrm{G}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A})$ game on the left of Fig. 10. The VSS advantage of $\mathbb{A}$ is defined by

$$\mathbf{Adv}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A}) = \Pr[\mathrm{G}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A})] \ .$$

In the game, the adversary at line 81 returns an access structure description A and two sets $\mathcal{S}, \mathcal{S}'$ of shares. Line 82 enforces the broadcast constraint that the public portions of all shares are the same. Line 83 says that all shares have passed verification. Line 84 says that the parties underlying both sets of shares are authorized. With these constraints, security (the adversary does not win) requires that the two sets of shares recover to a common, non-$\perp$ value.

VSS implies Auth. Let $\mathbb{VS} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}, \mathrm{Verify})$ be a verifiable-ADSS scheme. Define

```
procedure $\mathrm{Recover}'(\mathcal{S})$
if $(\exists\, S \in \mathcal{S} : \mathrm{Verify}(S) = 0)$ then return $\perp$
$Y \leftarrow \mathrm{Recover}(\mathcal{S})$; return $Y$
```

The following says that ADSS scheme $\mathbb{S}' = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}')$ satisfies Auth.

Proposition 2. *Let* $\mathbb{VS} = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}, \mathrm{Verify})$ *be a verifiable-ADSS scheme and let ADSS scheme* $\mathbb{S}' = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}')$ *be defined as above. Given adversary* $\mathbb{A}_{\mathrm{auth}}$ *we build adversary* $\mathbb{A}_{\mathrm{vss}}$*, about as efficient as* $\mathbb{A}_{\mathrm{auth}}$*, such that* $\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{S}'}(\mathbb{A}_{\mathrm{auth}}) \leq \mathbf{Adv}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A}_{\mathrm{vss}})$*.*

Proof. We define $\mathbb{A}_{\mathrm{vss}}$ as follows:

```
Adversary $\mathbb{A}^H_{\mathrm{vss}}$
$(\mathcal{S}, \mathcal{S}') \twoheadleftarrow \mathbb{A}^H_{\mathrm{auth}}$
$Y \leftarrow \mathrm{Recover}'(\mathcal{S})$; $Y' \leftarrow \mathrm{Recover}'(\mathcal{S}')$
if $(Y = \perp$ or $Y' = \perp)$ then return $\perp$
$A \leftarrow \mathcal{S}.\mathrm{as}$; $A' \leftarrow \mathcal{S}'.\mathrm{as}$; $(M, \mathcal{V}) \leftarrow Y$; $(M', \mathcal{V}') \leftarrow Y'$
return $(A, A', \mathcal{S}, \mathcal{S}')$
```

Our syntax demands $\mathrm{Recover}(\mathcal{S})$ return $\perp$ unless the public parts of all $S \in \mathcal{S}$ are the same and likewise for $\mathcal{S}'$. If $\mathbb{A}_{\mathrm{auth}}$ wins in game $\mathrm{G}^{\mathrm{auth}}_{\mathbb{S}'}(\mathbb{A}_{\mathrm{auth}})$ then $Y \neq \perp$ and $Y' \neq \perp$. Since additionally $\mathcal{S} \cap \mathcal{S}' \neq \emptyset$, we get that all shares in $\mathcal{S} \cup \mathcal{S}'$ have the same public part, ensuring that line 82 of game $\mathrm{G}^{\mathrm{vss}}_{\mathbb{VS}}(\mathbb{A}_{\mathrm{vss}})$ does not return false. The definition of $\mathrm{Recover}'$ tells us that line 83 also does not return false. Our syntax demands that if $\mathrm{Recover}(\mathcal{S})$ returns the non-$\perp$ value $(M, \mathcal{V})$ then access structures $S.\mathrm{as} = A$ of all $S \in \mathcal{S}$ are the same and additionally $\mathcal{S}.\mathrm{id} \in \mathrm{Acc}(A)$, and likewise for $\mathcal{S}'$. So line 84 also does not return false. Now if $\mathbb{A}_{\mathrm{auth}}$ wins we have $Y, Y' \neq \perp$ and $M \neq M'$, so $\mathbb{A}_{\mathrm{vss}}$ wins. □

A.2 Comparison with robustness

Robustness was introduced in [31] and formalized in [15]. We start by adapting the notion of the latter to ADSS. We consider the game of Fig. 11 and let $\mathbf{Adv}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A}) = \Pr[\mathrm{G}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A})]$ be the advantage of an adversary $\mathbb{A}$ in this game. The game picks R at random. The adversary runs in two phases. In the first, it returns A, M, T and state information St. The game then creates $\boldsymbol{S} \leftarrow \mathbb{S}.\mathrm{Share}^H(A, M, R, T)$. In its second stage, given St, the adversary can adaptively corrupt parties, one by one, obtaining their shares, as long as the set G of uncorrupted parties remains authorized and the set B of bad (corrupted) parties remains unauthorized. Finally the adversary outputs a set $\mathcal{B}$ of shares for the cor-

```
Game $\mathrm{G}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A})$

90 $H \twoheadleftarrow \Omega$;  $R \twoheadleftarrow \mathbb{S}.\mathsf{Rand}$; $(A, M, T, St) \twoheadleftarrow \mathbb{A}^{H}(\varepsilon)$
91 $\boldsymbol{S} \leftarrow \mathbb{S}.\mathrm{Share}^{H}(A,M,R,T)$; $n \leftarrow |\boldsymbol{S}|$;  $G \leftarrow [1..n]$
92 $\mathcal{B} \leftarrow \mathbb{A}^{H,\textsc{Corrupt}}(St)$
93 if ($\mathcal{B}.\mathrm{id} \neq B$ or $\mathcal{B}.\mathrm{as} \neq A$) then return false
94 $\mathcal{S} \leftarrow \boldsymbol{S}[G] \cup \mathcal{B}$; $Y' \leftarrow \mathbb{S}.\mathrm{Recover}^{H}(\mathcal{S})$
95 if $Y' \neq \perp$ then $(M', \mathcal{V}) \leftarrow Y'$
96 return ($Y' = \perp$ or $M' \neq M$)

97 procedure $\textsc{Corrupt}(i)$
98 if ($G \setminus \{i\} \notin \mathrm{Acc}(A)$) then return $\perp$
99 if ($B \cup \{i\} \in \mathrm{Acc}(A)$) then return $\perp$
9A $B \leftarrow B \cup \{i\}$; $G \leftarrow G \setminus \{i\}$ ; return $\boldsymbol{S}[i]$
```

Fig. 11. Top: Defining robustness of ADSS scheme $\mathbb{S}$. Adapted from the secret-sharing definitions of [15, 31].

rupted parties. We require that, for any identity, there is at most one share in $\mathcal{B}$ with that identity. The game requires that the set of identities across all the shares in $\mathcal{B}$ be precisely the set of corrupted parties, and that all these shares name access structure A. The adversary wins if the message M' returned by $\mathbb{SS}$.Recover, on $\mathcal{S} = \boldsymbol{S}[G] \cup \mathcal{B}$, is different from M, meaning either some other string or $\perp$.

The following says that Errx implies this Rob notion, meaning if an enriched ADSS scheme has the former, then the ADSS scheme it induces (this means the scheme in which Recover no long returns coins and other algorithms are unchanged) automatically has the latter. Errx is thus stronger than Rob. We can also give examples to show it is *strictly* stronger.

Proposition 3. *Let* $\mathbb{S}' = (\mathrm{Acc}, \mathrm{Share}, \mathrm{Recover}')$ *be an enriched ADSS scheme satisfying full correctness. Define* $\mathrm{Recover}(\mathcal{S})$ *to let* $(M, R, \mathcal{V}) \leftarrow \mathrm{Recover}'(\emptyset, \mathcal{S})$ *and return* $(M, \mathcal{V})$*, and now let* $\mathbb{S}$ *be the ADSS scheme* $(\mathrm{Acc}, \mathrm{Recover}, \mathrm{Share})$*. Given adversary* $\mathbb{A}_{\mathrm{rob}}$ *we build adversary* $\mathbb{A}_{\mathrm{errx}}$ *such that* $\mathbf{Adv}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A}_{\mathrm{rob}}) \leq \mathbf{Adv}^{\mathrm{errx}}_{\mathbb{S}'}(\mathbb{A}_{\mathrm{errx}})$*. The running time of* $\mathbb{A}_{\mathrm{errx}}$ *is about that of* $\mathbb{A}_{\mathrm{rob}}$*.*

Proposition 3. We assume line 93 does not return false and define adversary $\mathbb{A}^{H}_{\mathrm{errx}}$ as follows:

```
Adversary $\mathbb{A}^{H}_{\mathrm{errx}}$
$R \twoheadleftarrow \mathbb{S}.\mathsf{Rand}$ ; $(A, M, T, St) \twoheadleftarrow \mathbb{A}^{H}_{\mathrm{rob}}(\varepsilon)$
$\boldsymbol{S} \leftarrow \mathbb{S}.\mathrm{Share}^{H}(A,M,R,T)$; $n \leftarrow |\boldsymbol{S}|$;  $G \leftarrow [1..n]$
$\mathcal{B} \leftarrow \mathbb{A}^{H,\textsc{CorruptSim}}_{\mathrm{rob}}(St)$; $\mathcal{S} \leftarrow \boldsymbol{S}[G] \cup \mathcal{B}$
return $(\emptyset, \mathcal{S})$

procedure $\textsc{CorruptSim}(i)$
    if $(G \setminus \{i\} \notin \mathrm{Acc}(A))$ then return $\perp$
    if $(B \cup \{i\} \in \mathrm{Acc}(A))$ then return $\perp$
    $B \leftarrow B \cup \{i\}$ ; $G \leftarrow G \setminus \{i\}$; return $\boldsymbol{S}[i]$
```

This adversary runs $\mathbb{A}_{\mathrm{rob}}$, simulating the latter's $\textsc{Corrupt}$ oracle. It returns the set of shares that game $\mathrm{G}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A}_{\mathrm{rob}})$ would pass to Recover, with the known information set to $\emptyset$. Let $Y' \leftarrow \mathrm{Recover}'(\mathcal{S})$ and if $Y' \neq \perp$ then parse it as $(M', R', \mathcal{V}') \leftarrow Y'$. Suppose $\mathbb{A}_{\mathrm{rob}}$ wins in game $\mathrm{G}^{\mathrm{rob}}_{\mathbb{S}}(\mathbb{A}_{\mathrm{rob}})$. This means either $Y' = \perp$ or $M' \neq M$. We want to show $\mathbb{A}_{\mathrm{errx}}$ wins in game $\mathrm{G}^{\mathrm{errx}}_{\mathbb{S}'}(\mathbb{A}_{\mathrm{errx}})$. This means we must show that $Y' \neq \mathrm{UniqueExplanation}(\emptyset, \mathcal{S})$. This is true because all members of $\mathrm{Explanations}^{H}(A, \mathcal{S})$ have the form $(A, M, R, \mathcal{V})$ for some $\mathcal{V}$. □

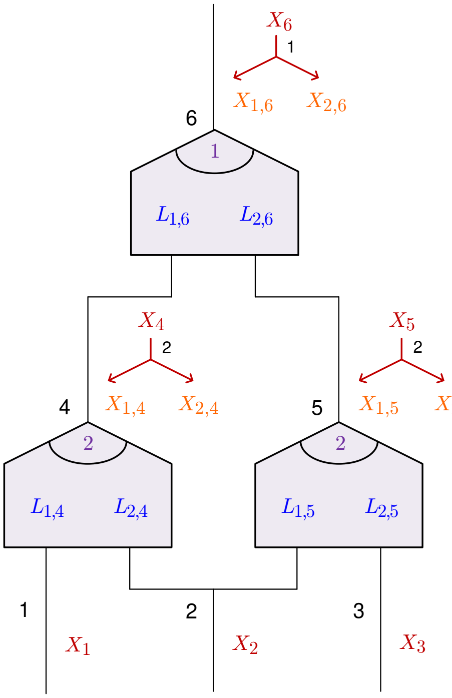
Robustness of an ADSS scheme does *not* imply that it has our Auth property. For example, in a robust 2-of-3 threshold scheme, if a share $\boldsymbol{S}[1]$ for M is combined with shares $\boldsymbol{S}[2]$ and $\boldsymbol{S}[3]$ for another secret M', then all parties, including the first, must recover M', and party 1 is provided no indication that M' is not the secret that was used to create $\boldsymbol{S}[1]$. In the same situation, an auth scheme would return $\perp$, so that party 1 is not given a secret inconsistent with her share. In fact, Auth implies non-robustness, and robustness permits that a party can recover anything that an adversary chooses if it controls the remaining shares.

A.3 Comparison with repairability

Repairable threshold schemes [25, 32] allow a party to reconstruct a missing share by interacting with fellow shareholders. In our applications, we do not anticipate that shareholders have any desire or ability to interact with one another prior to recovery. They might not even know who other shareholders are, or how to reach them. We anticipate that if a party has lost her share, or thinks it may no longer be accurate, she can ask the dealer to regenerate it. If reproducibility was targeted, the dealer can give the party the same share as before. Without reproducibility, the dealer would need to reshare the secret, which means it must contact all other shareholders and get them to replace their shares.

B Base-Level Scheme S2

We describe an alternative to $\mathbb{S}1$ that supports arbitrary access structures instead of just threshold ones. The ac-



Tokens

$X_1 \leftarrow f_R^\lambda\,(1)$

$X_2 \leftarrow f_R^\lambda\,(2)$

$X_3 \leftarrow f_R^\lambda\,(3)$

$X_4 \leftarrow f_R^\lambda\,(4)$

$X_5 \leftarrow f_R^\lambda\,(5)$

$X_6 \leftarrow f_R^\lambda\,(6)$

Encrypting the plaintext

$C \leftarrow M \oplus H^{|M|}(X_6)$

Gate-4 labels

$(X_{1,4}\,,\, X_{2,4}) \leftarrow \text{share}_R^{2,2,4}(X_4)$ // 2-of-2 share X_4 to get $(X_{1,4}\,,\, X_{2,4})$

$L_{1,4} \leftarrow X_{1,4} \oplus H^\lambda(1{,}4,\, X_1)$ // encrypt $X_{1,4}$ using token X_1 as the key

$L_{2,4} \leftarrow X_{2,4} \oplus H^\lambda(2{,}4,\, X_2)$ // encrypt $X_{2,4}$ using token X_2 as the key

Gate-5 labels

$(X_{1,5}\,,\, X_{2,5}) \leftarrow \text{share}_R^{2,2,5}(X_5)$ // 2-of-2 share X_5 to get $(X_{1,5}\,,\, X_{2,5})$

$L_{1,5} \leftarrow X_{1,5} \oplus H^\lambda(1{,}5,\, X_2)$ // encrypt $X_{1,5}$ using token X_2 as the key

$L_{2,5} \leftarrow X_{2,5} \oplus H^\lambda(2{,}5,\, X_3)$ // encrypt $X_{2,5}$ using token X_3 as the key

Gate-6 labels

$(X_{1,6}\,,\, X_{2,6}) \leftarrow \text{share}_R^{1,2,6}(X_6)$ // 1-of-2 share X_6 to get $(X_{1,6}\,,\, X_{2,6})$

$L_{1,6} \leftarrow X_{1,6} \oplus H^\lambda(1{,}6,\, X_4)$ // encrypt $X_{1,6}$ using token X_4 as the key

$L_{2,6} \leftarrow X_{2,6} \oplus H^\lambda(2{,}6,\, X_5)$ // encrypt $X_{2,6}$ using token X_5 as the key

```
procedure $\mathbb{S}2.\text{Share}^H(\mathcal{C}, M, R, T)$
200 $\langle n, q, in, th\rangle \leftarrow \mathcal{C}$
201 for $i \in [1..n+q]$ do $X_i \leftarrow f_R^\lambda(i)$
202 for $g \in [n+1..n+q]$ do
203     $(\iota_1, \ldots, \iota_\eta) \leftarrow in(g);\ \ k \leftarrow th(g)$
204     $(X_{1,g}, \cdots, X_{\eta,g}) \leftarrow \text{share}_R^{k,\eta,g}(X_g)$
205     for $i \in [1..\eta]$ do $L_{i,g} \leftarrow X_{i,g} \oplus H^\lambda(i, g, X_{\iota_i})$
206 $L \leftarrow \langle L_{i,g}\colon\ g \in [n+1..n+q],\ i \in [1..|in(g)|]\rangle$
207 $C \leftarrow H^{|M|}(X_{n+q}) \oplus M;\ \ \widetilde{\mathcal{C}} \leftarrow \langle C, L\rangle$
208 for $i \in [1..n]$ do $S_i \leftarrow \langle i, \mathcal{C}, X_i, \widetilde{\mathcal{C}}, \varepsilon\rangle$
209 return $(S_1, \ldots, S_n)$

procedure $\text{share}_R^{k,\eta,g}(M)$
210 $M_1 \| \cdots \| M_m \leftarrow M$ where $|M_1| = \cdots = |M_m| = \beta$
211 for $(i,j) \in [1..j-1] \times [1..m]$ do $a_{i,j} \leftarrow f_R^\beta(g, i, j)$
212 for $j \leftarrow 1$ to $m$ do $\varphi_j(x) = M_j + \sum_{i=1}^{k-1} a_{i,j} \cdot x^i$
213 for $i \leftarrow 1$ to $\eta$ do $S_i \leftarrow \varphi_1(i) \,\|\, \cdots \,\|\, \varphi_m(i)$
214 return $(S_1, \ldots, S_\eta)$
```

```
procedure $\mathbb{S}2.\text{Recover}^H(\mathcal{S})$
220 $\{S_1, \ldots, S_t\} \leftarrow \mathcal{S};\ \ X_1, X_2, \ldots \leftarrow \bot$
221 for $i \in [1..t]$ do $\langle \iota_i, \mathcal{C}, X_{\iota_i}, \widetilde{\mathcal{C}}, \varepsilon\rangle \leftarrow S_i$
222 $\langle n, q, in, th\rangle \leftarrow \mathcal{C};\ \ \langle C, L\rangle \leftarrow \widetilde{\mathcal{C}}$
223 $\langle L_{i,g}\colon\ g \in [n+1..n+q],\ i \in [1..|in(g)|]\rangle \leftarrow L$
224 for $g \leftarrow n+1$ to $n+q$ do
225     $(\iota_1, \ldots, \iota_\eta) \leftarrow in(g);\ \ k \leftarrow th(g)$
226     for $i \leftarrow 1$ to $\eta$ do $X_{i,g} \leftarrow L_{i,g} \oplus H^\lambda(i, g, X_{\iota_i})$
227     $X_g \leftarrow \text{recover}_R^{k,\eta,g}(X_{1,g}, \ldots, X_{\eta,g})$
228 if $X_{n+q} = \bot$ then return $\bot$
229 return $(H^{|C|}(X_{n+q}) \oplus C,\ \mathcal{S})$

procedure $\text{recover}_R^{k,\eta,g}(S_1, \ldots, S_\eta)$
230 $\mathcal{P} \leftarrow \{(i, S_i)\colon\ i \in [1..\eta],\ S_i \neq \bot\}$
231 if $|P| < k$ then return $\bot$
232 $m \leftarrow$ bytelength (relative to $\beta$) of all 2nd components of $\mathcal{P}$
233 for $j \in [1..m]$ let $\mathcal{P}_j$ be $\mathcal{P}$ with 2nd components having just byte $j$
234 for $j \leftarrow 1$ to $m$ do $\varphi_j(x) \leftarrow \text{Interpolate}_\beta(\mathcal{P}_i)$
236 return $\varphi_1(0) \cdots \varphi_m(0)$
```

Fig. 12. Secret-sharing scheme $\mathbb{S}2$ for achieving classical privacy and accommodating any access structure. On reconstruction, shares are either unchanged or absent. $\mathbb{S}2 = \mathbb{S}2[\beta, f, \lambda]$ depends on $\beta, \lambda, \mu \in \mathbb{N}$ and $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$ satisfying $|f_R^\ell(\cdot)| = \ell$. Hash function $H^\ell(x)$ returns ℓ uniform bits. The access structure is described by a circuit $\mathcal{C} = \langle n, q, in, th\rangle$ of threshold gates each having fewer than 2^β inputs. Sharing depends on random bits $R \in \{0,1\}^\kappa$. No tag T is supported. **Top:** Illustration of sharing with an access structure having AND gates 4 and 5, and OR gate 6. Each wire i is associated with a λ-bit token X_i. The boxed text describes how the dealer computes randomizer U, tokens X_i, ciphertext C, and the $L_{i,j}$ labels. The share for party i has a X_i for its secret part and a public part that includes C, U, and all the $L_{i,j}$ labels. **Bottom:** Definition of the scheme. Arithmetic at line 213 is in the finite field with 2^β points. Procedure Interpolate is as before. Procedures share and recover use arguments and local variables M and S_i distinct from the caller's variables by those names.

cess structure is represented by a circuit of threshold gates, a compact way to describe any access structure. That threshold gates are rich enough to represent any access structure follows from the fact that AND and OR gates are threshold gates, these two gates are already enough to represent any monotone Boolean function, and access structures must be monotone.

We name our scheme $\mathbb{S}2$. It combines the folklore idea of Yao's secret-sharing scheme [3, 28, 42] with Benaloh-and-Leichter's scheme for monotone formulas [16]. The reason we call Yao's scheme "folklore" is because there is no written description of it by him. Rather, he sketched the idea in one or more talks, including one in 1989 [2, p. 228].

The reason for attending to non-threshold access structures is that natural ones *do* arise. They tend to be simple—things like "2 and (1 or 3)", meaning that one requires the participation of shareholder-2 and *either* the participation of shareholder 1 or 3.

Threshold circuits. The top of Fig. 12 depicts a threshold circuit $\mathcal{C} = (n, q, \mathit{in}, \mathit{th})$ with $n = 3$ inputs and $q = 3$ gates. The input wires are numbered 1, 2, 3. The gates, and the wires coming out of them, are numbered 4, 5, 6. Wire 6 is the output wire. The drawing shows connectivity that could be described by a function in from gates to sets of wires where $\mathit{in}(4) = \{1, 2\}$, $\mathit{in}(5) = \{2, 3\}$, and $\mathit{in}(6) = \{4, 5\}$. The threshold value for gates 4 and 5 (written near its apex) is 2, $\mathit{th}(4) = \mathit{th}(5) = 2$, so these are 2-out-of-2 gates, meaning two-input AND gates. The threshold value of gate 6 is 1, $\mathit{th}(6) = 1$, so this is a two-input OR gate. The circuit computes the boolean function $x_1x_2 \vee x_2x_3 = x_2(x_1 \oplus x_3)$ over bits x_1, x_2, x_3 and thereby encodes the access structure $\mathcal{A} = \{\{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$.

Proceeding more formally, we follow a minimalist formalization for garbled circuits [9], saying that a *threshold circuit* (that is, a circuit of threshold gates) is a 4-tuple $\mathcal{C} = (n, q, \mathit{in}, \mathit{th})$. The values $n \geq 2$ and $q \geq 1$ represent the number of input wires and the number of gates, respectively. We number input wires $\text{Inp} = [1..n]$, gates $\text{Gates} = [n + 1..n + q]$, and all wires $\text{Wires} = [1..n + q]$. We identify a gate with the wire coming out of it. The output wire for the entire circuit is wire $n + q$. Function $\mathit{in}\colon \text{Gates} \to \mathcal{P}(\text{Wires})$ identifies the inputs to each gate, $|\mathit{in}(g)| \geq 2$. We will alternatively regard $\mathit{in}(g)$ as a numerically ordered list (e.g., $\mathit{in}(4) = (1, 2)$ rather than $\mathit{in}(4) = \{1, 2\}$). Function $\mathit{th}\colon \text{Gates} \to \mathbb{N}$ is the threshold value of each gate (how many of the inputs must be 1 for the output to be). We require that for all $g \in \text{Gates}$, $\emptyset \neq \mathit{in}(g) \subseteq [1..g - 1]$ (so no cycles) and $1 \leq \mathit{th}(g) \leq |\mathit{in}(g)|$.

For $k \in [1..n]$ and $X \in \{0,1\}^n$ let $\mathrm{TH}_k(X)$ be 1 if X has k or more 1-bits, and 0 otherwise. For $X \in \{0,1\}^n$ and $I \subseteq [1..n]$, let $X[I]$ be the $|I|$-bit substring of X that includes only the bits at positions in I (indexing starting at 1). For a threshold circuit $\mathcal{C} = (n, q, \mathit{in}, \mathit{th})$ and $X \in \{0,1\}^n$, define $\mathcal{C}(X) = \mathsf{Eval}(\mathcal{C}, X)$ by

```
procedure Eval($\mathcal{C}, X$)
$(n, q, \mathit{in}, \mathit{th}) \leftarrow \mathcal{C}$
for $g \leftarrow n + 1$ to $n + q$ do
    $X[g] \leftarrow \mathrm{TH}_{\mathit{th}(g)}\ (X[\mathit{in}(g)])$
return $X[n + q]$
```

For compactness, the code above extends the n-bit string X to $n + q$ bits, using the additional q bits to record the values flowing on non-input wires.

A threshold circuit $\mathcal{C} = (n, q, \mathit{in}, \mathit{th})$ names an n-party access structure $\mathcal{A} = \mathrm{Acc}(\langle\mathcal{C}\rangle)$ that contains $\mathcal{G} \subseteq [1..n]$ exactly when $\mathcal{C}(G) = 1$, where G is the n-bit string with $G[i] = 1$ when $i \in \mathcal{G}$ and $G[i] = 0$ when $i \notin \mathcal{G}$.

Base-level scheme $\mathbb{S}2$. We begin with an informal description, using the example at the top of Fig. 12. Consider the task you face in reconstructing a secret M. You obtain from shares the circuit shown in the figure—everything drawn in black—and the *gate labels* $L_{i,j}$ written in blue. You also extract from each share the strings we call C and U. All these things are public. The secret part of the share S_i from party i is the λ-bit *token* we denote X_i. For input wires, you either *have* the token (if party i provided a share) or you do not, whence one can regard $X_i = \perp$. You now propagate tokens up the circuit, getting tokens or $\perp$-values for each gate, in numerical order. For each gate, if you have the threshold number of tokens for incoming wires then you will be able to propagate your tokens across the gate, getting a token for the output wire of the gate. If you don't have a threshold number of tokens for the gate, then the outgoing token is $\perp$. At line 226, we take the convention that if $X_{\iota_i} = \perp$ for some i, then $X_{i,g} = \perp$. Continuing in this way, you obtain a token for the output wire exactly when you held input tokens for an authorized set of users. If you obtain an output token, you use it decrypt the ciphertext C that accompanies the circuit. The result is the recovered secret.

How do you propagate a threshold number of tokens from the input wires of a gate to its output wire? The gate label for each gate functions as a ciphertext which gets decrypted using a the corresponding token as the key. The decryption also depends on U and the gate number. The plaintext that results is a *share* of the token for the wire coming out of the gate. The recovered shares are combined using polynomial interpolation (Shamir's method) to recover the needed token. That's all there is to it. The top-right of Fig. 12 shows the sharing process, while what we just described the recovery process.

More formally now, scheme $\mathbb{S}2 = \mathbb{S}2[\beta, f, \lambda]$ is parameterized by a block length β (likely 8), a PRF $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$ satisfying $|f_R^\ell(\boldsymbol{x})| = \ell$ for all $\boldsymbol{x}$, and integer λ (the length of gate-labels).The scheme depends on a hash function $H\colon \{0,1\}^\mathbb{N} \times \{0,1\}^{**} \to \{0,1\}^*$, given as an oracle and satisfying $|H(\ell, \boldsymbol{x})| = \ell$. We write $H^\ell(\boldsymbol{x})$ for $H(\ell, \boldsymbol{x})$. The message space for $\mathbb{S}2$ is $\mathbb{SS}.\mathsf{Msg} = \{0,1\}^*$ and the coins are $\mathsf{Rand} = \{0,1\}^\kappa$.

Strings in $\mathbb{S}2.\mathsf{Access}$ encode threshold circuits (n, q, in, th) where $|in(j)| < 2^\beta$ for $j \in [n+1..n+q]$. Each circuit $\mathcal{C}$ encoded by a string in $\mathbb{S}2.\mathsf{Access}$ represents the access structure $\mathbb{S}2.\mathrm{Acc}(\langle \mathcal{C} \rangle)$ as described above. The sharing and recovery algorithms of $\mathbb{S}2$ are given in Fig. 12. It is not hard to check the basic correctness of $\mathbb{S}2[\beta, f, \lambda]$, which we do in Appendix C.3.

We move on to show that $\mathbb{S}2[\beta, f, \lambda]$ satisfies Priv\$ security if f is a secure PRF. In particular, we give the following theorem, which relates the Priv\$ advantage of a given adversary $\mathbb{A}$ to the PRF advantage of a related PRF adversary plus "small terms" given that λ are sufficiently large. The proof of the following is in Appendix C.5.

Theorem 3. *Let $\mathbb{S}2 = \mathbb{S}2[\beta, f, \lambda]$ for valid parameters β, f, λ. Then $\mathbb{S}2$ satisfies* Priv\$*. Concretely, given input-selector $\mathbb{I} \in \mathbb{I}^{\mathrm{priv\$}}$ making q_D* Deal *queries, whose access structures has overall gate count of at most g, and given Priv-adversary $\mathbb{A}$ making q_H queries to H, we construct a PRF adversary $\mathbb{B}$ such that*

$$\mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S}2,\mathbb{I}}(\mathbb{A}) \le \mathbf{Adv}^{\mathrm{prf}}_f(\mathbb{B}) + \frac{g(g-1) + 2q_\mathrm{H}}{2^{\lambda+1}} \quad . \qquad (2)$$

Adversary $\mathbb{B}$ is efficient when $\mathbb{I}$ and $\mathbb{A}$ are.

Use of a random-oracle-modeled hash function in $\mathbb{S}2$ is only for convenience: the encryption at lines 205 and 207 could also have been done with a standard-model tool, like the same PRF f.

```
Game $\mathrm{G}^{\mathrm{prf}}_f(\mathbb{A})$
$b \twoheadleftarrow \{0,1\}$; $q \leftarrow 0$; $b' \twoheadleftarrow \mathbb{A}^{\mathrm{New},\mathrm{Fn}}$; return $(b = b')$

procedure $\mathrm{New}()$
$q \leftarrow q + 1$; $K_q \twoheadleftarrow \mathcal{K}$

procedure $\mathrm{Fn}(i, \ell, \boldsymbol{X})$
if $i \notin [1..q]$ then return $\perp$
if $b = 1$ then return $f^\ell_{K_i}(\boldsymbol{X})$
if $T[i, \ell, \boldsymbol{X}]$ then return $T[i, \ell, \boldsymbol{X}]$
return $T[i, \ell, \boldsymbol{X}] \twoheadleftarrow \{0,1\}^\ell$
```

Fig. 13. Security game capturing the PRF security of $f\colon \{0,1\}^\kappa \times \mathbb{N} \times \mathcal{X} \to \{0,1\}^*$.

C Proofs

C.1 PRF security definition

Before providing any proofs we define PRF-advantage, adapting a multiuser variant from Bellare, Canetti, and Krawczyk [6], which lets the adversary simultaneously attack any number of independently keyed instances, the adaption being that their formalization was for fixed-output-length (FOL) PRFs, while we are using variable-output-length (VOL) PRFs. When that number of instances is at most q, the advantage degrades by a multiplicative factor of q relative to the usual, single instance case. That result is in the reference above for the FOL case, but it is easy to check that it also holds in the VOL case.

Let $f\colon \mathcal{K} \times \mathbb{N} \times \mathcal{X} \to \{0,1\}^*$ be a function. Consider the game $\mathrm{G}^{\mathrm{prf}}_f$ given in Fig. 13. We define the (multi-user) PRF-advantage of adversary $\mathbb{A}$ attacking f as $\mathbf{Adv}^{\mathrm{prf}}_f(\mathbb{A}) = 2\Pr[\mathrm{G}^{\mathrm{prf}}_f(\mathbb{A})] - 1$.

C.2 Equivalence of Errx notions

We claim that the two definitions of Errx given in Section 6 are equivalent. To see this, we fix a hash function $H \in \Omega$, as well as some $K \in \mathsf{Known}$ and $\mathcal{S} \in \mathsf{Shares}$. Let $E_1 = \mathrm{Explanations}^H(K, \mathcal{S})$ be the set returned by the algorithm given in lines 78–7A. Let $E_2 = \mathrm{Explanations}^H(K, \mathcal{S})$ be the set returned by the alternate algorithm. First, suppose $K \in \mathsf{Access}$. Let $(K, M, R, \mathcal{V}) \in E_2$, which means that there exists some $T \in \mathsf{Tag}$, $G \in \mathrm{Acc}(K)$, such that for $\boldsymbol{S} \leftarrow \mathsf{Share}^H(K, M, R, T)$, we have that $\mathcal{V} = \boldsymbol{S}[G]$ and $\mathcal{S} \subseteq \boldsymbol{S}$ (viewing $\boldsymbol{S}$ as a set). Consider $\mathrm{Recover}^H(K, \mathcal{V})$, by full correctness, we will get back $(M, R, \mathcal{V})$. Furthermore,

$\mathcal{V}$.as $= K$ since honestly dealt shares should have the same access structure. This means that $(K, M, R, \mathcal{V}) \in E_1$. On the other hand, let $(\mathcal{V}.\text{as}, M, R, \mathcal{V}) \in E_1$, which means that $(M, R, \mathcal{V}) = \text{Recover}^H(K, \mathcal{S}')$ for some $\mathcal{S}' \subseteq \mathcal{S}$. By the validity requirement, $\mathcal{V}$ is authorized, meaning $\mathcal{V} = \boldsymbol{S}[G]$ for $\boldsymbol{S} = \mathsf{Share}^H(\mathcal{V}.\text{as}, M, R, \mathcal{V}.\text{tag})$, and some $G \in \text{Acc}(K)$. This means that $(\mathcal{V}.\text{as}, M, R, \mathcal{V}) \in E_1$. Second, suppose $K \in \mathsf{Shares}$. Let $(A, M, R, \mathcal{V}) \in E_2$, which means that for some $T \in \mathsf{Tag}$ and $G \in \text{Acc}(A)$, $\mathcal{S} \subseteq \boldsymbol{S} = \mathsf{Share}^H(A, M, R, T)$ and $K \subseteq \mathcal{V} = S[G]$. By full correctness, $\text{Recover}^H(K, \mathcal{V})$ must return $(M, R, \mathcal{V})$. Furthermore, $\mathcal{V}.\text{as} = A$. Hence $(A, M, R, \mathcal{V}) \in E_2$. On the other hand, let $(\mathcal{V}.\text{as}, M, R, \mathcal{V}) \in E_1$, which means that $(M, R, \mathcal{V}) = \text{Recover}^H(K, \mathcal{S}')$ for some $\mathcal{S}' \subseteq \mathcal{S}$. By the validity condition, $\mathcal{V}$ must be an authorized subset of $S = \mathsf{Share}^H(\mathcal{V}.\text{as}, M, R, \mathcal{V}.\text{tag})$, meaning $\mathcal{V} = \boldsymbol{S}[G]$ for some $G \in \text{Acc}(\mathcal{V}.\text{as})$. By requirements of Recover, we know that $K \subseteq \mathcal{V} \subseteq \mathcal{S}'$. Hence $(\mathcal{V}.\text{as}, M, R, \mathcal{V}) \in E_2$. We conclude that $E_1 = E_2$.

C.3 Correctness of the constructions

Basic correctness of $\mathbb{S}1[\beta, f]$. We show $\mathbb{S}1$ satisfies basic correctness. Let $\langle k, n\rangle \in \mathbb{S}1.\mathsf{Access}$, $M \in \mathsf{Msg}$, $R \in \{0,1\}^\kappa$, and $\boldsymbol{S} \leftarrow \mathbb{S}1.\text{Share}(\langle k, n\rangle, M, R, \epsilon)$. Let $G \subseteq \mathcal{P}([1..n])$. If $G \in \text{Acc}(\langle k, n\rangle)$ then $|G| \geq k$, which means that line 113 will not return $\perp$ in $\mathbb{S}1.\text{Recover}(\boldsymbol{S}[G])$ and the recovery will succeed (since 113 is the only place in $\mathbb{S}1$.Recover that can fail for a properly formatted set of shares). On the other hand, if $G \notin \text{Acc}(\langle k, n\rangle)$ then $|G| < k$, which means that line 113 will return $\perp$ in $\mathbb{S}1.\text{Recover}(\boldsymbol{S}[G])$.

Basic correctness of $\mathbb{S}2[\beta, f, \lambda]$. We check that $\mathbb{S}1$ satisfies basic correctness. Let $\mathcal{C} = \langle n, q, in, th\rangle \in \mathbb{S}2.\mathsf{Access}$, $M \in \mathbb{S}2.\mathsf{Messsage}$, $R \in \{0,1\}^\kappa$, $H \in \Omega$, and $\boldsymbol{S} \leftarrow \mathbb{S}2.\text{Share}^H(\mathcal{C}, M, R, \epsilon)$. Let $G \subseteq \mathcal{P}([1..n])$. We claim that if the check of line 21D holds for a gate $g \in [n+1..n+q]$ then the X_g recovered is the same as the X_g sampled in the original sharing; furthermore, if the check of line 21D does not hold for gate g, then $X_g = \perp$. This can be checked by induction on $g \in [n+1..n+q]$ and we omit the details. Let s_G be a string of length n such that $s_G[i] = 1$ if $i \in G$ and 0 otherwise. If $G \in \text{Acc}(\mathcal{C})$ then $\mathcal{C}(s_G) = 1$ by definition. Note that this means line 21D will succeed for $g = n+q$ when we run $\mathbb{S}2.\text{Recover}^H(\boldsymbol{S}[G])$, which means that $X_{n+q} \neq \perp$ and in turn M will be recovered correctly. On the other hand, suppose $G \notin \text{Acc}(\mathcal{C})$, then $\mathcal{C}(s_G) = 0$. Note that this means line 21D will fail for $g = n + q$ in $\mathbb{S}2.\text{Recover}^H\boldsymbol{S}[G]$, resulting in $X_{n+q} = \perp$. Hence $\mathbb{S}2.\text{Recover}^H\boldsymbol{S}[G]$ will return $\perp$ at line 21J.

Full correctness of $\mathbb{SS} = \text{AX}[\mathbb{S}, f]$. We first check that $\mathbb{SS}$ satisfies basic correctness if $\mathbb{S}$ does. Let $A \in \mathbb{SS}.\mathsf{Access}$, $M \in \mathbb{SS}.\mathsf{Messsage}$, $R \in \{0,1\}^\kappa$, $H \in \Omega$, and $\boldsymbol{S} \leftarrow \mathbb{S}2.\text{Share}^H(\mathcal{C}, M, R, \epsilon)$. Let $G \subseteq \mathcal{P}([1..\text{n}(\text{Acc}(A))])$. Consider $\mathbb{SS}.\text{Recover}^H(\boldsymbol{S}[G])$. If $G \in \text{Acc}(A)$, we note that by the basic correctness of $\mathbb{S}$, the correct $(K, \mathcal{G})$ can be recovered at line 316 and in turn the correct M and R. This means that the check at 319 will also succeed and the (A, M) returned by $\mathbb{SS}.\text{Recover}^H(\boldsymbol{S}[G])$ is correct. On the other hand, suppose $G \notin \text{Acc}(A)$. Then by the basic correctness of $\mathbb{S}$, $\mathbb{SS}.\text{Recover}^H(\boldsymbol{S}[G])$ will return $\perp$ at line 313. We move on to check the validity condition. Fix some $H \in \Omega, K$, and $\mathcal{S} \in \mathsf{Shares}$. Suppose $(M, R, \mathcal{V}) \leftarrow \mathbb{SS}.\text{Recover}^H(K, \mathcal{S})$. Then by construction (specifically line 31B), we know that $\mathcal{V} = \mathcal{S}$ indeed came from an honest sharing of M and R.

Full correctness of $\mathbb{SS} = \text{EX}[\mathbb{S}]$. Fix some scheme $\mathbb{S}$, $H \in \Omega$, $A \in \mathsf{Access}$, $M \in \mathbb{SS}.\mathsf{Msg}$, $R \in \mathsf{Rand}$, and $T \in \mathsf{Tag}$. Let $\mathcal{S} = \mathbb{S}.\mathsf{Share}^H(A, M, R, T)$ and $K \in \{A\} \cup \mathcal{P}(\mathcal{S})$. If $G \in \text{Acc}(A)$ then full correctness of $\mathbb{S}$ implies that $\mathbb{S}.\text{Recover}^H(K, \mathcal{S}[G]) = (M, R, \mathcal{S}[G])$, this also means that $\mathbb{SS}.\text{Recover}^H(K, \mathcal{S}[G]) = (M, R, \mathcal{S}[G])$ (line 83terminates with $i = 1$). If $G \notin \text{Acc}(A)$ then by monotonicity, we know that for any $\mathcal{S}' \subseteq \mathcal{S}[G]$, $\mathbb{S}.\text{Recover}^H(K, \mathcal{S}') = \perp$. This means that $\mathbb{SS}.\text{Recover}^H(K, \mathcal{S}[G]) = \perp$ ($\perp$ must be returned at line 84). Validity condition is inherited because both scheme share the same sharing algorithm and that $\mathbb{SS}.\text{Recover}^H(K, \mathcal{S})$ only returns $\mathbb{S}.\text{Recover}(K, \mathcal{S}')$ for some $\mathcal{S}' \subseteq \mathcal{S}$.

C.4 Proof of Proposition 1

Besides the use of the PRF f, scheme $\mathbb{S}1$ is essentially Shamir's scheme [40]. The shares of Shamir's scheme can be perfectly *simulated* as long as no more shares than the threshold is given out. Implementing this intuition, let us consider games G_0 and G_1 given in Fig. 14. Note that G_0 is the same as $\text{G}^{\text{priv}}_{\mathbb{S}1,\mathbb{I}}(\mathbb{A})$. Thus,

$$\mathbf{Adv}^{\text{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}) = 2 \cdot \Pr[\text{G}_0] - 1 \,. \tag{3}$$

Since $\mathbb{S}1$ does not use on any random-oracle-modeled hash function, we omit writing H and giving it to the adversary $\mathbb{A}$. We emphasize that since $\mathbb{I} \in \mathbb{I}^{\text{priv\$}}$, R is sampled uniformly at random for each Deal query regardless of the R value in the input. Coefficients $R_{i,j}$ are derived using f (with a uniformly random and independent seeds, R) in G_0 but are randomly sampled in

```
Game $\mathrm{G}_0, \mathrm{G}_1$ and $\mathrm{G}_2$
$c \twoheadleftarrow \{0,1\}$;  $\alpha \leftarrow 0$
$(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{I}^{\textsc{Deal}}$
if ( $\exists j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then return false
$c' \leftarrow \mathbb{A}(St, \boldsymbol{S}_1[\boldsymbol{B}[1]], \ldots \boldsymbol{S}_\alpha[\boldsymbol{B}[\alpha]], \boldsymbol{P})$
return $(c = c')$

procedure $\textsc{Deal}(A, \boldsymbol{M}_0, \boldsymbol{M}_1, R, T)$
$\boldsymbol{M} \leftarrow \boldsymbol{M}_c$;  $\alpha \leftarrow \alpha + 1$;  $\langle k, n\rangle \leftarrow A$;  $R \twoheadleftarrow \{0,1\}^\kappa$
$M_1 \| \cdots \| M_m \leftarrow \boldsymbol{M}$ where $|M_1| = \cdots = |M_m| = \beta$
for $(i,j) \in [1..(k-1)] \times [1..m]$ do
  $R_{j,i} \leftarrow f_R^\beta(i,j)$;  $\boxed{\underline{\mathrm{G}_1, \mathrm{G}_2}\text{:}\ R_{j,i} \twoheadleftarrow \{0,1\}^\beta}$
for $i \in [1..n]$ do
  for $j \in [1..m]$ do
    $B_{i,j} \leftarrow M_j + R_{j,1} \cdot i + R_{j,2} \cdot i^2 + \cdots + R_{j,k-1} \cdot i^{k-1}$
    $\boxed{\underline{\mathrm{G}_2}\text{:}\ B_{i,j} \twoheadleftarrow \{0,1\}^\beta}$
  $S_i \leftarrow \langle i, \langle k,n\rangle, B_{i,1} \cdots B_{i,m}, \varepsilon, \varepsilon\rangle$
$\boldsymbol{S}_\alpha \leftarrow (S_1, \ldots, S_n)$;  $\boldsymbol{P}[\alpha] \leftarrow \boldsymbol{S}_\alpha.\mathrm{pub}$;  return
```

Fig. 14. Game G_0, G_1, G_2 used in the proof of Proposition 1.

```
$\mathbb{B}^{\mathrm{New},\mathrm{Fn}}$
$c \twoheadleftarrow \{0,1\}$;  $\alpha \leftarrow 0$
$(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{I}^{\textsc{Deal}}$
if ( $\exists j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then return false
$c' \leftarrow \mathbb{A}(St, \boldsymbol{S}_1[\boldsymbol{B}[1]], \ldots \boldsymbol{S}_\alpha[\boldsymbol{B}[\alpha]], \boldsymbol{P})$
return $(c = c')$

procedure $\textsc{Deal}(A, \boldsymbol{M}_0, \boldsymbol{M}_1, R, T)$
$\boldsymbol{M} \leftarrow \boldsymbol{M}_c$;  $\alpha \leftarrow \alpha + 1$;  New();  $\langle k, n\rangle \leftarrow A$
$M_1 \| \cdots \| M_m \leftarrow \boldsymbol{M}$ where $|M_1| = \cdots = |M_m| = \beta$
for $(i,j) \in [1..(k-1)] \times [1..m]$ do
  $R_{j,i} \leftarrow \mathrm{Fn}(\alpha, \beta, (i,j))$
for $i \in [1..n]$ do
  for $j \in [1..m]$ do
    $B_{i,j} \leftarrow M_j + R_{j,1} \cdot i + R_{j,2} \cdot i^2 + \cdots + R_{j,k-1} \cdot i^{k-1}$
  $S_i \leftarrow \langle i, \langle k,n\rangle, B_{i,1} \cdots B_{i,m}, \varepsilon, \varepsilon\rangle$
$\boldsymbol{S}_\alpha \leftarrow (S_1, \ldots, S_n)$;  $\boldsymbol{P}[\alpha] \leftarrow \boldsymbol{S}_\alpha.\mathrm{pub}$;  return
```

Fig. 15. PRF-adversary $\mathbb{B}$ used in the proof of Proposition 1.

G_1. It is standard to check that

$$\Pr[\mathrm{G}_0] = \Pr[\mathrm{G}_1] + \mathbf{Adv}_f^{\mathrm{prf}}(\mathbb{B}) \ , \tag{4}$$

where $\mathbb{B}$ is the PRF-adversary given in Fig. 15. Consider the game G_2 given in Fig. 14. The values of $B_{i,j}$ are randomly sampled in G_2. We claim that

$$\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}_2] \ . \tag{5}$$

This is because for any random degree k polynomial over $\mathbb{F}$, any $k-1$ distinct points are uniformly distributed.

```
Game $\mathrm{G}_0, \mathrm{G}_1$ and $\mathrm{G}_2$
$c \twoheadleftarrow \{0,1\}$;  $\alpha \leftarrow 0$;  $(St, \boldsymbol{B}) \leftarrow \mathbb{I}^{\textsc{Deal}}$
if ( $\exists j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then return false
$c' \twoheadleftarrow \mathbb{A}^{\mathrm{Hash}}(St, \boldsymbol{S}^{(1)}[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}^{(\alpha)}[\boldsymbol{B}[\alpha]], \boldsymbol{P})$
return ($c' = c$)

procedure $\textsc{Deal}(\mathcal{A}, M_0, M_1, R, T)$
$\alpha \leftarrow \alpha + 1$;  $\boldsymbol{A}[\alpha] \leftarrow \mathcal{A}$;  $M_0^{(\alpha)} \leftarrow M_0$;  $M_1^{(\alpha)} \leftarrow M_1$
$R_\alpha \leftarrow R \twoheadleftarrow \{0,1\}^\kappa$;  $(n_\alpha, q_\alpha, in_\alpha, th_\alpha) \leftarrow \mathcal{A}$
for $i \in [1..n+q]$ do
  $X_i \twoheadleftarrow \{0,1\}^\lambda$;  $\boxed{\underline{\mathrm{G}_0}\text{:}\ X_i \leftarrow f_R^\lambda(i)}$
  If $X_i \in \mathcal{X}$ then
    bad $\leftarrow$ true;  $\boxed{\underline{\mathrm{G}_2}\text{:}\ X_i \twoheadleftarrow \{0,1\}^\lambda - \mathcal{X}}$
  $\mathcal{X} \leftarrow \mathcal{X} \cup \{X_i\}$
for $g \in [n+1..n+q]$ do
  $(\iota_1, \ldots, \iota_\eta) \leftarrow in(g)$;  $k \leftarrow th(g)$
  $(X_{1,g}, \cdots, X_{\eta,g}) \leftarrow \mathrm{share}_R^{k,\eta,g}(X_g)$
  For $i \in [1..\eta]$ do $L_{i,g} \leftarrow X_{i,g} \oplus H^\lambda(i, g, X_{\iota_i})$
$L \leftarrow \langle L_{i,g}\text{: } g \in [n+1..n+q],\ i \in in(g)\rangle$
$C \leftarrow H^{|M|}(X_{n+q}) \oplus M_c$;  $\widetilde{\mathcal{C}} \leftarrow \langle L, C\rangle$
for $i \in [1..n]$ do $S_i \leftarrow \langle i, \mathcal{A}, X_i, \widetilde{\mathcal{C}}, \varepsilon\rangle$
$\boldsymbol{S}^{(\alpha)} \leftarrow (S_1, \ldots, S_n)$ ;  $\boldsymbol{P}[\alpha] \leftarrow \boldsymbol{S}^{(\alpha)}.\mathrm{pub}$;  return

procedure $\mathrm{Hash}^\ell(x)$
if not $T[x,\ell]$ then $T[x,\ell] \twoheadleftarrow \{0,1\}^\ell$
return $T[x,\ell]$
```

Fig. 16. Games G_0, G_1, G_2 used in the proof of Theorem 3. Algorithm share is defined in Fig. 12.

Note the both games return false if the adversary attempts to obtain more shares (points) than the threshold k. Finally, since the information given to $\mathbb{A}$ does not depend on the bit c in game G_2, we have that

$$\Pr[\mathrm{G}_2] = \frac{1}{2} \ . \tag{6}$$

Putting (3), (4), (5) and (6) together concludes the proof.

C.5 Proof of Theorem 3

Equation (2) holds trivially if $g \geq 2^\lambda$, so we restrict to the case where $g < 2^\lambda$. Let $\mathbb{A}$ be an adversary and consider the game $\mathrm{G}_{\mathbb{S}2,\mathbb{I}}^{\mathrm{priv}}(\mathbb{A})$, where $\mathbb{I} \in \mathbb{I}^{\mathrm{priv}\$}$. Recall that this is the class of input-selectors that select coins R independently and uniformly at random for each $\textsc{Deal}$ query. Consider the games G_0, G_1 and G_2 given in Fig. 16. In contrast to $\mathrm{G}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}^{\mathrm{priv}}(\mathbb{A})$, we lazily sample H for each query via the procedure HASH given in game G_0 (which is the same one used for G_1 and G_2). Game

```
Game H_0, H_1 and H_2
$c \twoheadleftarrow \{0,1\};\ \alpha \leftarrow 0;\ (St, \boldsymbol{B}) \leftarrow \mathbb{I}^{\textsc{Deal}}$
for $j \leftarrow 1, \ldots, \alpha$ do
  for $\iota \in [1..n_j]$ do $\mathbf{s}_j[\iota] \leftarrow (\iota \in \boldsymbol{B}[j])$
  for $g \leftarrow n_j + 1$ to $n_j + q_j$ do
    $\mathbf{s}_j[g] \leftarrow \mathrm{TH}_{th^{(j)}(g),|in^{(j)}(g)|}(\mathbf{s}_j[in^{(j)}[g]])$
    $(\iota_1, \ldots, \iota_\eta) \leftarrow in^{(j)}(g);\ k \leftarrow th^{(j)}(g)$
    $(X^{(j)}_{1,g}, \cdots, X^{(j)}_{\eta,g}) \leftarrow \mathrm{share}^{k,\eta,g}_{R^{(j)}}(X^{(j)}_g)$
    For $i \in [1..\eta]$ do
      $\boxed{\underline{\mathbf{H}_2}\text{: if } \mathbf{s}_j[g] = 0 \text{ then } X^{(\alpha)}_{i,g} \twoheadleftarrow \{0,1\}^\lambda}$
      $L_{i,g} \leftarrow X_{i,g} \oplus H^\lambda(i, g, X_{\iota_i})$
if ( $\exists\, \alpha : \mathbf{s}_\alpha[n^{(\alpha)} + q^{(\alpha)}] = 1$ ) then return false
$c' \twoheadleftarrow \mathbb{A}^H(\boldsymbol{S}^{(1)}[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}^{(\alpha)}[\boldsymbol{B}[\alpha]], P^{(1)}, \ldots, P^{(\alpha)})$
return ($c' = c$)

procedure $\textsc{Deal}(\mathcal{A}, M_0, M_1, R)$
$\alpha \leftarrow \alpha + 1;\ \boldsymbol{A}[\alpha] \leftarrow \mathcal{A};\ R^{(\alpha)} \leftarrow R \twoheadleftarrow \{0,1\}^\kappa$
$M^{(\alpha)}_0 \leftarrow M_0;\ M^{(\alpha)}_1 \twoheadleftarrow M_1$
$(n^{(\alpha)}, q^{(\alpha)}, in^{(\alpha)}, th^{(\alpha)}) \leftarrow \mathcal{A}$
for $i \in [1..n^{(\alpha)} + q^{(\alpha)}]$ do
  $X^{(\alpha)}_i \leftarrow \{0,1\}^\lambda - \mathcal{X};\ V[X^{(\alpha)}_i] \leftarrow \alpha;\ \mathcal{X} \leftarrow \mathcal{X} \cup \{X\}$
$L^{(\alpha)} \leftarrow \langle L_{i,g} \twoheadleftarrow \{0,1\}^\lambda\colon\ g \in [n^{(\alpha)} + 1..n^{(\alpha)} + q^{(\alpha)}],\ i \in in^{(\alpha)}(g)\rangle$
$C^{(\alpha)} \leftarrow \{0,1\}^{|M|};\ \widetilde{\mathfrak{C}} \leftarrow \langle L^{(\alpha)}, C^{(\alpha)}\rangle$
for $i \in [1..n]$ do $S_i \leftarrow \langle i, \mathcal{A}, X^{(\alpha)}_i, \widetilde{\mathfrak{C}}, \varepsilon\rangle$
$\boldsymbol{S}^{(\alpha)} \leftarrow (S_1, \ldots, S_n)$ ; $P^{(\alpha)} \leftarrow \boldsymbol{S}^{(\alpha)}.\mathrm{pub}$; return

procedure $\mathrm{Hash}^\ell(x)$
$j \leftarrow \perp$; if not $T[x,\ell]$ then $T[x,\ell] \twoheadleftarrow \{0,1\}^\ell$
if $(i, g, X) \leftarrow x$ and $j \leftarrow V[X]$ then
  $Z \leftarrow X^{(j)}_{i,g} \oplus L^{(j)}_{i,g};\ (\iota_1, \ldots, \iota_\eta) \leftarrow in^{(j)}(g)$ ; $\kappa \leftarrow \iota_i$
if $X \leftarrow x$ and $j \leftarrow V[X]$ then $\kappa \leftarrow n^{(j)} + q^{(j)};\ Z \leftarrow M^{(j)}_c \oplus C^{(j)}$
if $(\ell = |Z|)$ and $(X = X^{(j)}_\kappa)$ then
  if $(\mathbf{s}_j[\kappa] = 0)$ then
    bad ← true; $\boxed{\underline{\mathbf{H}_0}\text{: } T[x,\ell] \leftarrow Z}$
  else $T[x,\ell] \leftarrow Z$
return $T[x,\ell]$
```

Fig. 17. Games H_0, H_1, H_2 used in the proof of Theorem 3. Algorithm *Share* is defined in Fig. 12.

```
Adversary $\mathbb{B}$
$c \twoheadleftarrow \{0,1\};\ \alpha \leftarrow 0;\ (St, \boldsymbol{B}) \leftarrow \mathbb{A}^{\textsc{Deal}}$
if ( $\exists\, j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then return false
$c' \twoheadleftarrow \mathbb{A}^{\mathrm{Hash},\textsc{Deal}}(St, \boldsymbol{S}^{(1)}[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}^{(\alpha)}[\boldsymbol{B}[\alpha]], P^{(1)}, \ldots, P^{(\alpha)})$
return ($c' = c$)

procedure $\textsc{Deal}(\mathcal{A}, M_0, M_1, R, T)$
$\mathrm{New}()$ ; $\alpha \leftarrow \alpha + 1$
$\boldsymbol{A}[\alpha] \leftarrow \mathcal{A};\ M^0_\alpha \leftarrow M_0;\ M^1_\alpha \twoheadleftarrow M_1$
$(n_\alpha, q_\alpha, in_\alpha, th_\alpha) \leftarrow \mathcal{A}$
for $i \in [1..n+q]$ do $X_i \leftarrow \mathrm{Fn}(\alpha, \lambda, i)$
for $g \in [n+1..n+q]$ do
  $(\iota_1, \ldots, \iota_\eta) \leftarrow in(g);\ k' \leftarrow th(g)$
  $(X_{1,g}, \cdots, X_{\eta,g}) \leftarrow \mathrm{share}^{k,\eta,g}_R(X_g)$
  For $i \in [1..\eta]$ do $L_{i,g} \leftarrow X_{i,g} \oplus H^\lambda(i, g, X_{\iota_i})$
$L \leftarrow \langle L_{i,g}\colon\ g \in [n+1..n+q],\ i \in in(g)\rangle$
$C \leftarrow H^{|M|}(X_{n+q}) \oplus M_c;\ \widetilde{\mathfrak{C}} \leftarrow \langle L, C\rangle$
for $i \in [1..n]$ do $S_i \leftarrow \langle i, \mathcal{A}, X_i, \widetilde{\mathfrak{C}}, \varepsilon\rangle$
$\boldsymbol{S}^{(\alpha)} \leftarrow (S_1, \ldots, S_n)$ ; $P^{(\alpha)} \leftarrow \boldsymbol{S}^{(\alpha)}.\mathrm{pub}$; return

procedure $\mathrm{Hash}^\ell(x)$
if not $T[x,\ell]$ then $T[x,\ell] \twoheadleftarrow \{0,1\}^\ell$
return $T[x,\ell]$
```

Fig. 18. PRF-adversary $\mathbb{B}$ used in the proof of Theorem 3. Algorithm *Share* is defined in Fig. 12.

G_0 is equivalent to $\mathrm{G}^{\mathrm{priv}}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}(\mathbb{A})$. Hence

$$\Pr[\,\mathrm{G}^{\mathrm{priv}}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}(\mathbb{A})\,] = \Pr[\,\mathrm{G}_0\,]\ . \qquad (7)$$

We claim that

$$\Pr[\,\mathrm{G}_0\,] = \Pr[\,\mathrm{G}_1\,] + \mathbf{Adv}^{\mathrm{prf}}_f(\mathbb{B})\ , \qquad (8)$$

where $\mathbb{B}$ is given on the left column of Fig. 18. This is because games G_0 and G_1 differ only in how X_i is defined, this being PRF-derived in G_0 and uniformly sampled in G_1. (Note that the PRF keys are the R-values sampled by $\mathbb{I}$, which are independently and uniformly random.) It is standard to build the PRF adversary whose PRF advantage bounds the closeness of these two games. We note that G_1 and G_2 are identical-until-bad. By the Fundamental Lemma of Game Playing [14] and the standard birthday argument,

$$\Pr[\,\mathrm{G}_1\,] - \Pr[\,\mathrm{G}_2\,] \le \Pr[\,\mathrm{G}_1 \text{ sets bad}\,] = \frac{g(g-1)}{2^{\lambda+1}}\ . \qquad (9)$$

Next, we shall rewrite the code of $\textsc{Deal}$ so that the labels L do not contain information about bit c. Concretely, consider H_0 given in Fig. 17. The label $L^{(\alpha)}_{i,j}$ are randomly sampled in H_0. Note that game H_0 "programs" the hash function given to $\mathbb{A}$ to behave consistently with the $L^{(\alpha)}_{i,j}$. Furthermore, in game H_0, we compute whether each wire is known to the adversary and store this information inside variable $\mathbf{s}$. Specifically, $\mathbf{s}_\alpha[g] = 1$ if and only if the adversary can compute the label for wire g in the α-th sharing from the set of corrupt shares. Game H_0 is constructed to behave identically to G_2, and

$$\Pr[\,\mathrm{G}_2\,] = \Pr[\,\mathrm{H}_0\,]\ . \qquad (10)$$

Next, let us consider games H_1 given in Fig. 17. Game H_1 no longer programs Hash to return the correct value when the corresponding gate is not corrupt (unknown to the adversary via corrupt shares). Since H_0 and H_1 are identical-until-bad,

$$\Pr[\,\mathrm{H}_0\,] - \Pr[\,\mathrm{H}_1\,] \le \Pr[\,\mathrm{H}_1 \text{ sets bad}\,]\ . \qquad (11)$$

We shall bound $\Pr[\mathrm{H}_1]$ and $\Pr[\mathrm{H}_1 \text{ sets } \mathsf{bad}]$. First, bit c is only used when computing the value Z for the output gates in HASH, and in H_1, HASH is not programmed to output Z when the gate is not corrupt (which must be true for all output gates). Hence H_1 does not leak any information about bit c, and

$$\Pr[\mathrm{H}_2] = \frac{1}{2} \,. \tag{12}$$

Next, consider H_2, which differ from H_1 only in the value of $X_{i,g}^{(\alpha)}$ for $g \in [(n^{(\alpha)}+1)..(n^{(\alpha)}+q^{(\alpha)})]$ such that $\mathbf{s}_\iota[g] = 0$ (gate g is not corrupt). Similar to the proof of Proposition 1, we can substitute the values of $X_{i,g}^{(\alpha)}$ to uniform random ones. So,

$$\Pr[\,\mathrm{H}_1 \text{ sets } \mathsf{bad}\,] = \Pr[\,\mathrm{H}_2 \text{ sets } \mathsf{bad}\,] \,. \tag{13}$$

Also, no information about $X_g^{(\alpha)}$ is given if $\mathbf{s}_\alpha[g] = 0$. Hence for each Hash query there is at most $2^{-\lambda}$ probability of setting bad. Using a union bound over q_{H} queries to Hash, we have

$$\Pr[\mathrm{H}_2 \text{ sets } \mathsf{bad}] \le \frac{q_{\mathrm{H}}}{2^\lambda} \,. \tag{14}$$

Finally, Equation (2) is derived by combining Equations (7–14) and the definition of $\mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}(\mathbb{A})$:

$$\mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}(\mathbb{A}) = 2\Pr[\mathrm{G}^{\mathrm{priv}}_{\mathbb{S}2[\beta,f,\lambda],\mathbb{I}}(\mathbb{A})] - 1 \,.$$

```
Game $\mathrm{G}_x$  //$x \in \{0,1,2,3\}$
$c \twoheadleftarrow \{0,1\};\ i \leftarrow 0;\ (St, \boldsymbol{B}) \leftarrow \mathbb{III}^{\textsc{Deal}}$
if ( $\exists j : \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then return false
$c' \twoheadleftarrow \mathbb{AA}^{H,h}(St, \boldsymbol{S}'_1[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}'_i[\boldsymbol{B}[i]], P_1, \ldots, P_i)$
return $(c = c')$

procedure $\textsc{Deal}(A, M_0, M_1, R, T)$
$i \leftarrow i+1;\ \boldsymbol{A}[i] \leftarrow A$
$J_i \leftarrow \{0,1\}^{2\kappa};\ K_i \leftarrow \{0,1\}^{\kappa};\ L_i \twoheadleftarrow \{0,1\}^{\kappa}$
$X_i \leftarrow \langle A, M_c, R, T \rangle$
$C \leftarrow M \oplus f_K^{|M|}(\varepsilon);\ D \leftarrow R \oplus f_K^{\kappa}(0)$
$\boxed{\underline{\mathrm{G}_3}\textbf{:}\ C \twoheadleftarrow \{0,1\}^{|M|};\ D \twoheadleftarrow \{0,1\}^{\kappa}}$
$Z \leftarrow K_i;$ $\boxed{\underline{\mathrm{G}_2, \mathrm{G}_3}\textbf{:}\ Z \twoheadleftarrow \{0,1\}^{\kappa}}$
$\boldsymbol{S}_i \leftarrow \mathsf{Share}^h(A, Z, L_i, T)$
for $j \leftarrow 1, \ldots, |\boldsymbol{S}|$ do
   $\boldsymbol{S}'_i[j] \leftarrow \langle \boldsymbol{S}_i[j].\mathrm{id}, A, \boldsymbol{S}_i[j].\mathrm{sec}, \langle C, D, J_i, \boldsymbol{S}_i[j].\mathrm{pub} \rangle, T \rangle$
$P_i \leftarrow \boldsymbol{S}'_i.\mathrm{pub}$
return

procedure $H^\ell(X)$
if ($\exists j \le i : X = X_j$ and $\ell = 4\kappa$) then
   bad $\leftarrow$ true; $\boxed{\underline{\mathrm{G}_0}\textbf{:}\ T[X] \leftarrow J_j \| K_j \| L_j}$
if $T_\ell[X] = \perp$ then $T_\ell[X] \twoheadleftarrow \{0,1\}^\ell$
return $T_\ell[X]$

procedure $h^\ell(X)$
if $T'_\ell[X] = \perp$ then $T'_\ell[X] \twoheadleftarrow \{0,1\}^\ell$
return $T'_\ell[X]$
```

Fig. 19. Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$, and G_3 for proof of part 1 of Theorem 1. The boxed code are only executed by the game(s) indicated.

C.6 Proof of Theorem 1

For part 1, consider the games G_0 and G_1 given in Fig. 19. For simplicity, we will consider oracles h and H separately. Our game sequence will modify the code for H while keeping the code for h unchanged. In addition, we will give adversary access to both H and h, instead of Hash. Game G_0 is $\mathrm{G}^{\mathrm{priv}}_{\mathbb{SS},\mathbb{III}}(\mathbb{AA})$ with a lazily sampled H (via procedure Hash) and the sharing algorithm of $\mathbb{SS}$ inlined inside the $\textsc{Deal}$ oracle. Queries to Hash in G_0 are programmed to be consistent with queries to $\textsc{Deal}$ (note that Hash queries happen after all $\textsc{Deal}$ queries are made). The only difference between game G_1 and game G_0 is that Hash is not programmed to be consistent with $\textsc{Deal}$ by omitting the boxed code. By construction, G_0 and G_1 are identical-until-bad. Hence

$$\begin{aligned} \frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\mathrm{priv}}_{\mathbb{SS},\mathbb{III}}(\mathbb{AA}) &= \Pr[\mathrm{G}_0] \\ &= \Pr[\mathrm{G}_1] + (\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]) \\ &\le \Pr[\mathrm{G}_1] + \Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}\,] \,, \end{aligned} \tag{15}$$

where the inequality is by the Fundamental Lemma of Game Playing [14]. We move on to bound $\Pr[\mathrm{G}_1]$ and $\Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}]$. Consider game G_2 and game G_3 given in Fig. 19. Game G_2 differs from G_1 only by the value of Z given to $\mathbb{SS}$.Share. Game G_3 differs from G_2 only in uniform sampling of ciphertext C and D. We build input selector $\mathbb{I}$ (which is in class $\mathbb{I}^{\mathrm{priv\$}}$) as well as Priv\$ adversaries $\mathbb{A}_0$ and $\mathbb{A}_1$ (all given in Fig. 20) such that

$$\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}_2] + \mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}_0) \tag{16}$$

and

$$\Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}] = \Pr[\mathrm{G}_2 \text{ sets } \mathsf{bad}] + \mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}_1) \,. \tag{17}$$

To check the above, notice that the only difference between G_1 and G_2 is the input message to Share. Hence, we can utilize the Priv\$ game for the underlying scheme $\mathbb{S}$ to bridge the different between G_1 and G_2. Furthermore, adversary $\mathbb{A}_0$ is built to simulate and return the

```
Adversary $\mathbb{I}^{\textsc{Deal}}$
$i \leftarrow 0$;  $b \twoheadleftarrow \{0,1\}$;  $(St, \boldsymbol{B}) \twoheadleftarrow \mathbb{II}^{\textsc{DealSim}}$
$\mathbf{X} \leftarrow (X_1, \ldots, X_i)$;  $\mathbf{C} \leftarrow (C_1, \ldots, C_i, D_1, \ldots, D_i)$
$St' \leftarrow (St, \mathbf{X}, \mathbf{C}, \mathbf{J}, \boldsymbol{T})$
return $(St', \boldsymbol{B})$

subroutine $\textsc{DealSim}(A, M_0, M_1, R, T)$
$i \leftarrow i + 1$
$\boldsymbol{T}[i] \leftarrow T$;  $\mathbf{J}[i] \twoheadleftarrow \{0,1\}^{\jmath}$
$K_i \twoheadleftarrow \{0,1\}^{\kappa}$ ;  $L_i \twoheadleftarrow \{0,1\}^{\kappa}$
$C_i \leftarrow M_b \oplus f_{K_i}^{|M|}(\varepsilon)$;  $D_i \leftarrow R \oplus f_{K_i}^{\kappa}(0)$
$Z \twoheadleftarrow \{0,1\}^{\kappa}$
return $\textsc{Deal}(A, K_i, Z, L_i, T)$
```

```
Adversary $\mathbb{A}_x^h(St', \boldsymbol{S}_1, \ldots, \boldsymbol{S}_i, \ldots)$
//$x \in \{0, 1, \varepsilon\}$
$d \twoheadleftarrow \{0,1\}$;  $(St, \mathbf{X}, \mathbf{C}, \mathbf{J}, \boldsymbol{T}) \leftarrow St'$
for $i \leftarrow 1, \ldots, |\boldsymbol{T}|$ do
  for $j \leftarrow 1, \ldots, |\boldsymbol{S}_i|$ do
    $\boldsymbol{S}'_i[j] \leftarrow \langle \boldsymbol{S}_i[j].\mathrm{id}, \boldsymbol{S}_i[j].\mathrm{sec},$
                 $\langle C_i, D_i, J_i, \boldsymbol{S}_i[j].\mathrm{pub}\rangle, T\rangle$
  $P_i \leftarrow \boldsymbol{S}'_i.\mathrm{pub}$
$b' \twoheadleftarrow \mathbb{AA}^{H,h}(St, \boldsymbol{S}'_1, \ldots, \boldsymbol{S}'_i, P_1, \ldots, P_i)$
$\underline{\mathbb{A}_0}$: return $(b = b')$
$\underline{\mathbb{A}_1}$: return bad
$\underline{\mathbb{A}}$: if $d$ then $(b = b')$ else return bad

subroutine $H^{\ell}(X)$
if ($\exists j \le q_d \;:\; X = X_j$) then bad $\leftarrow$ true
if $T_{\ell}[X] = \perp$ then $T_{\ell}[X] \twoheadleftarrow \{0,1\}^{\ell}$
return $T_{\ell}[X]$
```

Fig. 20. Left: input-selector $\mathbb{I}$. Right: Adversaries $\mathbb{A}_0$, $\mathbb{A}_1$ and $\mathbb{A}$ for proof of part 1 of Theorem 1.

```
Adversary $\mathbb{B}_{\boldsymbol{x}}^{\mathrm{New},\mathrm{Fn}}$   //$x \in \{0, 1, \varepsilon\}$
$d \twoheadleftarrow \{0,1\}$;  $c \twoheadleftarrow \{0,1\}$;  $(St, \boldsymbol{B}) \leftarrow \mathbb{II}^{\textsc{Deal}}$
If ( $\exists j \;:\; \boldsymbol{B}[j] \in \mathrm{Acc}(\boldsymbol{A}[j])$ ) then
  return false
$c' \twoheadleftarrow \mathbb{AA}^{H,h}(St, \boldsymbol{S}'_1[\boldsymbol{B}[1]], \ldots,$
            $\boldsymbol{S}'_i[\boldsymbol{B}[i]], P_1, \ldots, P_i)$
$\underline{\mathbb{B}_0}$: return $(c = c')$
$\underline{\mathbb{B}_1}$: return bad
$\underline{\mathbb{B}}$: if $d$ then $(c = c')$ else return bad

subroutine $\textsc{Deal}(A, M_0, M_1, R, T)$
$\mathrm{New}()$;  $i \leftarrow i + 1$;  $\boldsymbol{A}[i] \leftarrow A$
$J_i \leftarrow \{0,1\}^{\jmath}$;  $K_i \leftarrow \{0,1\}^{\kappa}$;  $L_i \twoheadleftarrow \{0,1\}^{\kappa}$
$X_i \leftarrow \langle A, M_c, R, T\rangle$;  $R_i \leftarrow R$
$C_i \leftarrow M_c \oplus \mathrm{Fn}_i^{|M_c|}(\varepsilon)$
$D_i \leftarrow R \oplus \mathrm{Fn}^{\kappa}(0)$;  $Z \twoheadleftarrow \{0,1\}^{\kappa}$
$\boldsymbol{S}_i \leftarrow \mathbb{S}.\mathrm{Share}^h(A, Z, L_i, T)$
for $j \leftarrow 1, \ldots, |\boldsymbol{S}_i|$ do
  $\boldsymbol{S}'_i[j] \leftarrow \langle \boldsymbol{S}_i[j].\mathrm{id}, A, \boldsymbol{S}_i[j].\mathrm{sec},$
             $\langle C_i, D_i, J_i, \boldsymbol{S}_i[j].\mathrm{pub}\rangle, T\rangle$
$P_i \leftarrow \boldsymbol{S}'_i.\mathrm{pub}$
return

subroutine $H^{\ell}(X)$
if ($\exists j \le q_d \;:\; X = X_j$) then bad $\leftarrow$ true
if $T_{\ell}[X] = \perp$ then $T_{\ell}[X] \twoheadleftarrow \{0,1\}^{\ell}$
return $T_{\ell}[X]$

procedure $h^{\ell}(X)$
if $T'_{\ell}[X] = \perp$ then $T'_{\ell}[X] \twoheadleftarrow \{0,1\}^{\ell}$
return $T'_{\ell}[X]$
```

```
Adversary $\mathbb{P}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{T}, \boldsymbol{L}, St)$
for $i \leftarrow 1, \ldots, |\boldsymbol{A}|$ do
  $A \leftarrow \boldsymbol{A}[i]$;  $T \leftarrow \boldsymbol{T}[i]$;  $J_i \leftarrow \{0,1\}^{\kappa}$
  $C \twoheadleftarrow \{0,1\}^{|M|}$;  $D \twoheadleftarrow \{0,1\}^{\kappa}$
  $Z \twoheadleftarrow \{0,1\}^{\kappa}$
  $\boldsymbol{S}_i \leftarrow \mathsf{Share}^h(A, Z, L_i, T)$
  for $j \leftarrow 1, \ldots, |\boldsymbol{S}_i|$ do
    $\boldsymbol{S}'_i[j] \leftarrow \langle \boldsymbol{S}_i[j].\mathrm{id}, A, \boldsymbol{S}_i[j].\mathrm{sec},$
          $\langle C, D, J_i, \boldsymbol{S}_i[j].\mathrm{pub}\rangle, T\rangle$
$\mathbb{AA}^{H,h}(St, \boldsymbol{S}_1[\boldsymbol{B}[1]], \ldots, \boldsymbol{S}_{|\boldsymbol{A}|}[\boldsymbol{B}[|\boldsymbol{A}|]],$
          $P_1, \ldots, P_{|\boldsymbol{A}|})$
$p \twoheadleftarrow [q]$
return $(M_p, R_p)$

subroutine $H^{\ell}(X)$
$q \leftarrow q + 1$;  $\langle A, M_q, R_q, T\rangle \leftarrow X$
if not $T_{\ell}[X]$ then $T_{\ell}[X] \twoheadleftarrow \{0,1\}^{\ell}$
return $T_{\ell}[X]$

procedure $h^{\ell}(X)$
if $T'_{\ell}[X] = \perp$ then $T'_{\ell}[X] \twoheadleftarrow \{0,1\}^{\ell}$
return $T'_{\ell}[X]$
```

Fig. 21. Adversaries for part 1 of Theorem 1. **Left:** adversaries $\mathbb{B}_0, \mathbb{B}_1$, and $\mathbb{B}$. **Right:** Predictor $\mathbb{P}$.

return value of either game G_1 or game G_2, while $\mathbb{A}_1$ is built to simulate and return the flag $\textsf{bad}$. This justifies (16) and (17). Since $\mathbb{A}$ behaves as $\mathbb{A}_d$ based on a randomly chosen bit d,

$$2 \cdot \mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}) = \mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}_0) + \mathbf{Adv}^{\mathrm{priv\$}}_{\mathbb{S},\mathbb{I}}(\mathbb{A}_1)\ . \quad (18)$$

We proceed to bound $\Pr[\mathrm{G}_2]$ and $\Pr[\mathrm{G}_2 \text{ sets } \textsf{bad}]$. We build PRF adversaries $\mathbb{B}_0$ and $\mathbb{B}_1$ (given in the left panel of Fig. 21) such that

$$\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}_3] + \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B}_0) \quad (19)$$

and

$$\Pr[\mathrm{G}_2 \text{ sets } \textsf{bad}] = \Pr[\mathrm{G}_3 \text{ sets } \textsf{bad}] + \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B}_1)\ . \quad (20)$$

The above is true because the only different between game G_2 and game G_3 is how values of C and D are derived (game G_2 uses f while game G_3 samples them uniformly at random). Since $\mathbb{B}$ behaves as $\mathbb{B}_d$ based on a randomly chosen bit d,

$$2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B}) = \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B}_0) + \mathbf{Adv}^{\mathrm{prf}}_{f}(\mathbb{B}_1)\ . \quad (21)$$

Finally, we claim that

$$\Pr[\mathrm{G}_3] = \frac{1}{2} \quad (22)$$

and

$$\Pr[\mathrm{G}_3 \text{ sets } \textsf{bad}] \le (q_{\mathrm{D}} + q) \cdot \mathbf{pred}(\mathbb{II})\ . \quad (23)$$

Equation (22) is by the fact that no information about bit c is leaked to either the input selector nor the adversary. Equation (23) is justified as follows. Consider $\mathbb{P}$ given in the right panel of Fig. 21, which makes at most $q_{\mathrm{D}} + q$ queries to Hash. Predictor $\mathbb{P}$ randomly selects and (M, R) from one of the Hash queries to return as the its guess. If G_3 sets $\textsf{bad}$, then it must be that some query X to Hash matches some X_j during the execution of $\mathbb{AA}$. We lastly need to check that the inputs and oracle for $\mathbb{AA}$ is simulated correctly for $\mathbb{AA}$ by $\mathbb{P}$—this is possible because in G_3, variables C, D and Z are all uniformly random and can be simulated by $\mathbb{P}$. This justifies Equation (23). Finally, Equation (1) is obtained by combining Equations (15–23).

For part 2, consider the game $\mathrm{G}^{\mathrm{auth}}_{\mathbb{SS}}(\mathbb{A})$, modified as per our Section 6 discussion on adjusting Auth, to allow for Recover also returning coins. Let $(M, R, \mathcal{V})$ and $(M', R', \mathcal{V}')$ be the variables defined on line $52'$ and $53'$. If $\mathcal{V} \cap \mathcal{V}' \neq \bot$ and $(M, R) \neq (M', R')$ it must be that there is a collision among the J-values in the two different sharings—that is, a collision on the first 2κ-bits in the output of H. This is because line 31B ensures that the J_i values are the same across all shares. Overall, there are at most $q_{\mathrm{H}} + 2$ queries to H (since Recover calls H exactly once). Hence the game outputs true with probability at most $(q_{\mathrm{H}} + 1)(q_{\mathrm{H}} + 2)2^{-2\kappa}$.

C.7 Proof of Theorem 2

For part 1, consider the adversary $\mathbb{A}$ (constructed from $\mathbb{AA}$) and games $\mathrm{G}_0, \mathrm{G}_1$ given in Fig. 22. Adversary $\mathbb{A}$ runs $\mathbb{AA}$ to obtain inputs that it forwards to $\mathbb{SS}$.Recover, and returns only the valid sets of shares $\mathcal{V}_{\mathbb{SS}}, \mathcal{V}'_{\mathbb{SS}}$ returned by $\mathbb{SS}$.Recover. By construction $\mathrm{G}_0 = \mathrm{G}^{\mathrm{auth}}_{\mathbb{SS}}(\mathbb{AA})$ and $\mathrm{G}_1 = \mathrm{G}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A})$. By construction of EX (in particular line 83 in Fig. 9) and the full correctness of $\mathbb{SS}$, we know that if neither of the two runs of $\mathbb{SS}$.Recover returns $\bot$ then it must be that $\mathcal{V}_{\mathbb{SS}} = \mathcal{V}_{\mathbb{S}}$, $\mathcal{V}'_{\mathbb{SS}} = \mathcal{V}'_{\mathbb{S}}$, $M_{\mathbb{SS}} = M_{\mathbb{S}}$, and $M'_{\mathbb{SS}} = M'_{\mathbb{S}}$. Hence

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathbb{SS}}(\mathbb{AA}) = \Pr[\mathrm{G}_0] \le \Pr[\mathrm{G}_1] = \mathbf{Adv}^{\mathrm{auth}}_{\mathbb{S}}(\mathbb{A})\ , \quad (24)$$

which concludes the proof for part 1.

For part 2, fix some $H \in \Omega$. We claim that for any $(K, \mathcal{S})$,

$$\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S}) = \mathrm{UniqueExplanation}^{H}(K, \mathcal{S})\ . \quad (25)$$

Before we show this we first point out some useful facts about $\mathbb{SS}$.Recover and $\mathbb{S}$.Recover. First, if $\mathbb{S}.\mathrm{Recover}^{H}(K, \mathcal{S}) = (M, R, \mathcal{S})$ then $\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S}) = (M, R, \mathcal{S})$ (take $\mathcal{S}_1 = \mathcal{S}$ at the for loop at line 82). Second, if $\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S}) \neq \bot$ then it must be that $\mathbb{S}.\mathrm{Recover}^{H}(K, \mathcal{S}') \neq \bot$ for some $\mathcal{S}' \subseteq \mathcal{S}$. Now to show (25), consider the set $E = \mathrm{Explanations}^{H}(K, \mathcal{S})$ and consider the following cases.

Case 1: $E = \emptyset$. First, we necessarily have

$$\mathrm{UniqueExplanation}^{\mathrm{Hash}}(K, \mathcal{S}) = \bot$$

by the definition of UniqueExplanation. Second, $\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S})$ is also $\bot$ since there does not exists $\mathcal{S}_i \subseteq \mathcal{S}$ that makes $\mathbb{S}.\mathrm{Recover}^{H}(K, \mathcal{S}_i)$ return non-$\bot$.

Case 2a: $E \neq \emptyset$, $\mathrm{UniqueExplanation}^{H}(K, \mathcal{S}) = \bot$. We will show that $\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S}) = \bot$. Let $\mathcal{S}_1, \ldots, \mathcal{S}_w$ be the K-plausible shares defined on line 80 of $\mathbb{SS}.\mathrm{Recover}^{H}(K, \mathcal{S})$. Let $\mathcal{S}_i$ be the first share (smallest i) for which $(M, R, \mathcal{V}) \leftarrow \mathbb{S}.\mathrm{Recover}^{H}(K, \mathcal{S}_i)$ and $\mathcal{S}_i = \mathcal{V}$ is true. Note that $\mathbb{SS}.\mathrm{Recover}(K, \mathcal{S}_i)$ must also return $(M, R, \mathcal{V})$. Hence $\mathcal{S}_i \in \mathrm{Explanations}^{H}(K, \mathcal{S})$. We claim that there must exist some $j > i$ such that $\mathcal{S}_j \in \mathrm{Explanations}^{H}(K, \mathcal{S})$ and $\mathcal{S}_j \not\subseteq \mathcal{S}_i$. This is the by

```
Adversary $\mathbb{A}^H$
    $(K, \mathcal{S}, K', \mathcal{S}') \twoheadleftarrow \mathbb{AA}^H$
    $(M_{\mathbb{SS}}, R, \mathcal{V}_{\mathbb{SS}}) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$
    $(M'_{\mathbb{SS}}, R', \mathcal{V}'_{\mathbb{SS}}) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K', \mathcal{S}')$
    Return $(\mathcal{V}_{\mathbb{SS}}, \mathcal{V}'_{\mathbb{SS}})$
```

```
Game $\mathrm{G}_0$ / $\mathrm{G}_1$
    $H \twoheadleftarrow \Omega$
    $(K, \mathcal{S}, K', \mathcal{S}') \twoheadleftarrow \mathbb{AA}^H$
    $(M_{\mathbb{SS}}, R, \mathcal{V}_{\mathbb{SS}}) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$
    $(M'_{\mathbb{SS}}, R', \mathcal{V}'_{\mathbb{SS}}) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K', \mathcal{S}')$
    $(M_{\mathbb{S}}, \mathcal{V}_{\mathbb{S}}) \leftarrow \mathbb{S}.\mathrm{Recover}^H(\mathcal{V}_{\mathbb{SS}})$
    $(M'_{\mathbb{S}}, \mathcal{V}'_{\mathbb{S}}) \leftarrow \mathbb{S}.\mathrm{Recover}^H(\mathcal{V}'_{\mathbb{SS}})$
    return $\mathcal{V}_{\mathbb{SS}} \cap \mathcal{V}'_{\mathbb{SS}} \neq \emptyset$ and $(M_{\mathbb{SS}}, R) \neq (M'_{\mathbb{SS}}, R')$ // $\mathrm{G}_0$
    return $\mathcal{V}_{\mathbb{S}} \cap \mathcal{V}'_{\mathbb{S}} \neq \emptyset$ and $M_{\mathbb{S}} \neq M'_{\mathbb{S}}$ // $\mathrm{G}_1$
```

Fig. 22. Adversary $\mathbb{A}$ (left) and games G_0 and G_1 (right) used in the proof of Theorem 2.

the fact that $\mathrm{UniqueExplanation}^H(K, \mathcal{S})$ returns $\perp$. To see this, suppose for contradiction that for all $j > i$ such that $\mathcal{S}_j \in \mathrm{Explanations}^H(K, \mathcal{S})$, $\mathcal{S}_j \subseteq \mathcal{S}_i$. Then since the list of K-plausible shares were exhaustive, $\mathcal{S}_i$ must have made the if statement at line 73–75 true, which contradicts the assumption for case 2a. The existence of such $\mathcal{S}_j$ means that for $(M_j, R_j, \mathcal{V}_j) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S}_j)$, $\mathcal{V}_j \not\subseteq \mathcal{V}_i$. Lastly, there exists some $k \geq j$ such that $\mathcal{S}_k = \mathcal{V}_j$. Furthermore, we have that for $(M_k, R_k, \mathcal{V}_k) \leftarrow \mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}_k)$, $\mathcal{V}_k \not\subseteq \mathcal{V}_i$. This means that $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$ returns $\perp$ at line 88.

Case 2b: $E \neq \emptyset$, $\mathrm{UniqueExplanation}^H(K, \mathcal{S}) \neq \perp$. We will show that $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S}) \neq \perp$. Let $\mathcal{S}_0 \subseteq \mathcal{S}$ be a set of shares whose recovery $(M, R, \mathcal{V}) \leftarrow \mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$ makes the if-statement true at line 73 of $\mathrm{UniqueExplanation}^H(K, \mathcal{S})$. Let $\mathcal{S}_1, \ldots, \mathcal{S}_w$ be the K-plausible shares defined on line 80 of $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$. Now consider the sequence of K-plausible shares resulting from $\mathcal{S}_0 \subseteq \mathcal{S}$ (this is a subsequence of $\mathcal{S}_1, \ldots, \mathcal{S}_w$), say $\mathcal{S}'_1, \ldots, \mathcal{S}'_v$. Suppose $\mathcal{S}'_1 = \mathcal{S}_j$ for some $j \geq 1$ (there is a unique j that satisfy this). We claim that for any $0 < i < j$ either $\mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}_i) = \perp$ or for $(\cdot, \cdot, \mathcal{V}_i) \leftarrow \mathcal{S}_i$, $\mathcal{V}_i \neq \mathcal{S}_i$. This means that during the run of $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$, the for-loop at line 83 ends with $i = j$. Seeking a contradiction, suppose $\mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}_i) \neq \perp$ and for $(\cdot, \cdot, \mathcal{V}_i) \leftarrow \mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}_i)$, $\mathcal{V}_i = \mathcal{S}_i$. We know that $\mathcal{S}_i \in \mathrm{Explanations}^H(K, \mathcal{S})$. Hence $\mathcal{S}_i = \mathcal{V}_i \subseteq \mathcal{V}_j$. This means that $j \leq i$, which contradicts the assumption that $i < j$. Lastly, note that $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S})$ only returns $\perp$ if there exists $\mathcal{S}' \in \{\mathcal{S}_1, \ldots, \mathcal{S}_w\} - \mathcal{P}(\mathcal{S}_j)$ such that $\mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}') = (\cdot, \cdot, \mathcal{S}')$ and $\mathcal{S}' \not\subseteq \mathcal{S}_j$ (note that $\mathbb{SS}.\mathrm{Recover}^H(K, \mathcal{S}') = \mathbb{S}.\mathrm{Recover}^H(K, \mathcal{S}')$ here). But this cannot be true since this would mean that $\mathcal{S}' \in \mathrm{Explanations}^H(K, \mathcal{S})$ and it would have made $\mathrm{UniqueExplanation}^H(K, \mathcal{S})$ return $\perp$.