Strong Secret Sharing with Snitching

Jan Bormet[1(✉)], Stefan Dziembowski[2,4], Sebastian Faust[1],
Tomasz Lizurej[3], and Marcin Mielniczuk[2]

[1] Technische Universität Darmstadt, Darmstadt, Germany
{first.last}@tu-darmstadt.de
[2] University of Warsaw, Warsaw, Poland
stefan.dziembowski@crypto.edu.pl, m.mielniczuk@uw.edu.pl
[3] NASK, Warsaw, Poland
tomasz.lizurej@crypto.edu.pl
[4] IDEAS Institute, Warsaw, Poland

Abstract. One of the main shortcomings of classical distributed cryptography is its reliance on a certain fraction of participants remaining honest. Typically, honest parties are assumed to follow the protocol and not leak any information, even if behaving dishonestly would benefit them economically. More realistic models used in blockchain consensus rely on weaker assumptions, namely that no large coalition of corrupt parties exists, although every party can act selfishly. This is feasible since, in a consensus protocol, active misbehavior can be detected and "punished" by other parties. However, "information leakage", where an adversary reveals sensitive information via, e.g., a subliminal channel, is often impossible to detect and, hence, much more challenging to handle.

A recent approach to address this problem was proposed by Dziembowski, Faust, Lizurej, and Mielniczuk (ACM CCS 2024), who introduced a new notion called *secret sharing with snitching*. This primitive guarantees that as long as no large coalition of mutually trusting parties exists, every leakage of the shared secret produces a "snitching proof" indicating that some party participated in the illegal secret reconstruction. This holds in a very strong model, where mutually distrusting parties use an MPC protocol to reconstruct any information about the shared secret. Such a "snitching proof" can be sent to a smart contract (modeled as a "judge") deployed on the blockchain, which punishes the misbehaving party financially.

In this paper, we extend the results from the work of CCS'24 by addressing its two main shortcomings. Firstly, we significantly strengthen the attack model by considering the case when mutually distrusting parties can also rely on a trusted third party (e.g., a smart contract). We call this new primitive *strong* secret sharing with snitching (SSSS). We present an SSSS protocol that is secure in this model. Secondly, unlike in the construction from CCS'24, our protocol does not require the *honest* parties to perform any MPC computations on hash functions. Besides its theoretical interest, this improvement is of practical importance, as it allows the construction of SSSS from any (even very "MPC-unfriendly") hash function.

1 Introduction

Secure protocols are one of the cornerstones of cryptographic research. While analyzing their security, one assumes that a monolithic adversary controls some subset of the parties while the remaining parties are fully honest. But what happens if all parties are malicious and act in their self-interest? Clearly, if a single monolithic adversary controls *all* parties, then we cannot hope for much. Indeed, it is well known that in the monolithic adversary model, a certain fraction of honest parties is necessary to achieve important cryptographic tasks [20, 14, 4]. Hence, to achieve positive results, we must relax on the "single monolithic adversary" assumption and instead consider multiple subadversaries that each corrupt a disjoint subset of the parties. This setting is motivated by recent applications in blockchains, where the consensus protocol is run among parties divided into subsets of colluders that act in their self-interest.

A fundamental challenge in a setting where all parties are malicious but are controlled by different subadversaries is *collusion*. Blockchains such as Bitcoin or Ethereum prevent collusion by requiring parties to

use some expensive resource (money or computing power) to participate in the protocol. When parties collude and, e.g., start to accept fraudulent transactions, then everyone can observe such misbehavior, resulting in financial punishment for the colluding parties.[5] Unfortunately, preventing collusion is significantly more challenging in the case of cryptographic protocols designed to ensure some form of privacy. This is because, in privacy applications, collusion may happen via undetectable communication channels, where the colluding parties avoid leaving traces that would suffice for punishment. For example, consider the case of secret sharing, where a dealer shares a secret s among two shareholders, P_1 and P_2. The dealer may implement a naive punishment mechanism, where evidence of collusion is when one of the shareholders presents both shares. Trivially, colluding shareholders can avoid punishment by running, e.g., secure computation to recover the secret s (or any function of s that does not suffice for punishment).

Recently, Dziembowski et al. introduced the concept of *individual cryptography* [18, 17] and present a new variant of secret sharing called *secret sharing with snitching (SSS)*. In secret sharing with snitching, we consider a tuple $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of *subadversaries*, where each $\mathcal{A}_i$ corrupts some subset of the shareholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$. While the subadversaries may collude and illegally attempt to reconstruct the shared secret, Dziembowski et al. [18] extend the classical secret sharing by a punishment mechanism punish, which enables any subadversary $\mathcal{A}_a$ that participates in collusion to "snitch" on the illegal reconstruction attempt. A "snitching proof" can be submitted to a trusted third party, called a *judge*, who can punish one of the colluders based on the provided evidence. Assuming the existence of a *secret-state* trusted party would obviously trivialize the problem of secret sharing. Dziembowski et al. [18] assume that the judge has *public state* and argue that, in blockchain applications, the judge can be instantiated with a smart contract that automatically enforces the financial punishment in a native currency of the underlying blockchain. This effectively prevents collusion among the subadversaries since any illegal reconstruction attempt faces the risk of punishment.

A key challenge in designing an SSS scheme is to protect against the *MPC attacks*. An MPC attack leverages secure computation to circumvent the punishment mechanism and allows parties to learn non-trivial information on the shared secret without risking punishment. In a nutshell, the construction of [18] relies on the assumption that computing a large number of hashes via an MPC protocol is infeasible. Clearly, computing some (small) number of hashes within MPC can be done efficiently, which is why [18] makes the model more realistic by introducing a special oracle denoted Ω_H^{MPC}. This oracle extends the standard definition of a random oracle by letting subadversaries query the oracle in two different modes: a *fast* mode, which represents individual computations of the hash function and a *slow* mode, which models that subadversaries jointly evaluate the hash function on shared secret inputs. Unfortunately, the construction of [18] has two crucial shortcomings that we describe below.

Asymmetry between the honest parties and the adversary. First, the protocol of [17] is secure only as long as colluders do not have access to a trusted third party. Hence, in a sense, it relies on the asymmetry between the honest parties, who rely on the "judge" (implemented, e.g., as a smart contract), and the adversary, who cannot use such help. To illustrate how this can lead to real-life attacks, assume that the judge is a smart contract on the blockchain. Before the collusion, the subadversaries set up their own judge (to distinguish it from the judge used by the honest party, we will call such a judge an *arbiter*) that serves as an "insurance" that compensates the punished party. Since such insurance set up by the colluders disincentivizes snitching, it eliminates the risk of punishment, thus breaking the security guarantees offered by an SSS scheme.

MPC computation of hashes by the honest parties. Second, the construction of [18] assumes that honest parties access the Ω_H^{MPC} in "slow mode", i.e., in the sharing phase of the SSS scheme of [18], the honest parties must query the random oracle on secret shared inputs. Obviously, a random oracle cannot be evaluated via an MPC protocol, and when replacing Ω_H^{MPC} by an actual MPC protocol, the security analysis relies on a heuristic (which is the case for the construction in [18]).

[5] Such punishment is either baked directly into the protocol or happens indirectly when the underlying currency loses value.

1.1 Our contribution

This work addresses both shortcomings and gives a new construction that offers security in a significantly more realistic security model. We summarize our contribution below.

- We introduce the notion of *strong secret sharing with snitching*, which models the existence of a (public-state) trusted arbiter accessible to the colluders and can compensate the shareholder that the judge punished. This results in a new notion of *strong punishability*, which guarantees that any illegal reconstruction attempt can be punished even in the presence of arbitrary (polynomial time) compensation strategies.
- We give a construction of a *strong* t-out-of-n secret sharing with snitching. Our construction relies on a novel self-punishment mechanism and leverages non-interactive zero-knowledge proofs to ensure that self-punishment is computationally indistinguishable from real punishment. As an additional benefit, our construction does not require honest parties to call the Ω_H^{MPC} in slow mode. Hence, in contrast to [18], our security analysis is in the traditional random oracle model. To achieve this goal, our scheme constructs a new punishment mechanism, avoiding that parties evaluate hash functions in MPC during the sharing phase.
- Finally, we rigorously prove the security of our construction in the slow/fast query model introduced in [17].

At the conceptual level, we believe that modeling the adversarial arbiter incorporates an intriguing new attack vector, which so far has mainly been ignored in the literature. Although our primary application is in blockchain settings, where such an arbiter may be instantiated as a smart contract, we note that it can also be implemented in other scenarios. For example, instead of sending messages to an arbiter implemented as a smart contract, malicious parties could simply post them on a public forum (or broadcast them to each other). Although in this case, the punished shareholder cannot be compensated, the snitcher still can be penalized in other ways, e.g., by losing its "criminal reputation" and being excluded from future malicious collusions. This is possible if a snitcher can be identified uniquely, e.g., in the case when the subadversaries just send all their shares to a single designated subadversary (so only this subadversary can snitch; see Section 1.3 for a description of this strategy in the two-party case). We leave the study of the security of other cryptographic protocols in the presence of adversarial arbiters as a significant open problem for future work.

1.2 Technical overview

We now give a technical outline of our work. To avoid confusion, we will often refer to the secret sharing scheme of [17] as "*weak* secret sharing with snitching". We will also write "(t, n)-secret sharing" to denote the "t-out-of-n secret sharing". We start by describing the construction of (n, n)-weak secret sharing with snitching (SSS) scheme, which does not require the evaluation of hash functions in MPC. We then introduce our new notion of strong punishability that models the attacks that use the arbiter and show how to extend our (n, n)-SSS scheme to an (n, n)-strong secret sharing with snitching (SSSS) scheme. Looking ahead, our transformation achieves strong punishability by allowing parties to punish *themselves* in a way that is indistinguishable from a real punishment from any arbiter's perspective, thus claiming undue compensation. Finally, we show how to use the threshold transformation of [18] to obtain a (t, n)-SSS scheme from many $(2, 2)$-SSS instances of our (n, n)-SSS scheme and again show how to extend the resulting (t, n)-SSS scheme to the strong punishability model.

The weak n-out-of-n scheme. The main idea behind our scheme and the one of [17] is that, to prevent the MPC attacks, the reconstruction of the secret involves a massive computation of hashes. One way of achieving this is to encode the secret in such a way that the decoding requires a massive amount of hashes. If it were completely impossible to compute hashes in MPC, one could simply use the hash function H (modeled as a random oracle [3]) to derive a one-time-pad, that is, encode a value m as $(Y, H(Y) \oplus m)$ for $Y \leftarrow\!\!\$\ \{0,1\}^\kappa$, requiring just a single hash query to decode. However, it *is* certainly possible to evaluate a

hash function in MPC once, so one has to amplify the number of hashes needed to decode. One can extend this idea and require the parties to invert the hash function given a hint about the input. In other words, one would encode m as

$$\mathsf{Enc}_d(m) = (Y, H_0(Y, Z), H_1(Y, Z) \oplus m),$$

where d is some parameter, $Y \leftarrow\!\!\$\ \{0,1\}^\kappa, Z \leftarrow\!\!\$\ \{0,1\}^d$ and the H_b's are two different hash functions. That way, the decoding requires an expected $\approx 2^{d-1}$ hashes to brute-force Z. For concrete security one needs to pick a value of d such that it is still feasible to compute such a number of hashes in the clear but not in MPC, where the input Y is shared among a set of parties. This encoding can be combined with a punishment mechanism that, through knowledge of Y, allows someone to construct a proof of collusion. Assuming that it is infeasible to evaluate H many times in MPC, the colluders need to reconstruct the entire Y at a single colluder for brute-forcing Z. This colluder can then use Y to craft a proof of collusion together with the dealer and snitch on the colluders. The construction of [18] essentially computed this encoding in MPC and used the pair (Y, Z) as the proof of collusion.

In this work, we avoid computing hashes in MPC, so the encodings need to be computed by the dealer. This renders the previous approach infeasible: the dealer, knowing both Y and Z could easily *frame* the shareholders. By "framing" we mean that the dealer could construct proof of collusion without the shareholders actually colluding. To this end, we use the following strategy: first, the secret $X \in \{0,1\}^\kappa$ is secret-shared into $X_1, \ldots, X_\kappa$ to ensure that the reconstruction requires multiple decodings. Each X_j is then encoded as $\mathcal{E}_j := \mathsf{Enc}_d(X_j)$. Then, in an MPC functionality, the parties first select a random subset $\mathcal{U}$ of the indices with $|\mathcal{U}| = \lfloor \kappa/2 \rfloor$. The MPC functionality shares all $\mathcal{E}_j$ with $j \in \mathcal{U}$ as well as the set $\mathcal{U}$ among the parties. In contrast to the scheme of [18], this functionality does not require any hash evaluations in MPC, enabling the instantiation of H with a hash function designed to be maximally MPC-*unfriendly*. Note that the resulting scheme involves two layers of secret sharing, which is visualized in Fig. 1.

The judge is convinced that the shareholders colluded if it is presented a large enough subset $\mathcal{W} \subset \mathcal{U}$. This makes sense since the shareholders only know the encodings $\mathcal{E}_j$ with $j \in \mathcal{U}$ and all the $\mathcal{E}_j$ with $j \notin \mathcal{U}$ have been discarded in the MPC functionality. As the decoding requires a massive hash computation, the parties cannot run all of them in MPC. Therefore, one of them will have to learn a large portion of the values Y sampled by the dealer during the decoding and can recover the actual indices with the dealer's help. The details of this construction are presented in Section 4.

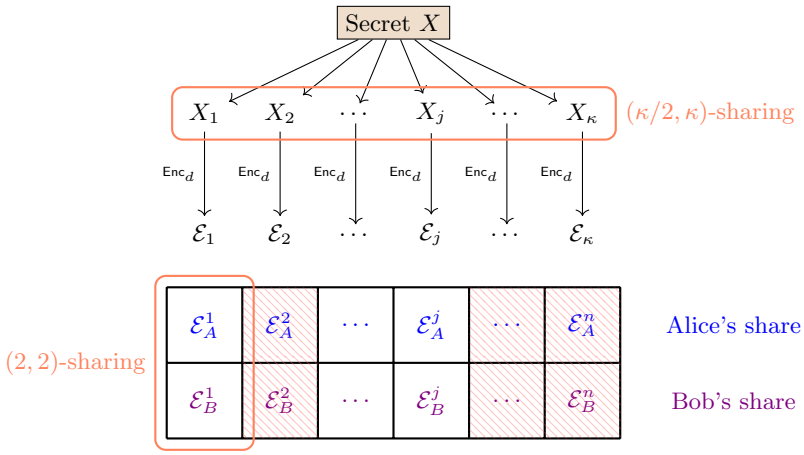


Fig. 1: The schematic representation of the sharing phase of the $(2,2)$-SSS. The crossed-out columns are discarded in the MPC functionality.

The above approach has two main advantages over the construction of [18]: First, in [18], the parties have to engage in an MPC protocol during sharing that involves evaluations of H. This has the downside that

the MPC-hardness of H needs to be balanced because it is "easy enough" to evaluate once in MPC while being "hard enough" to be infeasible to evaluate many times. In our scheme, one can instead design H to be maximally MPC-unfriendly and still efficient to evaluate when the entire input is known. Second, in [18], the punishment information submitted to the judge consists of a preimage of H. In our protocol, the judge does not need to evaluate H at all. Instead, the punishment information is a large enough subset $\mathcal{W}$ of the set $\mathcal{U}$ of indices. This is a significant advantage when trying to achieve strong punishability, as we will see in the following paragraph.

1.3 The strong n-out-of-n scheme

The construction above deters collusion attempts by posing the threat of punishment should any colluding party snitch to the dealer. As mentioned above, in this paper, we additionally consider the attacks where the colluders have access to a trusted arbiter. As in the case of the judge, the arbiter has a public state. It is easy to see that granting the colluders access to a *secret*-state trusted third party would be too strong to permit any meaningful anti-collusion mechanism.[6] As in the case of the judge, in the blockchain settings, an arbiter can be implemented as a smart contract. The funds used for compensation can be deposited by the colluders in advance and claimed back once the protocol ends.

Using such an arbiter, they can get around the threat of punishment as follows: Suppose Alice wants Bob to collude with her to reconstruct the secret illegally, but Bob does not want to risk being punished. Alice can set up a malicious arbiter (think of it as a smart contract) that would provide "insurance" to Bob by guaranteeing to reimburse Bob if he is punished. After Bob has verified that the arbiter is in place, he has nothing to lose by sending his share to Alice. If Alice snitches on Bob, the judge punishes him, but he is, in turn, reimbursed by the arbiter. Overall, if the reimbursement value is equal to the punishment for snitching, no one is incentivized to snitch.

To address this problem, we introduce the notion of *strong punishability* to model secret sharing with snitching in the presence of arbiters, which is a stronger adversarial model. In this model, we add an arbiter algorithm to the adversarial strategy that can choose to compensate parties for their punishments. For a detailed discussion of this new model, we refer the reader to Section 3. Further, we show how to strengthen our weak SSS scheme to the setting of strong punishability in Section 7. Our construction of a *strong* secret sharing with snitching scheme (SSSS) achieves resilience against the arbiter by allowing any party to punish *themselves* and cash in on the reimbursement. Even though we do not model the actual incentives of the parties in this work, it is clear that self-punishment could let the party reap any snitching reward that might be offered, while the compensation by the arbiter has to amount to at least the amount that is slashed from the account of the punished party. Therefore, the arbiter (and hence, the parties who deposited money in it) would only suffer losses. We restrict ourselves to the cryptographic setting and leave a detailed game-theoretic analysis of the underlying incentives for future work.

The key requirement for our idea to work is that self-punishment must be indistinguishable from a real punishment to any outside observer, particularly the arbiter, as otherwise, the arbiter could simply halt once it detects the self-punishment. We achieve this through a non-interactive zero-knowledge proof of knowledge. We first ask each shareholder $\mathsf{P}_i \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ to commit to a public key pk_i. Then, to punish party P_i, the dealer submits a proof of knowledge of a witness to the following statement:

$$\begin{gathered}\text{I know a snitching proof that will be accepted by the judge } \mathsf{J} \\ \textit{or} \\ \text{I know the secret key corresponding to } \mathsf{pk}_i \in \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}.\end{gathered}$$

In the first branch of the OR-statement, the witness is the "original" snitching proof, while in the second branch the witness is the self-snitching party's secret key corresponding to its public key in the PKI. The arbiter cannot distinguish between the two cases due to the zero-knowledge property of the proof. The indistinguishability is crucial; otherwise, the arbiter could attempt to only compensate for real punishments. Note that the first branch of the OR-statement requires performing a zero-knowledge proof over the snitching

[6] This is because this trusted party could simply receive from the parties their shares, and reconstruct the secret.

proof, which would be hard to model if this proof involved evaluations of H, modeled as a random oracle. Luckily, our construction (unlike the one of [17]) avoids this. We refer to Section 7 for the technical details of the strong snitching construction.

We recall that in our weak SSS scheme, the judge does not need to evaluate H, which is in contrast to [18]. This is a crucial advantage as it avoids any practical complications that may arise when trying to construct a proof system for proof of knowledge for a witness to the statement "J accepts" where J involves evaluation of a function H that is hard to compute in MPC and in our proof modeled via the random oracle.[7]

The strong t-out-of-n scheme. Finally, we show in Section 8 how to use the threshold transformation of [18] to obtain a (t, n)-SSS scheme from many $(2, 2)$-SSS instances of our (n, n)-SSS scheme. We then apply the above technique to the resulting (t, n)-SSS scheme to obtain a (t, n)-SSSS scheme.

1.4 Related work

The most closely related works are in the area of *individual cryptography* [17, 18], which studies how to enforce that certain information is entirely known to a party individually, even in the presence of MPC. In particular, [17] first introduces *proofs of individual knowledge* as well as a secret sharing primitive where any reconstruction results in one party individually knowing the entire secret. In [18], this idea is combined with a punishment mechanism to the setting of *secret sharing with snitching*, where at least one party obtains proof of (illegal) reconstruction for any reconstruction of the secret, which can be submitted to a judge to punish colluders. The model in [18] does not consider arbiters among colluders, which is the main focus of our work.

There is a recent line of research on *threshold traitor tracing* [24, 9, 8], which has a similar goal of identifying colluders, called traitors, that contribute with their secret key share to a so-called reconstruction or decoder box. [24] and [9] consider the setting of secret sharing while [8] considers threshold encryption. These works roughly follow the model of *traitor tracing in broadcast encryption* [13], which is significantly different from our model and has been extensively studied for efficient constructions [26, 30, 5, 19, 29, 10, 6, 11, 7, 31, 23]. It considers the setting where a tracer can identify the set of parties that have colluded to construct a decoder box that successfully decrypts ciphertexts. Crucially, this box is modeled as a stateless algorithm that can be run by the tracer. In contrast to our notions, traitor tracing does not model individual parties actively performing illegal reconstruction (respectively decryption), possibly using MPC. By excluding stateful, MPC-empowered adversaries from their model, they permit more efficient and functional constructions.

Our result is related to a line of work on *collusion-free* protocols [1, 27, 33], which deals with the problem of detecting attempts by parties to collude and coordinate their actions during the execution of a protocol and to recent work on preventing collusion in the PIR schemes [22]. These works do not consider MPC attacks.

2 Preliminaries

For $p \in [0, 1]$ and $n \in \mathbb{N}$, let $\mathsf{Bin}(n, p)$ be the binomial distribution, that is, $\Pr[X = k] = \binom{n}{k} \cdot p^k \cdot (1 - p)^{n-k}$ for any $X \sim \mathsf{Bin}(n, p)$. Let $\mathsf{Bernoulli}(p)$ denote the Bernoulli distribution, which is the same as $\mathsf{Bin}(1, p)$. We will use the fact that if $X_1, \ldots, X_n \sim \mathsf{Bernoulli}(p)$ are i.i.d, then $\sum_i X_i \sim \mathsf{Bin}(n, p)$. Let e denote the Euler's constant, and let $\exp(x) := e^x$. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. We will use the following formulation of the Chernoff bound [28, Theorem 4.5].

Lemma 1 (Chernoff bound). *Suppose that the X_i are independent random values taking values in $\{0, 1\}$, let $X = \sum_i X_i$, $\mu = \mathbb{E}[X]$. Then for any $\delta > 0$ it holds that*

$$\Pr[X \geq (1 + \delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{2 + \delta}\right).$$

[7] The fact that the proof system we require as a building block does not need to reason about evaluations of H even further eases the design constraints on H. One can focus on making H as MPC-unfriendly as possible without worrying about the efficiency of the proof system.

2.1 Basic cryptographic building blocks

We assume that all adversaries are probabilistic polynomial time (PPT) machines. A computational security parameter will be denoted as 1^κ. For any $t \leq n$, let $(\mathsf{share}_{t,n}, \mathsf{rec}_{t,n})$ be the t-out-of-n secret sharing scheme of choice, such as Shamir's secret scheme [32]. We assume that on input $X \in \{0,1\}^\kappa$ the algorithm $\mathsf{share}_{t,n}$ outputs a sequence $(X_1, \ldots, X_n)$. Algorithm $\mathsf{rec}_{t,n}$ takes as input a sequence $\{(i, X_i)\}_{i \in \mathcal{S}}$ (where $|\mathcal{S}| \geq t$) and outputs S. In the case $t = n$ we will, for brevity, omit the index i and simply write $\mathsf{rec}_{n,n}(X_1, \ldots, X_n)$. Let $(\mathsf{Encr}, \mathsf{Decr})$ denote an arbitrary semantically secure symmetric-key encryption scheme, where Encr is a probabilistic algorithm and Decr is deterministic. We assume that the keys are sampled uniformly from $\{0,1\}^\kappa$, and hence, we omit the key generation algorithm KGen in this definition.

A *commitment scheme* is a tuple of algorithms $(\mathsf{setup}, \mathsf{commit})$, where (a) setup is a probabilistic algorithm that on input the security parameter 1^κ outputs public parameters *params* and a trapdoor td and (b) commit is a deterministic algorithm that on input the public parameters *params*, some randomness $r \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$ outputs a *commitment* Γ. For brevity, we often omit explicitly mentioning that commit inputs the public parameters *params*.

A commitment scheme satisfies the two properties: "hiding" and "binding". *Hiding* guarantees that for any two messages m_0, m_1 with $|m_0| = |m_1|$ the distribution ensembles $\{C_{0,\kappa}\}_\kappa$ and $\{C_{1,\kappa}\}_\kappa$ are computationally indistinguishable. Here, for bit $b \in \{0,1\}$, the distribution $C_{b,\kappa}$ is defined by the following experiment. Compute $params \leftarrow \mathsf{setup}(1^\kappa)$ and $\Gamma_b := \mathsf{commit}(m_b; r)$ for a uniformly random $r \in \{0,1\}^*$. Output $(params, \Gamma_b)$.

To simplify our definitions and proofs in this paper, we assume that our commitment scheme is *perfectly binding* and extractable, i.e., for every *params* and every Γ, there exists at most one m such that there exists an r such that $\Gamma = \mathsf{commit}(m; r)$ and there exists a PPT algorithm Ext that, given the trapdoor td and a commitment Γ, outputs the message m to which Γ is bound. For an introduction to these notions and standard cryptographic definitions such as "negligible functions" (denoted "$\mathsf{negl}(\kappa)$") and "computational indistinguishability of probability ensembles" (denoted "$\overset{c}{\approx}$") see, e.g., [21, 25].

We rely on non-interactive zero-knowledge proofs of knowledge (NIZKPoK) systems.

Definition 1. *Given an NP relation $\mathcal{R}$ and a corresponding language $\mathcal{L} = \{\chi | (\chi, \omega) \in \mathcal{R}\}$ a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system is a tuple of PPT algorithms $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:*

- Completeness*: For all $(\chi, \omega) \in \mathcal{R}$ it holds that $\mathsf{Verify}(\chi, \mathsf{Prove}(\chi, \omega)) = 1$.*
- Knowledge-soundness*: There exists a PPT extractor Ext such that for all PPT adversaries $\mathcal{A}$ it holds that*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(\chi, \pi) = 1 & (\chi, \pi) \leftarrow\!\$\, \mathcal{A}(1^\kappa) \\ \wedge (\chi, \omega) \notin \mathcal{R} & \omega \leftarrow\!\$\, \mathsf{Ext}^{\mathcal{A}(\cdot)}(\chi, \pi) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

- Zero-knowledge: *There exists a PPT simulator Sim that for all $(\chi, \omega) \in \mathcal{R}$ it holds that $(\mathsf{pp}, \chi, \mathsf{Prove}(\chi, \omega)) \overset{c}{\approx} (\mathsf{pp}, \chi, \mathsf{Sim}(\chi))$.*

where a common $\mathsf{pp} \leftarrow\!\$\, \mathsf{Setup}(1^\kappa)$ is implicit input to all algorithms.

2.2 Secure multiparty computation

Our protocol relies on a non-adaptively secure protocol for evaluating an $n+1$-party functionality, which we now recall.

The model from [12] considers a real world where an ℓ-party protocol Π for evaluating some (possibly probabilistic) function is executed with some adversary $\mathcal{A}$ and an environment $\mathcal{E}$ with input some auxiliary information $z \in \{0,1\}^*$. All parties are PPT and take as input a security parameter. The environment orchestrates the execution of the protocol Π by activating the parties and providing inputs to them. For simplicity, we consider static corruption of up to $t < \ell$ parties and a synchronous communication model. Parties that the adversary corrupts follow her instruction, while honest parties follow the protocol instructions. The protocol execution ends when the environment halts and outputs a bit, which is the output of

the protocol execution. We denote by $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)$ the random variable when the above experiment is executed with uniform randomness and let $\{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$.

In the ideal world, we consider an ideal functionality $\mathcal{I}$ that runs with the honest parties, some ideal-world adversary denoted Sim, and the environment $\mathcal{E}$ that orchestrates the execution. The parties only send inputs to the ideal functionality and receive their outputs, where the order in which this happens is decided by the environment $\mathcal{E}$. Notice that the ideal functionality for distributed protocols gives the ideal adversary Sim additional powers to capture what an adversary $\mathcal{A}$ can always do in the real world. This, in particular, includes aborting the protocol prematurely, often referred to as "early abort". We denote by $\mathsf{IDEAL}_{\mathcal{I},\mathsf{Sim},\mathcal{E}}(\kappa, z)$ the random variable when the above experiment is executed with uniform randomness and let $\{\mathsf{IDEAL}_{\mathcal{I},\mathsf{Sim},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$. Given the ideal and real world, security via the universal composability framework is defined via the following definition.

Definition 2. *Let $\kappa \in \mathbb{N}$. Let $\mathcal{I}$ and Π be as above. We say that Π realizes $\mathcal{I}$ under universal composition, if for every adversary $\mathcal{A}$, there exists an ideal adversary Sim such that for any environment $\mathcal{E}$,*

$$\{\mathsf{IDEAL}_{\mathcal{I},\mathsf{Sim},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}. \tag{1}$$

To prove the universal composability of complex protocols, the main ingredient is the composition theorem. This theorem considers a so-called *hybrid world*. The hybrid world is similar to the real world except that parts of a complex protocol are represented via an ideal functionality $\mathcal{G}$. In the $\mathcal{H}$-hybrid world, parties in the protocol additionally have access to the functionality $\mathcal{G}$. We denote an execution of a protocol Π in the $\mathcal{G}$-hybrid world with an adversary $\mathcal{A}$ and an environment $\mathcal{E}$ as $\mathsf{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)$ and the distribution ensemble as $\{\mathsf{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$. To simplify notation, when $\mathcal{G}$ is the empty functionality, then we are in the real world and write $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}(\kappa, z)$ to denote a real-world execution of the protocol Π. Informally, the composition theorem says that if a protocol Φ realizes a functionality $\mathcal{G}$ under universal composition, then Φ can securely replace the sub-routine $\mathcal{G}$ of a protocol Π. More concretely, we have the following theorem [12].

Theorem 1. *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let Π be an ℓ-party protocol in the $\mathcal{F}$-hybrid world and let Φ be an ℓ-party protocol that securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then for any adversary $\mathcal{A}$ in the $\mathcal{G}$-hybrid model there exists an adversary Sim in the $\mathcal{F}$-hybrid model such that for any environment $\mathcal{E}$ we have:*

$$\{\mathsf{EXEC}^{\mathcal{G}}_{\Pi^{\Phi},\mathcal{A},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathsf{EXEC}^{\mathcal{F}}_{\Pi,\mathcal{S},\mathcal{E}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \tag{2}$$

Notice that in the above, if $\mathcal{G}$ is the empty functionality, then the right side of Eq. (2) becomes the real world $\mathsf{REAL}_{\Pi^{\Phi},\mathcal{A},\mathcal{E}}(\kappa, z)$.

3 Model and definitions

Let $H\colon \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$ be a function modeled via a random oracle [3] denoted as Ω_H.

3.1 Basic settings and terminology.

Our definition of *strong* secret sharing with snitching is inspired by the "weak" one of [18] (see [18, Section 2.1]). The main difference between our definition and the one of [18] is that, instead of requiring "punishability", we require "strong punishability" (see Section 3.2). The other differences with [18] are that we include a setup algorithm, and we do not provide the hash oracle Ω_H^{MPC} to honest parties. This is because, as highlighted in the introduction, unlike in [18], the honest parties do not need to compute the hash function H in MPC. Formally, a *strong secret sharing with snitching (SSSS) with respect to an oracle Ω_H* is a tuple

$$\Pi := (\mathsf{J}, \mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n, \mathsf{setup}),$$

where J, D and the P_i are interactive probabilistic polynomial-time machines with access to Ω_H (machine J is deterministic, while the others are randomized). An SSSS scheme Π comes with two parameters: the

reconstruction threshold $t \in \mathbb{N}$ (with $t \leq n$) and the *slow query budget* $\sigma \in \mathbb{N}$ (the notion of "slow queries" is defined below). We will also say that Π is (t, n)-*SSSS*. We refer to J as the *judge* and to D as the *dealer*. The parties P_i are called *shareholders*.

All parties operate in a synchronous network, are connected by secure channels, and have access to a broadcast channel. The protocol progresses in rounds, where messages sent in round j arrive at the destination in round $j+1$. We restrict the judge's state to be public to allow for implementation as a smart contract on blockchains. As in [18], we consider two types of adversarial models, the *monolithic* and the *distributed* one. Both adversaries (or the "subadversaries", see below) can corrupt parties. Once a (sub)adversary corrupts some party, it takes complete control over it. It learns P_j's entire state and can receive and send messages in P_j's name. A party that is not corrupt is *honest*. The judge J is always honest.

Monolithic adversarial model. The monolithic model is the standard cryptographic model, in which a *monolithic* active probabilistic polynomial-time (PPT) adversary $\mathcal{M}$ can corrupt parties from the set $\{\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n\}$ (the judge J is incorruptible). The adversary can access Ω_H freely. A monolithic adversary is rushing.

Distributed adversarial model. Following the previous works on individual cryptography [17, 18], we consider a *distributed* adversarial model, in which a protocol is attacked by a *distributed* adversary, which is a tuple $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of probabilistic polynomial-time machines called *subadversaries*.

In the execution of a protocol, the subadversaries can communicate freely and actively attack the protocol. Each subadversary can corrupt some shareholders or the dealer (also in this setting J is incorruptible). Let $\mathcal{C}_\ell$ denote the set of parties corrupted by $\mathcal{A}_\ell$. We will call such $\mathcal{C}_\ell$, the *coalition of* $\mathcal{A}_\ell$. We require that all the coalitions are pairwise disjoint, i.e., we consider only distributed adversaries $\mathcal{A}$ such that no P_j is corrupt by more than one subadversary. We also require that $\mathcal{A}$ is t-*bounded* (where the t is the reconstruction threshold, see above), by which we mean that the size of each coalition $\mathcal{C}_i$ is less than t. Sometimes, we abuse notation and write $\mathcal{C}_\mathcal{B}$ where $\mathcal{B}$ is either a monolithic adversary or a subadversary to denote the set of parties corrupted by $\mathcal{B}$.

Querying the random oracle in MPC. Similarly to [18], our model relies on a function $H\colon \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ that can be accessed by all parties and subadversaries through the oracle Ω_H. Such queries are henceforth called *fast queries*. To model resilience against distributed adversaries that can perform MPC attacks, we allow subadversaries to evaluate H over MPC with private inputs. To this end, we provide an oracle Ω_H^{MPC} that can be accessed by a distributed adversary $\mathcal{A}$ in the following way. Each subadversary $\mathcal{A}_\ell$ sends to Ω_H^{MPC} its *private input* $x_\ell \in \{0,1\}^{2\kappa}$. The oracle computes $y := H(x_1 \oplus \cdots \oplus x_a)$ and samples $(y_1, \ldots, y_a) \in (\{0,1\}^\kappa)^a$ such that $y_1 \oplus \cdots \oplus y_a = y$ (in other words $(y_1, \ldots, y_a)$ is the a-out-of-a secret sharing of y).[8] This process counts as a single *slow query*. Our definitions and constructions generally rely on the fact that H is *inefficient to compute in MPC*; hence, the distributed adversary has a limited slow-query budget σ, while the number of fast queries is only restricted by the polynomial running time of the subadversaries. Crucially, and in contrast to [18], our construction does not require honest parties to evaluate H in MPC.

Protocol execution. The SSSS protocol is divided into the following phases.

Setup phase $\Pi.\mathsf{setup}$**:** During this phase, a probabilistic algorithm $\mathsf{setup}(1^\kappa)$ is executed to sample auxiliary private input aux_i for every party P_i as well as some common public input pub. Each party P_i receives $(\mathsf{aux}_i, \mathsf{pub})$.

Sharing phase $\Pi.\mathsf{share}$**:** In this phase, all the parties take as input 1^κ and the common public input pub. The dealer D additionally takes a *secret* $S \in \{0,1\}^*$ as input. The parties P_i receive their auxiliary

[8] Clearly, the assumption that the adversary uses additive a-out-of-a secret sharing can be made without loss of generality. In [18], the ability to use H in an MPC computation is modeled by providing an arbitrary function φ_{in} that is executed on the subadversaries's inputs by the oracle before invoking H. It is easy to see that our model for Ω_H^{MPC} is equivalent as the distributed adversary can compute φ_{in} in MPC with the result being additively secreted-shared, and have each subadversary send their output share to Ω_H^{MPC}.

private input aux_i. At the end of this phase, the judge may output error, in which case, the protocol terminates. Otherwise, the parties proceed to one of the next phases.

Punishment phase $\Pi.\mathsf{punish}$: This phase is parametrized by a set of subadversaries A called *snitchers* and the target P_i who does not belong to A's coalition $\mathcal{C}_A$. The dealer D receives information from the snitchers A and may send a message to the judge J. The judge outputs $(\mathsf{punish}, \mathsf{P}_i)$ for some $i \in [n]$ or $\perp$. Note that the dealer is involved in the punishment phase. This is not a problem in real-life blockchain applications, as it is the dealer who is interested in punishing the shareholders.

Reconstruction phase $\Pi.\mathsf{rec}$: The dealer D does not participate in this phase. At its end, one of the shareholders outputs $\widehat{S} \in \{0,1\}^* \cup \{\perp\}$.

Remark 1 (on the involvement of the dealer in the punishment phase). Unlike in [18], our scheme requires that the dealer be involved in the punishment phase. This is necessary because, in our scheme, the punishment phase is interactive and requires decommitments from all the parties, and so this phase may only run once. To avoid sabotage by adversaries initiating the punishment phase with invalid proofs, we need to either (a) authenticate the sender of the proof and punish them if it is invalid; or (b) only allow the dealer to initiate punishments. However, the former approach would enable arbiters to distinguish self-punishments from real punishments by establishing the identity of the snitcher. We finally remark that this responsibility can be outsourced to any third party without secrecy issues.

3.2 Security properties.

The security properties that we require are "correctness", "unframability", and "strong punishability". The first two of them are very similar to the ones of SSS [18]. We state them below for completeness.

Correctness. The correctness requirement states that if all the parties are honest, then for every input S with overwhelming (in κ) probability, the output $\widehat{S}$ in the reconstruction phase equals the dealer's secret S.

Experiment $\mathsf{Unframability}^{\Pi}_{\mathcal{M}}(1^\kappa)$

1. The experiment runs the setup phase $\Pi.\mathsf{setup}$ and runs $\mathcal{M}(1^\kappa, \mathsf{pub})$ to obtain the set of corrupted parties $\mathcal{C}_{\mathcal{M}}$.
2. The experiment runs $\mathcal{M}(1^\kappa, \{\mathsf{aux}_i\}_{i\in\mathcal{C}_{\mathcal{M}}}, \mathsf{pub})$ and executes the sharing and the punishment phases $\Pi.\mathsf{share}$ and $\Pi.\mathsf{punish}$, where $\mathcal{M}$ controls the dealer and the corrupted parties in $\mathcal{C}_{\mathcal{M}}$ and the experiment controls the remaining honest parties $\mathcal{H}$ and the judge J.
3. If at the end of the punishment phase, the judge J outputs $(\mathsf{punish}, \mathsf{P})$ for some $\mathsf{P} \in \mathcal{H}$, then the experiment outputs 1. Otherwise, it outputs 0.

Fig. 2: The $\mathsf{Unframability}$ experiment

Unframability. We want to prevent situations where shareholders who follow the protocol honestly can get punished by the judge, even if all parties, excluding the judge, behave maliciously. The unframability property guarantees every honest shareholder P, that even if the dealer and all other shareholders maliciously collaborate, they can only *frame* P with negligible probability. The definition of unframability already appeared in [18]. We rephrase it as an experiment.

Definition 3 (Unframability). *A (t,n)-SSSS scheme Π is* unframable *if for every monolithic adversary $\mathcal{M}$, there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Unframability}^{\Pi}_{\mathcal{M}}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa).$$

Strong Punishability. The strong punishability property guarantees that if the shareholders engage in an illegal secret reconstruction protocol, then some shareholders (called snitchers) can collaborate with the dealer D to generate proof for the judge that will convince her to punish some shareholder P_i. The punishment proof is produced by all the subadversaries except the one who corrupted P_i. The snitchers should succeed with a probability of at least the amount of non-trivial information the colluding shareholders obtain about the shared secret as measured in terms of the "distinguishing advantage". Strong punishability requires that this holds even in the presence of attacks, where adversaries set up an arbiter who compensates the malicious parties should they get punished.

Experiment $\mathsf{Sharing}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi, r; \kappa]\!]$

1. The experiment executes the setup phase $\Pi.\mathsf{setup}$.
2. A pre-computation phase is executed. During it, the subadversaries can submit an arbitrary number of slow queries to Ω_H.
3. $\mathcal{A}$ sends two secrets $S_0, S_1 \in \{0,1\}^*$ of the same length.
4. The sharing phase is executed with the security parameters 1^κ and $S = S_r$. During this phase, the distributed adversary $\mathcal{A}$ attacks the protocol Π. Each subadversary $\mathcal{A}_i$ receives $\{\mathsf{aux}_j\}_{P_j \in \mathcal{C}_i}$ and pub as input.
5. The output of the experiment is equal to the output of $\mathcal{A}$.

Experiment $\mathsf{Snitching}[\![S, \mathcal{B} \leftrightarrows (\Pi, \mathsf{Arb}, \mathcal{A}, j, i); \kappa]\!]$

1. The experiment runs the setup phase $\Pi.\mathsf{setup}$.
2. If $j = 0$, then assign to j an arbitrary index such that $\mathsf{P}_i \notin \mathcal{C}_j$.
3. The experiment controls the uncorrupted parties, the judge, and the subadversary $\mathcal{A}_j$. It also executes $\mathcal{B}(1^\kappa, j, i, \{\mathsf{aux}_\ell\}_{P_\ell \notin \mathcal{C}_j}, \mathsf{pub})$, who acts on behalf of all the subadversaries except $\mathcal{A}_j$ as well as the dealer. The experiment forwards all communication between the subadversaries, the honest parties, the dealer and the judge.
4. A pre-computation phase is executed. During it, the subadversaries can submit an arbitrary number of slow queries to Ω_H.
5. The Sharing phase is executed with the security parameter 1^κ. During this phase, $\mathcal{A}$ attacks the protocol Π by first choosing S_0 and S_1 of the same length as S. Next, the S_b are discarded and S is the dealer's input in the sharing procedure instead.
6. The punishment phase is executed. During this phase, $\mathcal{B}$ and $\mathcal{A}_j$ can communicate with the dealer and the judge. The Judge J outputs its punishments, and Arb outputs its compensations.
7. The experiment checks if one of the following conditions is met:
   (a) If some party $\mathsf{P} \in \mathcal{C}_j$ is *punished and not compensated*, i.e., the output of J is $(\mathsf{punish}, \mathsf{P})$ and the output of Arb is not $(\mathsf{comp}, \mathsf{P})$, then output 1.
   (b) If some party $\mathsf{P} \notin \mathcal{C}_j$ is *punished and compensated*, i.e., the output of J is $(\mathsf{punish}, \mathsf{P})$ and the output of Arb is $(\mathsf{comp}, \mathsf{P})$, then output 2.

   Otherwise, the experiment outputs 0.

Fig. 3: The experiments $\mathsf{Sharing}$ and $\mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \mathsf{Arb}, \mathcal{A}, j); \kappa]\!]$ in which the adversary $\mathcal{A}_j$ is being snitched upon. Recall that $\mathcal{C}_j$ denotes the coalition of parties controlled by the adversary $\mathcal{A}_j$.

Arbiter. We allow a *probabilistic* arbitrary arbiter Arb to observe the inputs, outputs, and the state of the judge J. It can *receive messages* from the subadversaries throughout protocol execution, and send messages to subadversaries starting from the punishment phase. However, we restrict the arbiter's ability to *send* messages prior to the punishment to exclude the trivial attack, where colluders simply send their secret

share to the arbiter, who reconstructs and sends the resulting secret to the adversary.[9] At the end, after the judge terminates in the punishment phase, the arbiter outputs compensations $(\mathsf{comp}, \mathsf{P}_i)$ to indicate that a shareholder P_i should be compensated for a punishment. The only restriction on Arb is that it cannot compensate a party that is not punished, i.e., if Arb outputs $(\mathsf{comp}, \mathsf{P}_i)$, then J must have output $(\mathsf{punish}, \mathsf{P}_i)$ in the punishment phase. With this setting, we model the fact that the arbiter is able to compensate the losses of the participants after they potentially learn the secret, rather than to serve a trusted third party that could reconstruct the secret for them. Let $\perp$ be the arbiter that never compensates.

Snitching strategy. A snitching strategy $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for an adversarial strategy $(\mathsf{Arb}, \mathcal{A})$ is a PPT algorithm that, on input $j \in [a], i \in [n]$, takes control of the dealer and all parties controlled by the subadversaries in $\mathcal{A}$ except for $\mathcal{A}_j$. The goal of $\mathcal{B}_1$ is to betray the subadversary $\mathcal{A}_j$, and the goal of $\mathcal{B}_2$ is to betray the arbiter of the original adversarial strategy $(\mathsf{Arb}, \mathcal{A})$ as described in $\mathsf{Snitching}$ (Figure 3).

Further, we define the distinguishing advantage $\mathsf{Adv}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi; \kappa]\!]$ as

$$|\Pr[\mathsf{Sharing}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi, 0; \kappa]\!] = 1] - \Pr[\mathsf{Sharing}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi, 1; \kappa]\!] = 1]|$$

with the $\mathsf{Sharing}$ experiment from Fig. 3.

Strong punishability. Finally, we can define the guarantees for security against insured contracts:

Definition 4 (Strong Punishability). *A (t, n)-SSSS scheme Π achieves* strong punishability *if for all arbiters Arb and distributed adversaries $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ there exists a snitching strategy $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that*

$$\begin{aligned}&\Pr[\exists j \in [a] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B}_1 \leftrightarrows (\Pi, \mathsf{Arb}, \mathcal{A}, j, 0); \kappa]\!] = 1] + \\ &\Pr[\exists i \in [n] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B}_2 \leftrightarrows (\Pi, \mathsf{Arb}, \mathcal{A}, 0, i); \kappa]\!] = 2] \\ &\geq \mathsf{Adv}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi; \kappa]\!] - \mathsf{negl}(\kappa).\end{aligned}$$

Intuitively, the $\mathsf{Snitching}$ experiment (Figure 3), models a snitching strategy $\mathcal{B}$ betraying the subadversary $\mathcal{A}_j$ of the original strategy $(\mathsf{Arb}, \mathcal{A})$ in one of two ways. The first way (modeled by $\mathcal{B}_1$) is by making the Judge J output $(\mathsf{punish}, \mathsf{P})$ for some $\mathsf{P} \in \mathcal{C}_j$ (i.e. P is controlled by $\mathcal{A}_j$) and having P not compensated by the arbiter Arb (Step 7a), causing the experiment to output 1. In this case, the original strategy is flawed, as $\mathcal{A}_j$ is assumed to get punished without being compensated by the arbiter. Another way (modeled by $\mathcal{B}_2$) is to trigger compensation from Arb for some $\mathsf{P} \notin \mathcal{C}_j$, indicating that the arbiter is flawed as $\mathcal{B}$ can commit "insurance fraud" (Step 7b), causing the experiment to output 2. If for some strategy $(\mathsf{Arb}, \mathcal{A})$ there exists no snitching strategy that can cheat a subadversary in one of the ways mentioned above, then the protocol is insecure. We highlight that in Step 5 of $\mathsf{Snitching}$ experiment, we still let the adversary choose S_0 and S_1 but discard them such that the $\mathsf{Snitching}$ and $\mathsf{Sharing}$ experiments are not trivially distinguishable by $\mathcal{A}$ already in the sharing phase.

For completeness, we also present the *weak punishability* as a special case of strong punishability where there are no arbiters.

Definition 5 (Weak Punishability). *A (t, n)-SSS scheme Π achieves* weak punishability *if for all distributed adversaries $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ there exists a snitching strategy $\mathcal{B}$ such that*

$$\begin{aligned}&\Pr[\exists j \in [a] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \perp, \mathcal{A}, j, 0); \kappa]\!] = 1] \\ &\geq \mathsf{Adv}[\![(\perp, \mathcal{A}) \leftrightarrows \Pi; \kappa]\!] - \mathsf{negl}(\kappa).\end{aligned}$$

It follows directly from the definitions that strong punishability implies weak punishability. Indeed, the $\mathsf{Snitching}$ experiment never outputs 2 since the arbiter $\perp$ never compensates. We continue to just write *punishability* in the context of statements that apply to both strong and weak punishability.

[9] Alternative models may allow the arbiter to send messages prior to the punishment phase, but then would have to give access to the arbiter's oracle queries to the snitching strategy. We instead adopt the above approach to streamline the analysis and avoid introducing complexity that does not materially affect our results.

3.3 Remarks on the Definition

Dissecting the punishability definition. Similarly to [18], our punishability definitions argue that for all adversarial strategies that learn non-trivial information about the secret, there exists a snitching strategy that can thwart the original adversarial strategy. The probability of betraying the adversarial strategy is required to be at least negligibly close to the advantage of the adversarial strategy as measured in an indistinguishability experiment. We arrive at our more general definition of (strong) punishability through the following steps. First, observe the order of the quantifiers. Initially, any adversarial strategy $\mathcal{A}$ is fixed. Then, we require that for this strategy, there exists a snitching strategy $\mathcal{B}$ such that the probability that there exists a subadversary $\mathcal{A}_j \in \mathcal{A}$ that can be cheated by $\mathcal{B}$ is greater or equal to the advantage of $\mathcal{A}$. Crucially, the snitching strategy $\mathcal{B}$ needs to be fixed *before* sampling the randomness of the experiment and j is chosen *after* sampling the randomness of the experiment. This reflects the fact that, in practice, *anyone can become a snitcher*. To see this, consider the following, hypothetical attack strategy: $\mathcal{A}$ contains one subadversary per party. All subadversaries execute an MPC protocol, where one subadversary is selected at random to receive the secret shares of all other shareholders and reconstruct the secret. In practice, this shareholder could realize that it learned enough information to snitch on the other shareholders, and do so only *after* the execution of the MPC protocol. Our definition accurately reflects this scenario, as for (almost) all random coins of the experiment, there exists at least one j that $\mathcal{B}$ can snitch on by corrupting the subadversary $\mathcal{A}_{j'}$ with $j \neq j'$ that learns all the secret shares. Clearly, if the snitching strategy $\mathcal{B}$ could only adaptively corrupt subadversaries or were required to *always* snitch on a fixed subadversary (i.e. fixing j before the random coins of the experiment), we could only reflect scenarios similar to the above with a loss[10]. On the other hand, if the choice of the snitching strategy $\mathcal{B}$ could be made *after* fixing the random coins of the experiment, then the definition would be self-fulfilling, as this essentially gives the Turing machine $\mathcal{B}$ knowledge of all internal secrets of the experiment and $\mathcal{A}_j$. In contrast to [18], our definition allows both *cooperative* and *active* snitching strategies. While in [18], the snitcher can do no more but follow the original adversarial strategy and send its trace to the dealer, our definition allows $\mathcal{B}$ to recorrupt all subadversaries but $\mathcal{A}_j$ and control the dealer. Further, $\mathcal{B}$ can actively divert from $\mathcal{A}$ on behalf of the recorrupted subadversaries. This is not only more general and admits more protocols, but it is also necessary for our construction with strong punishability (cf. Section 7).

On top of that we introduce the notion of arbiter attacks in our strong punishability definition. We model arbiters as PPT algorithms that can observe essentially everything the judge does, including its inputs, outputs and state. Additionally, the arbiter can receive messages from subadversaries, modelling the fact that subadversaries may try to prove to the arbiter that they are adhering to the collusion strategy. In particular, we want to capture that subadversaries may try to send additional information that allows the arbiter to distinguish between a real punishment and a self-punishment. In the end, arbiters can choose whether to compensate a party based on this information. Hence, we consider the arbiter to be part of the adversarial strategy and quantify over all possible arbiters. This model covers a wide range of arbiters, including those that are based on smart contracts on blockchains. Note that our arbiter model is purely cryptographic. It does not reason about who is the issuer of the insurance, nor does it model any compensation amounts. While both our weak punishability definition and the punishability definition in [18] essentially say "every adversarial strategy *can be betrayed*", in the context of insurances and strong punishability we need to worry about what happens *if the punished party is compensated*. We argue that a scheme is secure in the presence of arbiters if it is secure either in the traditional sense, i.e., there exists a subadversary that can be punished by the snitcher and this subadversary *is not compensated* (c.f. step 7a in the Snitching experiment); or, if the snitcher can cheat the arbiter by triggering compensation for himself (c.f. step 7b in the Snitching experiment).

Finally, splitting the adversarial strategy into $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ models the fact that the insurance might be probabilistic and only compensate the party with some probability p_{comp}. Then, while the "insurance fraud"

[10] Actually, the "worst-case" scenario would be different from the above as instead of sending all reconstruction information to one party, the strategy could distribute it among all parties but one, if the scheme allows for it. In that case any adaptive snitching strategy would have a loss of at least $1/n$ in an (n, n) scheme, which is the probability of *not* recorrupting the party that learns no information.

only succeeds with probability p_{comp}, the "real punishment" occurs with probability $1 - p_{\mathsf{comp}}$, avoiding the loss in the security definition. We note that we could, instead, opt to model the snitcher's adaptivity in deciding whether to

- issue a real punishment or extort the arbiter
- choose a target in case of real-punishment, or
- choose a target party $i \in [n]$ on behalf of which to extort the arbiter;

by introducing a loss factor in the equation defining punishability. This, however, would lead to a definition that does not generalize weak punishability, unlike the formulation of Definition 4.

Punishability and Unframability Implies Secrecy. Interestingly, our definitions do not explicitly contain a secrecy requirement that any (monolithic) adversary that corrupts less than t parties can not learn non-trivial information about the secret. We argue that this property is already implied by unframability and punishability. To see this, consider a monolithic adversary $\mathcal{M}$ that corrupts $t-1$ parties and has non-negligible advantage ε in distinguishing the secret. We build a distributed adversary $\mathcal{A} := (\mathcal{M})$ that runs in the punishability experiment without arbiters and contains $\mathcal{M}$ as the only subadversary. Clearly, $\mathcal{A}$ has the same advantage ε as $\mathcal{M}$. Observe though, that there exists no snitching strategies $\mathcal{B}$ that can make Snitching output 1: First, step 7b can never be satisfied, as the arbiter *never* compensates. Second, in order to satisfy step 7a, the snitching strategy $\mathcal{B}$ would need to make the judge punish a party that is not corrupted by $\mathcal{M}$. Unframability though states that this can happen only with negligible probability. Hence, $\mathcal{A}$ has non-negligible advantage ε in distinguishing the secret, but all snitching strategies against $\mathcal{A}$ win the Snitching experiment with at most negligible probability, which contradicts punishability. We note that a similar argument holds for the punishability definition in [18].

4 Construction

In this section, we construct our main result, the n-out-of-n secret sharing with snitching scheme, abbreviated n-out-of-n SSS or (n,n)-SSS, which will be later used to construct the threshold scheme (cf. Section 8) and the strong secret sharing with snitching scheme (cf. Section 7). While a (n,n)-SSS has already been constructed in [18], the transform of Section 7 cannot be instantiated using the existing construction, as outlined in Remark 2.

4.1 Hash-based encoding

As a building block for our SSS scheme, we want to encode a secret in such a way that its decoding requires a massive computation of hashes. We also want the encoding to be efficient and require $O(1)$ hashes.

Encoding 1 *Let $d < \kappa$. For $b \in \{0,1\}$, let $H_b(x) = H(b \,||\, x)$.*

$\mathsf{Enc}_d^\kappa(m)$		$\mathsf{Dec}_d^\kappa(\mathsf{hint}, c)$	
1:	$Y \leftarrow\!\$\ \{0,1\}^\kappa; Z \leftarrow\!\$\ \{0,1\}^d$	1:	*brute-force Z from* hint
2:	$\mathsf{checksum} := H_0(Y,Z)$	2:	$\mathsf{pad} := H_1(Y,Z)$
3:	$\mathsf{hint} := (Y, \mathsf{checksum})$	3:	**return** $\mathsf{pad} \oplus c$
4:	$\mathsf{pad} := H_1(Y,Z)$		
5:	**return** $(\mathsf{hint}, \mathsf{pad} \oplus m)$		

Note that for this encoding to be decodable by PPT parties, we need $d = O(\log \kappa)$. The encoding of a secret requires 2 evaluations of H, whereas the decoding requires an expected $2^{d-1} + 1 = O(\kappa)$ evaluations of H. Note that the collision-resistance of H implies that the encoding has negligible correctness error, as the brute-forcing is poly-time.

The sharing phase $\Pi.\mathsf{share}$ of the (n,n)-SSS

1. All parties take as input 1^κ. The dealer D also takes as input $S \in \{0,1\}^*$.
2. D samples $X \leftarrow\!\!\$\ \{0,1\}^\kappa$ and computes $(X_1,\ldots,X_\kappa) := \mathsf{share}_{\lfloor\kappa/2\rfloor,\kappa}(X)$.
3. For $j := 1,\ldots,\kappa$, the dealer D computes $\mathcal{E}_j \leftarrow \mathsf{Enc}_d(X_j)$, making sure that partial nonces Y_j (cf. encoding 1) are unique for each j.
4. The parties $\mathsf{D},\mathsf{P}_1,\ldots,\mathsf{P}_n$ execute the randomized functionality $\mathcal{F}$:

   Functionality $\mathcal{F}$

   (a) **"oblivious sharing"**:
      i. The dealer's input is $\{\mathcal{E}_j\}_{j=1}^\kappa$. Parties $\mathsf{P}_1,\ldots,\mathsf{P}_n$ take no input.
      ii. Choose a random set $\mathcal{U} \subseteq [\kappa]$ such that $|\mathcal{U}| = \lfloor\kappa/2\rfloor$. Let $(U_1,\ldots,U_n) := \mathsf{share}_{n,n}(\mathcal{U})$.
      iii. For all $j \in \mathcal{U}$ let $(\mathcal{E}_1^j,\ldots,\mathcal{E}_n^j) \leftarrow \mathsf{share}_{n,n}(\mathcal{E}_j)$ and $S_i = \{\mathcal{E}_i^j\}_{j\in\mathcal{U}}$.

   (b) **"computing output commitments"**:
      i. Sample $R_1,\ldots,R_n \leftarrow\!\!\$\ \{0,1\}^\kappa$.
      ii. For $i \in [n]$, let $\Gamma_i := \mathsf{commit}(U_i;R_i)$.

   (c) **"output":**
      i. D and each P_i output $(\Gamma_1,\ldots,\Gamma_n)$. Each P_i outputs (R_i,U_i,S_i).

5. The dealer sends $(\Gamma_1,\ldots,\Gamma_n)$ to the judge J.
6. Each shareholder P_i confirms that the message the judge J received in the previous step is equal to Γ_i that she got from the computation of $\mathcal{F}$.
7. The dealer D computes $A := \mathsf{Encr}(X,S)$ and sends a message $(\mathsf{ciphertext},A)$ to the parties, who store it.

Fig. 4: The sharing phase of our n-out-of-n SSS scheme Π. Note that the parties output their respective sets $S_i = \{\mathcal{E}_i^j\}_{j\in\mathcal{U}}$ in the order of increasing index j.

4.2 Construction of the n-out-of-n SSS scheme

Our n-out-of-n SSS scheme Π is presented in Figs. 4 to 6. We now outline its construction, which conceptually consists of two parts: we first share a random secret $X \in \{0,1\}^\kappa$ and later use it to share an arbitrary secret. The scheme uses the following schemes in a black-box way: (i) a u-out-of-v secret sharing scheme $(\mathsf{share}_{u,v},\mathsf{rec}_{u,v})$, (ii) a commitment scheme $(\mathsf{commit},\mathsf{open})$, (iii) a symmetric encryption scheme $(\mathsf{Encr},\mathsf{Decr})$.

The dealer aims to share the random secret X in such a way that an "illegal" reconstruction may result in snitching. He first internally splits the random secret X into shares $X_1,\ldots,X_\kappa$, using $\lfloor\kappa/2\rfloor$-out-of-κ secret-sharing. Then, he encodes each X_j's as $\mathcal{E}_j$ using the Enc_d encoding. Subsequently, the parties jointly compute the following functionality $\mathcal{F}$. The dealer provides the encodings $\mathcal{E}_j$ as his input to $\mathcal{F}$. Within $\mathcal{F}$, a subset $\mathcal{U} \subset [\kappa]$ of size $\lfloor\kappa/2\rfloor$ is chosen at random, and those encodings $\mathcal{E}_j$ with $j \in \mathcal{U}$, together with the index set $\mathcal{U}$, are shared between the parties P_i using the n-out-of-n secret sharing. The dealer stores the sampled values Y_j (which are a part of the encodings $\mathcal{E}_j$) for the rest of the protocol.

The reconstruction is a straightforward inverse of this procedure: the shareholders first reconstruct $\mathcal{U}$, and then the $\mathcal{E}_j$ for $j \in \mathcal{U}$. The $\mathcal{E}_j$ are decoded using Dec_d and the decoded values are used to recover the secret X.

The proof of illegal reconstruction accepted by the judge J is a set $\mathcal{W} \subset \mathcal{U}$ that is *large enough*, cf. Fig. 5. The main idea of snitching is that $\mathcal{U}$ can be computed only by the shareholders *jointly*, and in order to reconstruct X, the shareholders need to compute a massive amount of hashes. It is infeasible to compute all these queries in MPC, so a substantial number of them will have to be computed individually by the shareholders. This means that one of the shareholders has to know a large set Trace of the values Y_j sampled during the encoding, which can then be submitted to the dealer. The dealer, upon receiving Trace inspects the values and finds the indices j matching the Y_j. The set of these indices is denoted $\mathcal{W}$ and is submitted

The punishment phase $\Pi.\mathsf{punish}$ of the (n,n)-SSS

1. The snitchers in $\mathcal{B}$ send a message $(\mathsf{snitch}, \mathsf{P}, \mathsf{Trace})$ to the dealer D (here Trace is a set of values, see the page 15 for an explanation).
2. The dealer D computes the set $\mathcal{W} = \{w \mid Y_w \in \mathsf{Trace}\}$. If $|\mathcal{W}| \geq \lfloor \kappa/8 \rfloor$, then D sends $(\mathsf{snitch}, \mathsf{P}, \mathcal{W})$ to the judge J.
3. Each shareholder P_i sends their decommitment (U_i, R_i) to the judge.
4. If for some $i \in [n]$ one of the following conditions happened:
   (a) P_i did not send the message in step 3;
   (b) P_i sent an invalid decommitment, i.e., $\mathsf{commit}(U_i; R_i) \neq \Gamma_i$;
   then the judge J outputs $(\mathsf{punish}, \mathsf{P}_i)$ and terminates the protocol.
5. Let $\mathcal{U} := \mathsf{rec}_{n,n}(U_1, \ldots, U_n)$. If $\mathcal{W} \subseteq \mathcal{U}$, then she outputs $(\mathsf{punish}, \mathsf{P})$.

Fig. 5: The punishment phase of our n-out-of-n SSS scheme Π. Note that the dealer D as well as the set of snitchers are formally controlled by $\mathcal{B}$.

The reconstruction phase $\Pi.\mathsf{rec}$ of the (n,n)-SSS

1. Let $\mathcal{U} := \mathsf{rec}_{n,n}(U_1, \ldots, U_n)$.
2. For $j \in \mathcal{U}$, let $X_j := \mathsf{Dec}\left(\mathsf{rec}_{n,n}(\mathcal{E}_1^j, \ldots, \mathcal{E}_n^j)\right)$.
3. Let $X := \mathsf{rec}_{\lfloor \kappa/2 \rfloor, \kappa}(\{(j, X_j)\}_{j \in \mathcal{U}})$.
4. Let $(\mathsf{ciphertext}, A)$ be the message in step 7 of the sharing phase. Output $\widehat{S} := \mathsf{Decr}(X, A)$.

Fig. 6: The reconstruction phase of our n-out-of-n SSS scheme Π. Note that in the sharing phase the parties receive the shares $\mathcal{E}_i^j$ in the order of increasing j, so they can match their shares in step 2.

by the dealer D to the judge J. Note that providing even a single index outside of $\mathcal{U}$ would render the proof invalid. This is necessary to protect honest parties against framing attacks, where a malicious dealer just tries to guess $\mathcal{W}$.

As stated before, in order to punish any party, the judge has to receive a large enough subset $\mathcal{W} \subset \mathcal{U}$. Intuitively speaking, since the set $\mathcal{U}$ is generated within the functionality $\mathcal{F}$ by all the parties (including the dealer D) and subsequently secret-shared among all the shareholders, no party possesses any information on $\mathcal{U}$ provided that at least one of them is honest. Hence, the parties trying to frame an honest party will have to guess a large subset $\mathcal{W} \subset \mathcal{U}$ without knowing $\mathcal{U}$, which only succeeds with negligible probability. The details of this argument appear in Section 5.

To verify the proof, the judge J needs to check whether $\mathcal{W} \subset \mathcal{U}$. However, his state is public, and he cannot store any secrets, including $\mathcal{U}$. This is why the judge only stores the commitments to parties' shares U_i of $\mathcal{U}$, which they are supposed to decommit in the punishment phase *after* the submission of $\mathcal{W}$, under the threat of punishment if they fail to.[11]

Finally, we share the real secret S using the random secret X as a key as $\mathsf{Encr}(X, S)$. Since the judge, which is supposed to be implemented as a smart contract, cannot participate in the MPC computation, the dealer submits the commitments to the judge, and the submissions are confirmed by the shareholders. Note that dishonest shareholders could lie about the commitments being invalid, causing an early protocol abort and reconstruct X with impunity. This is why our scheme admits such a two-stage design.

We refer to Section 6 for a discussion on the parameter d and the achieved complexity.

[11] The parties do *not* decommit to the shares of encodings $\mathcal{E}_j$ for the purpose of punishment. This is intended as we want to avoid (i) submitting large amounts of data to a smart contract (ii) increasing the data leakage beyond what the dishonest shareholders did.

5 Security analysis

In this section we provide a security proof for our (n, n)-SSS scheme.

Theorem 2. *Fix the number of parties $n \in \mathbb{N}$, the security parameter κ, the difficulty parameter $d \in \mathbb{N}$ such that $7 \leq d \leq \kappa$, and the bound on the number of slow queries σ such that $\sigma \leq 2^{d-7} \cdot \kappa$. Assume that: (a) $(\mathsf{Encr}, \mathsf{Decr})$ is a semantically secure encryption scheme, (b) $(\mathsf{commit}, \mathsf{open})$ is a commitment scheme, and (c) functionality $\mathcal{F}$ is implemented by a UC-composable maliciously secure MPC protocol.*

Then, the construction of Π presented in Section 4.2 is an (n, n)-SSS scheme with respect to Ω_H.

To prove the theorem, we must argue that Π is correct, unframable, and (weakly) punishable. The correctness is straightforward since every decoding has a negligible correctness error. We will now prove the unframability and the weak punishability of the protocol.

5.1 Unframability

On a high level, unframability holds because without the honest party's share of $\mathcal{U}$, the adversary can only guess which indices j are discarded by the functionality $\mathcal{F}$ in the sharing phase. In order to punish the honest shareholder, the adversary needs to guess any subset $\mathcal{W} \subset \mathcal{U}$ with $|\mathcal{W}| = \lfloor \kappa/8 \rfloor$ and $|\mathcal{U}| = \lfloor \kappa/2 \rfloor$. The probability of guessing such a $\mathcal{W}$ is bounded by $\binom{|\mathcal{U}|}{|\mathcal{W}|} / \binom{\kappa}{|\mathcal{W}|} = \mathsf{negl}(\kappa)$. We now follow with a formal proof.

Proof. Recall that, for a fixed honest shareholder P_{i^*}, the snitching message submitted to the judge J by the dealer D, necessary to frame the shareholder P_{i^*}, is a subset of indices $\mathcal{W} \subset \mathcal{U}$ of size at least $\lfloor \kappa/8 \rfloor$. The main intuition for the proof is that guessing such subset without any extra information succeeds only with negligible probability.

The proof of unframability follows through a series of hybrid arguments. We start with an experiment where the sharing, snitching and punishment phases are run with an honest shareholder P_{i^*}. In subsequent hybrids, we replace the cryptographic building blocks by their idealizations. In the final experiment, the adversary is forced to guess the subset $\mathcal{W} \subset \mathcal{U}$ with no extra information. Detailed analysis follows and Fig. 7 outlines the structure of the proof.

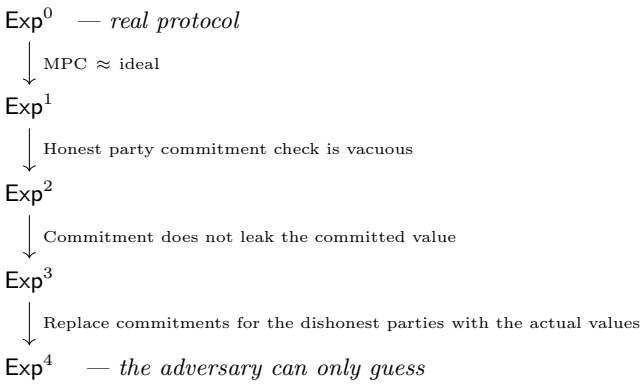


Fig. 7: Structure of the unframability proof for a fixed, honest party P_{i^*}

If the judge J outputs $\overline{\mathsf{Error}}$ in the sharing phase, framing is clearly impossible. We henceforth assume that the contrary event $\overline{\mathsf{Error}}$ occurs. Fix an honest party P_{i^*}. Consider the monolithic adversary $\mathcal{M}^0$ that corrupts the dealer D and some of the shareholders $\mathsf{P}_{j_1}, \ldots, \mathsf{P}_{j_m}$ with $m < n$. Let Exp^0 be the $\mathsf{Unframability}^{\Pi}_{\mathcal{M}^0}$ experiment. Let $\mathsf{Punish}^n_{i^*}$ be the event that the experiment Exp^n outputs 1.

We first show that we can instead consider an experiment Exp^1 with an adversary $\mathcal{M}^1$, where in Exp^1 we replace the instantiation Φ of the MPC sub-routine of protocol Π with the ideal functionality $\mathcal{F}$. Concretely, we will show the following lemma.

Lemma 2. *For any monolithic adversary $\mathcal{M}^0$ in experiment Exp^0, there exists a monolithic adversary $\mathcal{M}^1$ in experiment Exp^1 and a negligible function negl such that:*

$$\Pr\left[\mathsf{Punish}^0_{i^*}\right] \leq \Pr\left[\mathsf{Punish}^1_{i^*}\right] + \mathsf{negl}(\kappa). \tag{3}$$

Proof. To prove Lemma 2, we use the universal composition theorem (cf. Theorem 1) and the fact that protocol Φ realizes $\widetilde{\mathcal{F}}$ in the universal composability model. Notice that $\widetilde{\mathcal{F}}$ is an "augmented version" of the functionality $\mathcal{F}$ as given in Fig. 4, where $\widetilde{\mathcal{F}}$ is adjusted for universal composability.[12] Let Π^{Φ} be the SSS protocol consisting of the sharing and reconstruction phase, where in the sharing phase, we replace the functionality $\widetilde{\mathcal{F}}$ with Φ. Moreover, denote with $\Pi^{\widetilde{\mathcal{F}}}$ the same protocol except that in the sharing phase we use the augmented ideal functionality $\widetilde{\mathcal{F}}$. We consider the following two executions. $\mathsf{EXEC}_{\Pi^{\Phi},\mathcal{M}^0,\mathcal{E}}$ denotes the execution of the real protocol Π^{Φ} with adversary $\mathcal{M}^0$ and an environment $\mathcal{E}$. Notice that this corresponds essentially to Exp^0 when $\mathcal{E}$ is a "dummy" environment only delivering inputs to the parties. On the other hand, denote with $\mathsf{EXEC}^{\widetilde{\mathcal{F}}}_{\Pi,\mathcal{M}^1,\mathcal{E}}$ the execution of the protocol $\Pi^{\widetilde{\mathcal{F}}}$ in the presence of an adversary $\mathcal{M}^1$ running in the $\widetilde{\mathcal{F}}$-hybrid world. Notice that this essentially corresponds to an execution of Exp^1. By the composition theorem, we know that for any adversary $\mathcal{M}^0$ in the real world, there exists an adversary $\mathcal{M}^1$ in the $\widetilde{\mathcal{F}}$-hybrid world such that for any environment $\mathcal{E}$, we have:

$$\mathsf{EXEC}_{\Pi^{\Phi},\mathcal{M}^0,\mathcal{E}} \overset{c}{\approx} \mathsf{EXEC}^{\widetilde{\mathcal{F}}}_{\Pi,\mathcal{M}^1,\mathcal{E}}. \tag{4}$$

Given Eq. (4) we get:

$$\Pr\left[\mathsf{Punish}^0_{i^*}\right] - \Pr\left[\mathsf{Punish}^1_{i^*}\right] \leq \mathsf{negl}(\kappa), \tag{5}$$

which proves the statement of the lemma. Notice that Eq. (5) follows because if there is a noticeable difference between the probability that party P_i is punished in Exp^0 and Exp^1, then we can use this difference to break Eq. (4). Finally, notice that the augmented functionality $\widetilde{\mathcal{F}}$ is essentially as $\mathcal{F}$ except that the adversary may early abort the protocol. It is easy to see that early aborts do not affect the probability of $\mathsf{Punish}^1_{i^*}$ as the sharing phase will terminate with error, and punishment of part i is not possible. □

Let us now consider an experiment Exp^2, which is the same as Exp^1 except that we do the following changes to $\Pi.\mathsf{punish}$:

1. The honest party P_{i^*} sends just U_{i^*} in step 3, i.e., no longer sends the randomness R_{i^*} for opening the commitment;
2. we eliminate the verification checks of step 4 for the honest party P_{i^*}.

All shareholders P_i for $i \neq i^*$ are handled as previously. Notice that the above means that in Exp^2, we cannot check anymore if Γ_{i^*} is a commitment to U_{i^*}. However, since P_{i^*} is honest, and the commitment scheme satisfies perfect correctness, in Exp^1 she will always pass the check in step 4b. Hence, for any adversaries running in the respective experiments, we obtain that:

$$\Pr\left[\mathsf{Punish}^1_{i^*}\right] = \Pr\left[\mathsf{Punish}^2_{i^*}\right] \tag{6}$$

Next, consider the experiment Exp^3, which is as Exp^2 with further changes to $\Pi.\mathsf{punish}$: we replace P_{i^*}'s commitment $\Gamma_{i^*} := \mathsf{commit}(U_{i^*}; R_{i^*})$ with a commitment to the zero string $00\ldots0$ of the same length. Notice that both in Exp^2 and Exp^3, the commitment Γ_{i^*} is never opened, and the randomness R_{i^*} of the commitment never appears. Hence, by the computationally hiding property of the commitment scheme,

[12] Looking ahead, at the end of this proof, we will argue why these adjustments of $\widetilde{\mathcal{F}}$ do not matter for our analysis.

Exp^2 and Exp^3 are computationally indistinguishable. Hence, for any adversaries running in the respective experiments, we obtain that:

$$\Pr\left[\mathsf{Punish}^3_{i^*}\right] = \Pr\left[\mathsf{Punish}^2_{i^*}\right] + \mathsf{negl}(\kappa) \tag{7}$$

Finally, in experiment Exp^4 replace the commitments with the actual values for shareholders other than P_{i^*}. More formally,

1. Sharing phase step 4c.i: the functionality $\mathcal{F}$ outputs U_i instead of Γ_i to the parties P_i with $i \neq i^*$;
2. sharing phase step 5: the dealer sends U_i instead of Γ_i to the judge for $i \neq i^*$;
3. sharing phase step 6: the shareholders P_i with $i \neq i^*$ check if the value that was sent to judge is equal to U_i;
4. punishment phase: steps 3 and 4 are eliminated for the parties $\mathsf{P}_i, i \neq i^*$.

First, observe that due to the perfect binding of the commitment, the commitment Γ_i will always be later opened to U_i. Moreover, in both experiments, the adversary's view in the sharing phase is independent of U_{i^*}. Since $\mathcal{U}$ is perfectly secret shared using the n-out-of-n secret sharing, this implies that in both experiments, the adversary's behavior in the sharing phase is independent of $\mathcal{U}$. Finally, notice that in the punishment phase of Exp^3 when it sees U_{i^*} in clear, the shareholder P_i always has the choice not to send the opening of the commitment Γ_i. But this leads to a punishment of that shareholder and an early abort. Thus, we can conclude that

$$\Pr\left[\mathsf{Punish}^3_{i^*}\right] = \Pr\left[\mathsf{Punish}^4_{i^*}\right] \tag{8}$$

Observe that in Exp^4, the view of the adversary is independent of $\mathcal{U}$ until the punishment phase starts. Hence, the choice of $\mathcal{W}$ is independent of $\mathcal{U}$. Moreover, the adversary is committed to the U_i as this value has been sent to the judge during the sharing phase. Hence, the value used for $\mathcal{U}$ in step 5 of the punishment phase is chosen uniformly at random by the functionality. This implies that in experiment Exp^4, the adversary can only guess a subset $\mathcal{W}$ of $\mathcal{U}$ such that $|\mathcal{W}| = \lfloor \kappa/8 \rfloor$. This probability is maximized for $\mathcal{W}$ of minimal cardinality and can be thus bounded by $\binom{|\mathcal{U}|}{|\mathcal{W}|} / \binom{\kappa}{|\mathcal{W}|} = \mathsf{negl}(\kappa)$. Indeed,

$$\frac{\binom{|\mathcal{U}|}{|\mathcal{W}|}}{\binom{\kappa}{|\mathcal{W}|}} = \frac{\frac{|\mathcal{U}|!}{|\mathcal{W}|!(|\mathcal{U}|-|\mathcal{W}|)!}}{\frac{\kappa!}{|\mathcal{W}|!(\kappa-|\mathcal{W}|)!}} = \prod_{i=0}^{|\mathcal{W}|-1} \left(\frac{|\mathcal{U}|-i}{\kappa-i}\right) \leq \left(\frac{|\mathcal{U}|}{\kappa}\right)^{|\mathcal{W}|} \leq \left(\frac{|\mathcal{U}|}{\kappa}\right)^{\lfloor \frac{\kappa}{8} \rfloor}$$

which is clearly negligible in κ, since we assumed that $|\mathcal{U}| = \lceil 3\kappa/4 \rceil$, so and $|\mathcal{W}| \geq \lfloor \kappa/8 \rfloor$.

Putting this together with Eq. (3) and Eqs. (6) to (8), we get that any monolithic adversary in Exp^0 has negligible probability to break the unframability property. Since Exp^0 is the real experiment, this concludes the proof of unframability. □

5.2 Weak punishability

We will next prove the weak punishability of the n-out-of-n scheme. Assume the dealer D is honest and consider a distributed adversary $\mathcal{A}$. Let us consider the snitching strategy $\mathcal{B}$ that, for any $\ell \in [a]$, recorrupts without changes all the subadversaries except $\mathcal{A}_\ell$ in the $\mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \bot, \mathcal{A}, \ell); \kappa]\!]$ experiment.

Let us now introduce some auxiliary terminology. For every j, let Y_j and Z_j be the values corresponding to Y and Z in the encoding $\mathcal{E}_j$ respectively (cf. encoding 1). Recall that each Y_j uniquely identifies the index j. As in [17, 18], to reason about the information each subadversary submitted to the oracle, we define the *hash transcript* to be the sequence $\mathcal{T}$ of oracle inputs that Ω_H[13] received from all subadversaries. For $j \in [\kappa]$, a *scratch for* j is an element q of $\mathcal{T}$ of the form $q = 0 \,||\, Y_j \,||\, \alpha$. Call a scratch q for j *successful* if $\alpha = Z_j$.

A scratch is *slow* if the corresponding query was submitted through Ω_H^{MPC} and *fast* if it was submitted directly to Ω_H (cf. Section 3). Note that the adversary has to decide whether a scratch will be slow or

[13] The oracle Ω_H is assumed to be called by Ω_H^{MPC} under the hood.

fast *before* it knows whether or not it was successful. For an execution of $\mathcal{A}$ against Π, define the following variables:

$$N_{\mathsf{succ}} := |\{j\colon \text{ a successful scratch was made for } j\}| \tag{9}$$
$$N_{\mathsf{fast}} := |\{j\colon \text{ a fast scratch was made for } j\}| \tag{10}$$

Let Error denote the event that the judge J outputs Error during the sharing procedure. By *$\mathcal{A}$'s view* on the experiment $\mathsf{Sharing}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi, b; \kappa]\!]$ we mean the local randomness of the machines in $\mathcal{A}$ and all the information received by these machines. To make our notation unambiguous, for $b \in \{0,1\}$ and an event $\mathcal{E}$ we will sometimes write $\Pr_{b,\kappa}[\mathcal{E}]$ to denote the probability of $\mathcal{E}$ in the $\mathsf{Sharing}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi, b; \kappa]\!]$ experiment, and write $\Pr_{\mathsf{sn},\ell,\kappa}[\mathcal{E}]$ to denote the probability of $\mathcal{E}$ in the $\mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \bot, \mathcal{A}, \ell); \kappa]\!]$ experiment. The following fact will be useful, which follows from the perfect secrecy of the secret sharing scheme:

Lemma 3. *We have that*

$$\Pr_{0,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] = \Pr_{1,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right]$$

Proof. If $N_{\mathsf{succ}} < \lfloor \kappa/2 \rfloor$, then the number of shares $\mathcal{A}$ learns any information about is lower than the secret-sharing threshold. Hence, by the information-theoretic secrecy of the secret sharing scheme, $\mathcal{A}$'s view does not give information on X. Therefore, the distribution of the adversary's view is the same in both cases and so the probabilities are also the same. □

The rest of the proof of the theorem is divided into three parts.

Part 1: A proof that an adversary who wins the distinguishing game makes many successful scratches. We start with relating the probability of the event "$N_{\mathsf{succ}} \geq \lfloor \kappa/2 \rfloor, \overline{\mathsf{Error}}$" in the $\mathsf{Sharing}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi, b; \kappa]\!]$ experiments with the advantage of $\mathcal{A}$ in the distinguishing game. More precisely, we prove the following holds for any $b \in \{0,1\}$:

$$\mathsf{Adv}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi; \kappa]\!] \leq \Pr_{b,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] \tag{11}$$

In either of the $\mathsf{Sharing}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi, b; \kappa]\!]$ experiments, let Out be a random variable equal to the experiment's output. Introduce the following events: $R = \{\mathsf{Out} = 1\}$, $D = \left\{N_{\mathsf{succ}} \geq \lfloor \kappa/2 \rfloor, \overline{\mathsf{Error}}\right\}$. Applying the law of total probability, we obtain that:

$$\begin{aligned}
&\mathsf{Adv}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi; \kappa]\!] = |\Pr_{0,\kappa}[R] - \Pr_{1,\kappa}[R]| \\
=&|\Pr_{0,\kappa}[R \mid D]\Pr_{0,\kappa}[D] + \Pr_{0,\kappa}\left[R \;\middle|\; \overline{D}\right]\Pr_{0,\kappa}\left[\overline{D}\right] \\
&- \Pr_{1,\kappa}[R \mid D]\Pr_{1,\kappa}[D] - \Pr_{1,\kappa}\left[R \;\middle|\; \overline{D}\right]\Pr_{1,\kappa}\left[\overline{D}\right]| && (12)\\
=&|\Pr_{0,\kappa}[R \mid D]\Pr_{0,\kappa}[D] - \Pr_{1,\kappa}[R \mid D]\Pr_{1,\kappa}[D]| && (13)\\
=&\underbrace{|\Pr_{0,\kappa}[R \mid D] - \Pr_{1,\kappa}[R \mid D]|}_{\leq 1} \cdot \Pr_{b,\kappa}[D] && (14)\\
\leq&\Pr_{b,\kappa}[D] = \Pr_{b,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] && (15)
\end{aligned}$$

Here, in Eq. (14) we have used that $\Pr_{0,\kappa}[D] = \Pr_{1,\kappa}[D]$ (cf. Lemma 3) and that if $x, y \in [0,1]$, then $|x - y| \in [0,1]$. Similarly, in Eq. (13) we noticed $\Pr_{0,\kappa}\left[\overline{D}\right] = \Pr_{1,\kappa}\left[\overline{D}\right]$ and that $\Pr_{0,\kappa}\left[R \;\middle|\; \overline{D}\right] = \Pr_{1,\kappa}\left[R \;\middle|\; \overline{D}\right]$ (which is argued in a way analogous to Lemma 3), so the terms cancel out.

Part 2: A proof that an adversary that makes many successful scratches also makes at least one fast scratch for many indices. In this part, we prove that for both b and all ℓ:

$$\Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{fast}} \geq \kappa/4, \overline{\mathsf{Error}}\right] \geq \Pr_{b,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right]. \tag{16}$$

It is easy to see that for both b

$$\Pr_{b,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] = \Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right]. \tag{17}$$

This holds because in each of the experiments, the adversary's view is the same until both events occur. Note that Eq. (17) implies that we can rewrite Eq. (16) as:

$$\Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{fast}} \geq \frac{\kappa}{4}, \overline{\mathsf{Error}}\right] \geq \Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] - \mathsf{negl}(\kappa). \tag{18}$$

In order to show Eq. (18), it suffices to prove the following:

Lemma 4. *We have that:*

$$\Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{fast}} \geq \frac{\kappa}{4} \;\middle|\; N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] \geq 1 - \mathsf{negl}(\kappa). \tag{19}$$

Proof. Assume the events $N_{\mathsf{succ}} \geq \lfloor \kappa/2 \rfloor$ and $\overline{\mathsf{Error}}$ occurred in the $\mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \bot, \mathcal{A}, \ell); \kappa]\!]$ experiment. Let $\mathcal{S}$ be the set of indices:[14]

$$\mathcal{S} := \{j \in [\kappa] \colon \text{there was a successful scratch for } j, \text{ but no fast scratch}\}.$$

A large set $\mathcal{S}$ is obviously an obstruction to punishability. Since for each $j \in \mathcal{S}$ only slow scratches were made, we partition the set $\mathcal{S}$ into two disjoint subsets:

$$\mathcal{L} := \left\{j \in \mathcal{S} \colon \text{there were } \leq 2^{d-4} \text{ slow scratches for } j\right\}$$
$$\mathcal{G} := \left\{j \in \mathcal{S} \colon \text{there were } > 2^{d-4} \text{ slow scratches for } j\right\}$$

Clearly $|\mathcal{S}| = |\mathcal{L}| + |\mathcal{G}|$. We will bound the cardinality of both subsets.

Since, the value Z_j that is being searched for is distributed uniformly over $\{0,1\}^d$, for every $\ell \in [\kappa]$ we have that $\Pr[\ell \in \mathcal{L}] \leq 2^{d-4}/2^d = 1/16$. We aim to show that $\Pr[|\mathcal{L}| \geq \kappa/8] = \mathsf{negl}(\kappa)$.

Let us write $|\mathcal{L}| = \sum_{j=1}^{\kappa} \mathbf{1}_{\mathcal{L}}(j)$ where $\mathbf{1}_{\mathcal{L}}(j) = 1$ if $j \in \mathcal{L}$ and $\mathbf{1}_{\mathcal{L}}(j) = 0$ otherwise. We want to replace the summands with i.i.d. variables. For $j \in [\kappa]$, consider the random variable

$$T_j = \text{“the value } Z_j \text{ is found within at most } 2^{d-4} \text{ calls to the oracle”}.$$

The T_j only depend on the randomness of the random oracle and are independent of the randomness of the adversary.[15] Then we have that $T_j \sim \mathsf{Bernoulli}(1/16)$ and $T_j \geq \mathbf{1}_{\mathcal{L}}(j)$.[16] Moreover, since the solutions Z_j that are being searched for are sampled independently, the variables T_j are independent. Finally, let $T = \sum_{\ell=1}^{\kappa} T_\ell$. Then, clearly, $T \geq |\mathcal{L}|$ and so:

$$\Pr\left[|\mathcal{L}| \geq \frac{\kappa}{8}\right] \leq \Pr\left[T \geq \frac{\kappa}{8}\right]. \tag{20}$$

Furthermore, $T \sim \mathsf{Bin}\,(\kappa, 1/16)$ and $\mathbb{E}[T] = \kappa/16$. Hence, by the Chernoff bound (see Lemma 1) with $\delta = 1$, we have that

$$\Pr\left[T \geq \frac{\kappa}{8}\right] = \Pr\left[T \geq 2 \cdot \frac{\kappa}{16}\right] \leq \exp\left(-1^2 \cdot \frac{\kappa}{16} \cdot \frac{1}{2+\delta}\right) = \exp\left(-\frac{\kappa}{48}\right) = \mathsf{negl}(\kappa),$$

and so, by Eq. (20), $\Pr[|\mathcal{L}| \geq \kappa/8] = \mathsf{negl}(\kappa)$, as desired.

Now consider $\mathcal{G}$. Since all these scratches for indices of $\mathcal{G}$ are slow, and we assumed that the adversary does at most $\sigma \leq 2^{d-7} \cdot \kappa$ slow scratches, we obtain that:

$$|\mathcal{G}| \leq \sigma/(2^{d-4}) \leq 2^{d-7} \cdot \kappa/(2^{d-4}) = \frac{\kappa}{8}.$$

[14] Technically, the sets $\mathcal{S}, \mathcal{L}, \mathcal{G}$ are random variables. Similarly, $\mathbf{1}_{\mathcal{L}}(j)$ is still a random variable, not a number.

[15] Note that this technical step is necessary, as the adversary might choose the indices j adaptively, and so the variables $\mathbf{1}_{\mathcal{L}}(j)$ might *not* be independent.

[16] Technically, this inequality holds almost everywhere with respect to the probability measure.

We conclude that, with overwhelming probability, we have

$$|\mathcal{S}| = |\mathcal{G}| + |\mathcal{L}| \leq \kappa/8 + \kappa/8 = \kappa/4.$$

Since we assumed that $N_{\mathsf{succ}} \geq \lfloor \kappa/2 \rfloor$ occurred, a successful scratch was made for at least $\lfloor \kappa/2 \rfloor$ indices. This means that, with overwhelming probability, there are at least $\lfloor \kappa/2 \rfloor - |\mathcal{S}| \geq \kappa/2 - \kappa/4 = \kappa/4$ successful fast scratches. Thus, Eq. (19) (and hence Eq. (16)) holds. □

Part 3: A construction and analysis of the snitching adversaries. In order to enable snitching, we modify the behavior of $\mathcal{B}$ in the following way:

1. In the sharing phase, behave exactly as in the sharing experiment.
2. Record messages sent by the controlled parties P_i to oracle Ω_H in a set $\mathcal{H}$.
3. In the punishment phase, let

$$\mathsf{Trace} := \big\{Y \in \{0,1\}^\kappa \colon \exists \alpha \in \{0,1\}^d \colon (0 \,||\, Y \,||\, \alpha) \in \mathcal{H}\big\}.$$

Send $(\mathsf{snitch}, \mathsf{Trace})$ to the dealer D. Otherwise, behave honestly.

Clearly, the overhead for snitching is linear in the number of hash queries submitted to Ω_H by $\mathcal{B}$, and, consequently, at most linear in the running time of the adversary $\mathcal{B}$. For clarity, let $\mathsf{Punished} := \{\mathsf{Out} = 1\}$. We now have the following fact:

Lemma 5.

$$\Pr_{\mathsf{sn},\ell,\kappa}\left[\mathsf{Punished} \;\middle|\; N_{\mathsf{fast}} \geq \frac{\kappa}{4}, \overline{\mathsf{Error}}\right] \geq 1 - \mathsf{negl}(\kappa). \tag{21}$$

Proof. Let $\mathsf{CommitOpen}$ denote the event that all the shareholders P_i sent their values R_i, U_i to the judge in step 3 of the punishment phase and that the decommitment was successful in step 4. We first show that Eq. (21) holds when we add $\mathsf{CommitOpen}$ to the condition, i.e., we prove the following:

$$\Pr_{\mathsf{sn},\ell,\kappa}\left[\mathsf{Punished} \;\middle|\; N_{\mathsf{fast}} \geq \frac{\kappa}{4}, \overline{\mathsf{Error}}, \mathsf{CommitOpen}\right] \geq 1 - \mathsf{negl}(\kappa). \tag{22}$$

In order to prove Eq. (22), assume that the events $N_{\mathsf{fast}} \geq \kappa/4$, $\overline{\mathsf{Error}}$ and $\mathsf{CommitOpen}$ occurred. Note that $\mathsf{CommitOpen}$ implies that the set $\mathcal{U}$ computed in step 5 of the punishment phase equals the set $\mathcal{U}$ computed by $\mathcal{F}$. This means that the distributed adversary received no information on the Y_j for $j \notin \mathcal{U}$.

Recall that each of the nonces Y_j were sampled as $Y_j \leftarrow\!\!\$\ \{0,1\}^\kappa$. However, even if the adversary tries polynomially many samples, the probability that he obtains at least one value Y_j such that $j \notin \mathcal{U}$ is still $\mathsf{negl}(\kappa)$,[17] and hence, with overwhelming probability, none of the scratches done by the adversary is for $j \notin \mathcal{U}$. Consequently, it holds that $\mathcal{W} \subseteq \mathcal{U}$ in step 5 of the punishment phase.

Fix some $\ell \in [a]$. Since $N_{\mathsf{fast}} \geq \kappa/4$, then either $\mathcal{A}_\ell$ or $\mathcal{B}$ individually knows at least $\lfloor \kappa/8 \rfloor$ elements of Trace.[18] We are done in the latter case, and in the former it is enough to choose any other $\ell' \neq \ell$ and let $\mathcal{B}$ recorrupt $\mathcal{A}_{\ell'}$ instead of $\mathcal{A}_\ell$. Therefore, w.l.o.g., assume that $\mathcal{B}$ knows at least $\lfloor \kappa/8 \rfloor$ elements of Trace, whose set is denoted by $\mathsf{Trace}_{\mathcal{B}}$. Therefore, if $\mathcal{B}$ submits his $\mathsf{Trace}_{\mathcal{B}}$ in the punishment phase, the resulting proof of punishment will be accepted by the judge and thus the $\mathsf{Punished}$ event occurs. We thus obtain Eq. (22).

Let us now show that Eq. (22) implies Eq. (21). Observe that if $\overline{\mathsf{Error}}$ occurs and $\mathsf{CommitOpen}$ does not, then $\mathsf{Punished}$ still occurs with overwhelming probability. Indeed, if the sharing phase ended successfully (i.e., $\overline{\mathsf{Error}}$ happened), then the only possibility for $\mathsf{CommitOpen}$ not to occur is that at least one of the shareholders corrupted by $\mathcal{A}_\ell$ failed to send her share to the judge in step 3 of the punishment phase. In this case it is punished by the judge, and $\mathsf{Punished}$ occurs. Hence, Eq. (22) implies Eq. (21). This finishes the proof of the claim. □

[17] We need to account for polynomially many attempts because the dealer will reject invalid nonces, cf. Fig. 5.
[18] If we want to make sure that individual parties can snitch on their own, the proof size should be $\kappa/(4n)$ instead.

We can now combine the above claim with the inequalities proven in the previous parts.

$$\begin{aligned}
&\Pr[\exists \ell\colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \bot, \mathcal{A}, \ell); \kappa]\!] = 1] \\
&= \Pr_{\mathsf{sn},\ell,\kappa}[\mathsf{Punished}] \\
&\geq \Pr_{\mathsf{sn},\ell,\kappa}\left[N_{\mathsf{fast}} \geq \kappa/4, \overline{\mathsf{Error}}\right] - \mathsf{negl}(\kappa) && (23) \\
&\geq \Pr_{b,\kappa}\left[N_{\mathsf{succ}} \geq \left\lfloor \frac{\kappa}{2} \right\rfloor, \overline{\mathsf{Error}}\right] - \mathsf{negl}(\kappa) && (24) \\
&\geq \mathsf{Adv}[\![(\bot, \mathcal{A}) \leftrightarrows \Pi; \kappa]\!] - \mathsf{negl}(\kappa), && (25)
\end{aligned}$$

where Eq. (23) comes from Lemma 5, Eq. (24) — from Part 2 of the proof, and precisely Eq. (16), and Eq. (25) — from Part 1, and precisely Eq. (11).

To finish the proof of weak punishability, we must prove that the above analysis still holds if we replace ideal functionality for $\mathcal{F}$ with an MPC protocol. The only place where we used the assumption that $\mathcal{F}$ is implemented securely is where we assumed that the shareholders do not get the Y_j's for $j \notin \mathcal{U}$. Note that, to analyze this case, we can assume that both shareholders are corrupt by a monolithic adversary $\mathcal{M}$. Hence, the analysis can be performed analogously as in the proof of unframability.

6 Complexity analysis

The slow query budget σ. Recall that $d = O(\log \kappa)$, and so Theorem 2 yields $\sigma \leq \kappa 2^{O(\log \kappa)-7} = \mathsf{poly}(\kappa)$. Taking concretely $d = (m-1)\log \kappa + 7$ for an arbitrary constant m, the bound takes the form $\sigma \leq \kappa^m$, and so our scheme supports any slow query budget σ which is polynomial in κ of an arbitrarily (but fixed a priori) degree.

Randomness complexity. The randomness used in the functionality $\mathcal{F}$ is only used for computing $\mathcal{U}$ and $R_1, \ldots, R_n$. Since we require security even if all parties but one are corrupt, each party needs to provide randomness for all the R_i. Therefore, the randomness complexity of the protocol is $\Theta(\kappa n)$ bits per party, on top of the randomness necessary to implement the MPC protocol.

Communication complexity. Each encoding $\mathcal{E}_j$ is $\Theta(\kappa)$ bits long. The dealer's input is κ values $\mathcal{E}_j$, so their input is $\Theta(\kappa^2)$ bits. Each shareholder P_i outputs $\Theta(\kappa^2)$ bits:

(1) commitment randomness R_i of $\Theta(\kappa)$ bits;
(2) share U_i of $\mathcal{U}$. Since $|\mathcal{U}| = \Theta(\kappa)$, it follows that $|U_i| = \Theta(\kappa)$ bits;
(3) shares $S_i = \{V_i^j\}_{j \in \mathcal{U}}$. As each V_i^j is a share of $\mathcal{E}_j$, and $\mathcal{E}_j$ is $\Theta(\kappa)$ bits long and $|\mathcal{U}| = \Theta(\kappa)$, the total size of S_i is $\Theta(\kappa^2)$ bits.

Therefore, the communication complexity of such $(n+1)$-party computation protocol is $\Omega(n\kappa^2)$.

7 The strong snitching transformation

This section shows how to transform our SSS scheme with weak punishability into a strong secret sharing with snitching (SSSS) scheme that achieves strong punishability.

To introduce the idea, consider for now a simplified setting with an arbitrary SSS scheme $\Pi = (\mathsf{J}, \mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n)$ with weak punishability and *non-interactive judge* in the punishment phase. We aim to prevent insurance attacks without breaking unframability and correctness. The construction is based on the idea that we allow parties the ability to self-punish themselves to essentially commit "insurance fraud" and cash in on the arbiter. The key requirement is that the self-punishment should be indistinguishable from a real punishment to any arbiter. To see the practical interpretation of this change, consider an adversary that offers insurance to another party that compensates them in case they are punished, thus persuading the party to send over their secret key share. As soon as the arbiter is set up, this party could self-punish and claim the insurance, together with any snitching reward that the protocol might offer.

We construct a new (n,n)-SSSS-scheme $\Pi' = (\mathsf{J}', \mathsf{D}', \mathsf{P}'_1, \ldots, \mathsf{P}'_n)$ from Π as follows: Every party receives as auxiliary private input a secret key sk_i as well as common public input $\mathsf{pub} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ which are randomly sampled from a hard relation $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{R}$. Assume that in the punishment phase of Π the Judge algorithm is executed with $\mathsf{J}(1^\kappa, \mathsf{st}_\mathsf{J}, \omega)$ where ω is the witness supplied by the dealer in the punishment phase and st_J is the state of the Judge after the sharing phase.

Informally, the snitching proof for Π' will be a NIZKPoK for the following statement.

I know a proof π that will be accepted by J
or
I know the secret key sk_i corresponding to pk_i.

More formally, let PS be a NIZKPoK proof system for the relation $\mathcal{R}$, defined as follows. The statements of $\mathcal{R}$ are all $\chi := (1^\kappa, \mathsf{pub}, \mathsf{st}_\mathsf{J}, i)$ where 1^κ is the security parameter, pub is the common public input, st_J is the state of the judge J after the sharing phase, and i is the index of the punished party. A statement-witness pair $(\chi, \hat{\omega})$ is in $\mathcal{R}$ if and only if the following statement holds:

$$(\mathsf{J}(1^\kappa, \mathsf{st}_\mathsf{J}, i, \hat{\omega}) = (\mathsf{punish}, \mathsf{P}_i)) \vee ((\mathsf{pk}_i, \hat{\omega}) \in \mathsf{R})$$

We define the Judge algorithm J' of Π' as follows: given input $(1^\kappa, \mathsf{pub}, \mathsf{st}_\mathsf{J}, i, \pi)$ where π is a proof for the statement described above, J' outputs $(\mathsf{punish}, \mathsf{P}_i)$, if π verifies. During the sharing phase, J' behaves as J and additionally receives the public input $\mathsf{pub} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$.

The new scheme Π' is clearly still correct. Further, adversaries cannot exploit the self-punishment mechanism to break unframability, because they would either need to invert the hard relation R to learn an honest party's secret key sk_i or break the knowledge-soundness of PS. The key reason why Π' achieves strong punishability is that the arbiter cannot distinguish between a real punishment and a self-punishment. At a high level, the argument goes as follows: Because of the weak punishability of Π there exists a snitcher $\mathcal{B}$ that can trigger a punishment in step 7a of $\mathsf{Snitching}$. If this punishment of a party $\mathsf{P} \in \mathcal{C}_{\mathcal{A}_j}$ leads to a compensation for P by Arb with non-negligible probability, then one could construct another $\mathcal{B}'$ that behaves exactly the same, but self-punishes on behalf of the same party P. Crucially, the view of Arb between both scenarios is computationally indistinguishable due to the zero-knowledge property of PS (i.e. Arb can not learn from a proof π, which branch of the OR-statement was satisfied). Hence, the new snitcher extorts Arb by triggering a compensation with non-negligible probability and, in the end, the snitcher wins with a negligibly close probability.

Construction of SSSS. The problem with the transformation described above is that our SSS scheme requires an *interactive* judge in the punishment phase. Recall that in our SSS protocol (cf. Section 4), the judge J first receives the subset $\mathcal{W}$ of indices from the dealer. The judge stores $\mathcal{W}$ and then all parties are required to interact to decommit their shares of $\mathcal{U}$, which allows the judge to check whether $\mathcal{W} \subseteq \mathcal{U}$. Denote with D the set of all parties' decommitments. It is not clear how to design a proof system for this interactive phase, as clearly the set $\mathcal{W}$ can be used by the arbiter to determine that this is indeed a self-punishment. At the same time, the dealer cannot prove that he knows some $\mathcal{W} \subseteq \mathcal{U}$ without knowing $\mathcal{U}$ and we cannot ask the dealer to submit a proof after parties decommit $\mathcal{U}$, because then the dealer can just choose some arbitrary subset of $\mathcal{U}$.

We solve this issue using cryptographic commitments: If a snitcher and a dealer want to snitch on some party P_i, they compute a commitment $c := \mathsf{commit}(\mathcal{W}; r)$ to the subset $\mathcal{W}$ and send $(\mathsf{snitch}, \mathsf{P}_i, c)$ to J'. Then, all parties required by the judge to provide their inputs do so (in our case, they decommit their shares of $\mathcal{U}$). Finally, we modify PS to prove the statement:

$$(\mathsf{J}(1^\kappa, \mathsf{st}_\mathsf{J}, D, i, \mathsf{open}(c, \hat{\omega})) = (\mathsf{punish}, \mathsf{P}_i)) \vee ((\mathsf{pk}_i, \hat{\omega}) \in \mathsf{R}),$$

where D are the decommitments to the shares of $\mathcal{U}$ by all parties, $\mathsf{open}(c, \hat{\omega})$ is the decommitment of the commitment c and st_J now contains $\mathcal{U}$.[19] If the snitcher wants to self-punish, then they can simply commit

[19] In the left branch, $\hat{\omega} = (\mathcal{W}, r)$ and in the right branch $\hat{\omega} = \mathsf{sk}_i$.

The punishment phase $\Pi'.\mathsf{punish}$ of the (n,n)-SSSS

1. For a **real punishment**:
   (a) The snitchers in $\mathcal{B}$ send a message $(\mathsf{snitch}, \mathsf{P}, \mathsf{Trace})$ to the dealer D' (here Trace is a set of values, see the page 15 for an explanation).
   (b) he dealer D' computes the set $\mathcal{W} = \{w \mid Y_w \in \mathsf{Trace}\}$. If $|\mathcal{W}| \leq \lfloor \kappa/8 \rfloor$, then D' aborts.
   (c) The dealer samples $r \leftarrow\!\!\$\ \{0,1\}^\kappa$ and computes the commitment $c := \mathsf{commit}(\mathcal{W}; r)$ to the set $\mathcal{W}$.
   For a **self-punishment**:
   (a) The snitchers in $\mathcal{B}$ send a message $(\mathsf{self\text{-}snitch}, \mathsf{P}, \mathsf{sk}_\mathsf{P})$ to the dealer D'.
   (b) The dealer D' aborts, if $(\mathsf{pk}_\mathsf{P}, \mathsf{sk}_\mathsf{P}) \notin \mathsf{R}$.
   (c) The dealer samples $r \leftarrow\!\!\$\ \{0,1\}^\kappa$ and computes the commitment $c := \mathsf{commit}(0; r)$ to a constant $\perp$.
2. Then D' sends $(\mathsf{snitch}, \mathsf{P}, c)$ to the judge J'.
3. Each shareholder P_i sends their decommitment (U_i, R_i) to the judge. Denote with D the set of all decommitments.
4. If for some $i \in [n]$ one of the following conditions happened:
   (a) P_i did not send the decommitment in the previous step;
   (b) P_i sent an invalid decommitment, i.e., $\mathsf{commit}(U_i; R_i) \neq \Gamma_i$;
   then the judge J' outputs $(\mathsf{punish}, \mathsf{P}_i)$ and terminates the protocol.
5. Let $\mathcal{U} := \mathsf{rec}_{n,n}(U_1, \ldots, U_n)$.
6. For a real punishment, the dealer D' sets the witness to $\hat{\omega} := (\mathcal{W}, r)$. For a self-punishment, the dealer sets the witness to $\hat{\omega} := \mathsf{sk}_\mathsf{P}$.
7. Let the statement $\chi := (1^\kappa, \mathsf{pub}, \mathsf{st}_\mathsf{J}, D, \mathsf{P})$.
8. The dealer D' computes a prove π as $\mathsf{PS.Prove}(\chi, \hat{\omega})$ and sends π to the judge.
9. If $\mathsf{PS.Verify}(\chi, \pi) = 1$, then the judge outputs $(\mathsf{punish}, \mathsf{P})$.

Fig. 8: The punishment phase of our n-out-of-n SSSS scheme Π'. Note that the dealer D' as well as the set of snitchers are formally controlled by $\mathcal{B}$. The proof system PS is defined over relation $\mathcal{R}$ as introduced in Section 7 and R is the hard relation of public and secret keys sampled in the setup phase. The parameter $\mathsf{pub}\colon (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ refers to the public keys of the parties from the setup phase. We assume that both the constant $\perp$ and the set $\mathcal{W}$ follow an appropriate encoding such that the commitments are not trivially distinguishable by length. Finally, we abuse notation slightly and use the party P interchangeably with its index.

to a constant $\perp$ and fulfill the right branch of the OR-statement. We denote the scheme described above as Π'. A detailed overview of the punishment phase is depicted in Fig. 8. The sharing and reconstruction phases are identical to those of the weak scheme Π (Figs. 4 and 6).

Intuitively, the commitment trick does not break unframability because of the binding property of the commitment scheme (i.e., one can not decide on $\mathcal{W}$ *after* learning $\mathcal{U}$ from the other parties' decommitments). Further, the hiding property guarantees that c does not leak whether it is a real punishment (i.e., c is a commitment to a correct $\mathcal{W}$) or a self-punishment (i.e. c is a commitment to a random $\mathcal{W}$).

Theorem 3. *Suppose that (1) the relation* R *is a hard relation, (2) the proof system* PS *is correct, knowledge-sound and zero-knowledge, (3) the commitment scheme is hiding and perfectly binding and (4) the SSS-scheme* Π *as constructed in Section 4 is a* (n,n)*-SSS-scheme with weak punishability and slow-query budget* σ*. Then the protocol* $\Pi' = (\mathsf{J}', \mathsf{D}', \mathsf{P}'_1, \ldots, \mathsf{P}'_n)$ *is a* (n,n)*-SSSS-scheme with slow-query budget* σ*.*

Remark 2. Note that the transform *cannot* be instantiated with the weakly punishable scheme of [18], as proving knowledge of the snitching proof that would be accepted by the judge would then require computing NIZK proofs over a statement that involve random oracle evaluations. Even when disregarding the theoretical limitation of NIZK proofs over random oracles, we highlight that practical instantiations will need to instantiate the random oracle with a function that is hard to compute in MPC. It is to be expected that

MPC-hardness and the ability to efficiently evaluate in zero-knowledge are contradictory design constraints for hash functions.

In order to prove Theorem 3 we need to show that Π' is correct, unframable, and achieves strong punishability. First, the correctness of Π' follows directly from the correctness of Π. We move on to show unframability and strong punishability.

Unframability. Unframability of Π' follows from the unframability of Π, the knowledge-soundness of PS, the hardness of R and the perfect binding property of the commitment scheme.

Proof. Let $\mathcal{M}$ be a monolithic adversary that successfully breaks unframability of Π'. We make a series of game hops to show that $\mathcal{M}$ can be used to break the unframability of Π. Let Exp^0 be the unframability experiment of protocol Π' (Fig. 2).

$\underline{\mathsf{Exp}^1}$: In this experiment, we make a change to the punishment phase of Π'. We require the dealer to directly send the witness $\hat{\omega}$ to the judge J' along with the proof π in PS. The experiment aborts if $(\chi, \hat{\omega}) \notin \mathcal{R}$.

Lemma 6. *If* PS *is knowledge-sound (cf. Definition 1), then for all PPT adversaries* $\mathcal{A}$ *there exists a PPT adversary* $\mathcal{B}$ *and a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^1_{\mathcal{B}}(1^\kappa) = 1\right] \geq \Pr\left[\mathsf{Exp}^0_{\mathcal{A}}(1^\kappa) = 1\right] - \mathsf{negl}(\kappa).$$

Proof. The lemma follows directly from the knowledge-soundness of the proof system PS. □

$\underline{\mathsf{Exp}^2}$: In the next experiment, we add yet another abort condition to witness $\hat{\omega}$. We restrict $\hat{\omega}$ to be a witness for the left branch of the statement χ. Exp^2 aborts, if $(\mathsf{pk}_i, \hat{\omega}) \in \mathsf{R}$.

Lemma 7. *If* R *is a hard relation, then* Exp^1 *and* Exp^2 *are computationally indistinguishable.*

Proof. Assume there exists a successful PPT distinguisher $\mathcal{D}$ between Exp^1 and Exp^2 with non-negligible probability ε. In order to distinguish between the two experiments, $\mathcal{D}$ would have to trigger the additional abort condition in Exp^2 for a statement $\chi = (1^\kappa, \mathsf{pub}, \mathsf{st}_\mathsf{J}, i)$ where $\mathsf{P}_i \notin \mathcal{C}_\mathcal{D}$. We construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{D}$ to break the hardness of R. Given input $(1^\kappa, \mathsf{pk})$, $\mathcal{B}$ simulates Exp^2 to $\mathcal{D}$, injecting pk_i for a random i such that $\mathsf{P}_i \notin \mathcal{C}_\mathcal{D}$. If $\mathcal{D}$ initiates snitching against P_i that triggers the additional abort condition $(\mathsf{pk}, \hat{\omega}) \in \mathsf{R}$, $\mathcal{B}$ outputs $\hat{\omega}$. Clearly, $\mathcal{B}$ breaks the hardness of R with non-negligible probability of at least $\frac{\varepsilon}{n}$. □

$\underline{\mathsf{Exp}^3}$: In the last experiment, we extract the message x from the adversary's commitment c. Further, parse $\hat{\omega} := (\mathcal{W}, r)$. We abort if $x \neq \mathcal{W}$.

Lemma 8. *For all adversaries the success probability in* Exp^2 *and* Exp^3 *is identical.*

Proof. The lemma follows directly from the extractability and perfect binding property of the commitment scheme. To see this, note that both experiments abort, if $\mathsf{open}(c, \mathcal{W}, r) \neq x$. □

Let $\mathcal{M}$ be an arbitrary PPT adversary with $\Pr\left[\mathsf{Unframability}^{\Pi'}_{\mathcal{M}}(1^\kappa)\right] \geq p(\kappa)$ for non-negligible p. Following the lemmas above there exists a PPT adversary $\mathcal{M}_3$ with $\Pr\left[\mathsf{Exp}^{3,\Pi'}_{\mathcal{M}_3}(1^\kappa)\right] \geq p(\kappa)/n - \mathsf{negl}(\kappa)$.

We close the proof by showing that a successful adversary $\mathcal{M}_3$ against Exp^3 of Π' can be used to break the unframability of Π. We construct a PPT reduction $\mathcal{B}$ that uses $\mathcal{M}_3$ to break the unframability of Π. $\mathcal{B}$ simulates Exp^3 perfectly to $\mathcal{M}_3$ by forwarding all messages between $\mathcal{M}_3$ and the unframability experiment of Π. When $\mathcal{B}$ receives $\mathcal{W}$ in the punishment phase, it forwards $\mathcal{W}$ to the judge in the unframability experiment of Π.

Clearly, $\mathcal{B}$ breaks unframability whenever $\mathcal{M}_3$ wins in Exp^3. Considering the lemmas above, we can bound $\mathcal{B}$'s success probability as follows:

$$\Pr[\mathsf{Unframability}^{\Pi}_{\mathcal{B}}(1^\kappa)] \geq \Pr\left[\mathsf{Exp}^{3,\Pi'}_{\mathcal{M}_3}(1^\kappa)\right] \geq \frac{p(\kappa)}{n} - \mathsf{negl}(\kappa)$$

which is non-negligible. □

Strong Punishability. Our strong punishability proof makes use of the weak punishability of Π as well as the correctness and hiding of the commitment to the set $\mathcal{W}$ and the correctness and zero-knowledge properties of PS.

Proof. To prove strong punishability of Π', we first show that our changes maintain weak punishability. The proof carries on where the weak punishability proof of Π left off. Because of the perfect correctness of the commitment scheme and proof system PS, clearly any snitcher that can make J output $(\mathsf{punish}, \mathsf{P}_i)$ in Π can also make J' output $(\mathsf{punish}, \mathsf{P}_i)$ in Π'. Where $\mathcal{B}$ would have sent a set $\mathcal{W}$ to J in Π, it now sends a commitment c to $\mathcal{W}$ to J' in Π', followed by a proof π that the commitment contains $\mathcal{W}$ such that J would have output $(\mathsf{punish}, \mathsf{P}_i)$.

Hence, for all distributed adversaries $\mathcal{A}$ there exists a snitching strategy $\mathcal{B}$ such that

$$\Pr[\exists j \in [a] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi', \bot, \mathcal{A}, j, 0); \kappa]\!] = 1] \geq \mathsf{Adv}[\![(\bot, \mathcal{A}) \leftrightarrows \Pi'; \kappa]\!] - \mathsf{negl}(\kappa).$$

To carry this argument over to strong punishability, we observe what happens if the same $\mathcal{B}_1 = \mathcal{B}$ now runs in $\mathsf{Snitching}$ *with an arbitrary arbiter* Arb. Let p_{comp} be the probability that a party P gets punished and Arb compensates P in step 7a of $\mathsf{Snitching}[\![\mathsf{S}, \mathcal{B}_1 \leftrightarrows (\Pi', \mathsf{Arb}, \mathcal{A}, j, 0); \kappa]\!]$. We make the following case distinction:

Case 1: p_{comp} is negligible. In this case, $\mathcal{B}_1$ already breaks the strong punishability of Π and the proof is done.

Case 2: p_{comp} is non-negligible. In this case, we consider a snitching strategy $\mathcal{B}_2$ running in the $\mathsf{Snitching}$ experiment of Π' on input (j', i) for an arbitrary $j' \neq j$, such that now $\mathsf{P} \notin \mathcal{C}(\mathcal{A}_j)$ and $\mathsf{P}_i = \mathsf{P}$. The strategy $\mathcal{B}_2$ controls all the subadversaries except $\mathcal{A}_{j'}$ and behaves exactly the same as the subadversaries up to the punishment phase. In the punishment phase, $\mathcal{B}_2$ issues a self-punishment on behalf of $\mathsf{P}_i = \mathsf{P}$ as specified by Π'. In the following, we will argue that, due to the zero-knowledge property of the proof system PS and the hiding property of the commitment scheme, the view of the arbiter is computationally indistinguishable between both experiments, hence the arbiter will output $(\mathsf{comp}, \mathsf{P})$ with probability close to p_{comp}, leading the experiment to output 2 in step 7b.

Let p'_{comp} be the probability that Arb outputs $(\mathsf{comp}, \mathsf{P})$ in this experiment. Observe that in the original experiment, the view of Arb consists of the tuple $\mathsf{view}_{\mathsf{Arb}} := (\mathsf{st}_{\mathsf{J}}, c, \pi, \mathsf{P}, D, \mathsf{msgs})$, where st_{J} denotes the state of the judge after the sharing phase, c is the commitment to $\mathcal{W}$, π is the proof in PS that a valid $\mathcal{W}$ is contained in the commitment (i.e. left branch of the OR-proof), P is the fixed punished party, D is the set of decommitments by all parties and msgs are the messages received by the arbiter from all subadversaries. In the new experiment, the view of Arb consists of $\mathsf{view}'_{\mathsf{Arb}} := (\mathsf{st}_{\mathsf{J}'}, c', \pi', \mathsf{P}', D', \mathsf{msgs}')$, where c' is a commitment to sk_{P} and π' is a proof that sk_{P} is contained in the commitment such that $(\mathsf{pk}_{\mathsf{P}}, \mathsf{sk}_{\mathsf{P}}) \in \mathsf{R}$ in the proof system PS (right branch of the OR-proof), D' are the decommitments by the parties, and msgs' are the messages received by the arbiter.[20]

Next, we show through a series of hybrids that $\mathsf{view}_{\mathsf{Arb}} \overset{c}{\approx} \mathsf{view}'_{\mathsf{Arb}}$. First, observe that $\mathsf{st}_{\mathsf{J}} \equiv \mathsf{st}_{\mathsf{J}'}$, as the experiments are identical up to the punishment phase. We introduce a hybrid $\mathcal{H}_1$, where we replace the proofs π or π' respectively with a simulated proof π_{sim}. Due to the zero-knowledge property of PS, it holds that both $\mathsf{view}_{\mathsf{Arb}} \overset{c}{\approx} \mathsf{view}_{\mathsf{Arb},\mathcal{H}_1}$ and $\mathsf{view}'_{\mathsf{Arb}} \overset{c}{\approx} \mathsf{view}'_{\mathsf{Arb},\mathcal{H}_1}$.

In the second hybrid $\mathcal{H}_2$, we replace the commitments c or c' with a commitment to an arbitrary constant $\bot$. Due to the hiding property of the commitment scheme, it holds that both $\mathsf{view}_{\mathsf{Arb},\mathcal{H}_1} \overset{c}{\approx} \mathsf{view}_{\mathsf{Arb},\mathcal{H}_2}$ and $\mathsf{view}'_{\mathsf{Arb},\mathcal{H}_1} \overset{c}{\approx} \mathsf{view}'_{\mathsf{Arb},\mathcal{H}_2}$.

Let us now compare $\mathsf{view}_{\mathsf{Arb},\mathcal{H}_2}$ with $\mathsf{view}'_{\mathsf{Arb},\mathcal{H}_2}$. Clearly, it holds that $c_{\mathcal{H}_2} \equiv c'_{\mathcal{H}_2}$ and $\pi_{\mathcal{H}_2} \equiv \pi'_{\mathcal{H}_2}$. As a consequence, it must hold that $D_{\mathcal{H}_2} \equiv D'_{\mathcal{H}_2}$, $\mathsf{msgs}_{\mathcal{H}_2} \equiv \mathsf{msgs}'_{\mathcal{H}_2}$, and $\mathsf{P}_{\mathcal{H}_2} \equiv \mathsf{P}'_{\mathcal{H}_2}$, and thus $\mathsf{view}_{\mathsf{Arb},\mathcal{H}_2} \equiv \mathsf{view}'_{\mathsf{Arb},\mathcal{H}_2}$.

[20] Note that we abuse notation slightly and treat the elements of $\mathsf{view}_{\mathsf{Arb}}$ and $\mathsf{view}'_{\mathsf{Arb}}$ as random variables.

By transitivity of $\overset{c}{\approx}$ and $\equiv$, we conclude that $\mathsf{view}_{\mathsf{Arb}} \overset{c}{\approx} \mathsf{view}'_{\mathsf{Arb}}$. As a result, the arbiter outputs $(\mathsf{comp}, \mathsf{P})$, when running against $\mathcal{B}_2$ with probability $p'_{\mathsf{comp}} \geq p_{\mathsf{comp}} - \mathsf{negl}(\kappa)$ for a negligible function negl and $\mathcal{B}_2$ triggers step 7b of $\mathsf{Snitching}$ with non-negligible probability p'_{comp} leading the experiment to output 2 and

$$\begin{aligned}
&\Pr[\exists j \in [a] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B}_1 \leftrightarrows (\Pi', \mathsf{Arb}, \mathcal{A}, j, 0); \kappa]\!] = 1] + \\
&\Pr[\exists i \in [n] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B}_2 \leftrightarrows (\Pi', \mathsf{Arb}, \mathcal{A}, 0, i); \kappa]\!] = 2] \\
&\geq (\Pr[\exists j \in [a] \colon \mathsf{Snitching}[\![\mathsf{S}, \mathcal{B} \leftrightarrows (\Pi, \bot, \mathcal{A}, j, 0); \kappa]\!] = 1] - p_{\mathsf{comp}}) + p'_{\mathsf{comp}} \\
&\geq \mathsf{Adv}[\![(\mathsf{Arb}, \mathcal{A}) \leftrightarrows \Pi'; \kappa]\!] - \mathsf{negl}(\kappa).
\end{aligned}$$

□

8 The threshold t-out-of-n SSSS scheme

The threshold SSS construction of [18] uses black-box a transformation from any $(2,2)$-SSS scheme to a (t,n)-SSS scheme, yet with a non-black box proof. In this section, we show that if we apply this transformation to our $(2,2)$-SSS scheme, then the resulting (t,n)-SSS scheme can be adapted to (t,n)-SSSS by applying the ideas from Section 7.

First, we note that the weak punishability model in [18] only admits snitching strategies $\mathcal{B}$ that behave *semi-honestly* or *passive* with respect to the recorrupted subadversaries of the original adversarial strategy $\mathcal{A}$, as snitchers merely pass over their trace of queries to Ω_H to the dealer. In this paper, we work with a more general model where the snitcher and the dealer can actively collaborate to snitch on colluders. We do this to model the possibility for snitchers to self-punish, as described in Section 7 for our (n,n)-SSSS construction. Observe though that our (n,n)-SSS construction follows the model from [18], as the snitcher only sends its trace to the dealer. Hence, we can apply the threshold transformation as described in [18].

Threshold transformation. The threshold transformation is similar to the one of [18], and on a high level, works as follows. First, the dealer samples a random secret R and (t,n)-Shamir shares R into $R_1, \ldots, R_n$. Then, the dealer performs a second round of (t,n)-Shamir sharing, where he shares each R_i into $R_i^1, \ldots, R_i^n$. This results in an $n \times n$ matrix of Shamir shares. The dealer proceeds to run $n^2 - n$ instances of the $(2,2)$-SSS scheme between the parties P_i and P_j, sharing the secret R_i^j for all $i \neq j \in [n]$. The Judge J runs all the judges of the separate instances as subroutines J_i^j. If all instances finish the sharing phase without errors, the dealer encrypts the secret S as $C = G(R) \oplus S$ for some PRG G and sends C to all parties. To reconstruct S, the set of t parties needs to recover their shares R_i of R. To recover R_i, the parties need to run the reconstruction protocol in at least t of the $(2,2)$ instances that $(2,2)$-SSS share the shares of R_i. The construction in [18] allows punishing up to $t-1$ colluders. To prepare for strong punishability, we restrict ourselves to only punish *one* colluder. Let Π be the (t,n)-SSS scheme obtained by applying the threshold transformation to our (n,n)-SSS scheme from Section 4.

Theorem 4. *If the underlying (n,n)-SSS scheme with weak punishability is secure with slow-query budget σ, then the (t,n)-SSS scheme Π is secure with weak punishability and slow-query budget $t \cdot \sigma$.*

The proof is almost identical to the one from [18]. For completeness, we provide it in the following sections.

8.1 Correctness

Proof. We assume that all the inputs of the reconstruction algorithm of the t-out-of-n SSS scheme were computed as in the sharing algorithm. What is more, all the values S_i^j in the reconstruction algorithm are reconstructed to values S_i^j from the sharing algorithm with overwhelming probability, due to the correctness property of the $\Pi.\mathsf{share}_{\oplus}$ algorithm, the fact that $\kappa \geq 2\log_2(\sigma/t)$ and the union bound. The other steps of the reconstruction algorithm are deterministically reverting the process of the sharing algorithm, thus correctness holds. □

8.2 Punishability

Proof. Let $\Pi_{\oplus}$ be the underlying $(2,2)$-SSS protocol. Fix a distributed adversary $\mathcal{A}$ and assume that each coalition consists of less than t parties. We need to show that:

$$\mathsf{Adv}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi;\kappa]\!] - \mathsf{negl}(\kappa) \leq \Pr[\exists \ell\colon \mathsf{Snitching}[\![\mathsf{S},\mathcal{B} \leftrightarrows (\Pi,\bot,\mathcal{A},\ell);\kappa]\!] = 1].$$

First, consider the $\mathsf{Sharing}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi, b;\kappa]\!]$ experiment. Let Broken denote the event that the adversary submitted a query of the form "k" $||$ R for some $k \in \mathbb{N}$ to Ω_H (i.e., one of the queries necessary to compute $G(R)$). It is easy to see that we have: $\mathsf{Adv}[\![(\bot,\mathcal{A}) \leftrightarrows \Pi;\kappa]\!] \leq \Pr[\mathsf{Broken}]$. Similar to what we did in the proof of Theorem 2 with overwhelming probability none of the queries done by the adversary in the precomputation phase is a scratch, and there are no collisions between the values of Y_j used in the instances of $\Pi_{\oplus}$. From now on, let us assume that none of these unlikely events have happened.

Let

$$\mathsf{Broken}_{\oplus} := \left\{N_{\mathsf{succ}} \geq \lfloor \kappa/2 \rfloor, \overline{\mathsf{Error}}\right\},$$

let Broken_i^j be the event that $\mathsf{Broken}_{\oplus}$ occurred in the $(\mathsf{P}_i,\mathsf{P}_j)$-instance of $\Pi_{\oplus}$ and let Fast_i^j be the event that there were at least $\kappa/4$ fast scratches in that instance. Furthermore, let Broken_i be the event that Broken_i^j occurred for i,j such that P_i and P_j do not share a coalition. Finally, let $\mathsf{Broken}(t)$ be the event that Broken_i occurred for at least t distinct values of i. Since the size of each coalition is strictly less than t and the secret sharing scheme is information-theoretically hiding, we obtain that $\Pr[\mathsf{Broken}] \leq \Pr[\mathsf{Broken}(t)]$.

Note that the punishability of the 2-out-of-2 sharing scheme, as discussed in Section 5.2, still holds even if the adversary controls some parties who are not shareholders but might participate in a reconstruction attempt. Since we have assumed that there are no collisions among the Y_j, the sets of scratches made in different instances of the $\Pi_{\oplus}.\mathsf{share}$ procedure also disjoint. Since the slow query budget of the adversary is σ, it follows that in at least one instance, the adversary made at most σ/t slow queries, and, consequently, made at least one scratch query while reconstructing S_i^j, i.e.,

$$\Pr[\mathsf{Broken}(t)] \leq \Pr\left[\exists i \exists j\colon \mathsf{Fast}_i^j\right] + \mathsf{negl}(\kappa).$$

It is easy to see that the above still holds in the $\mathsf{Snitching}[\![\mathsf{S},\mathcal{B} \leftrightarrows (\Pi,\bot,\mathcal{A},b);\kappa]\!]$ experiment, and that Fast_i^j implies that either of the parties P_i and P_j is punishable, as argued in Section 5.2. □

8.3 Unframability

Proof. Let $\Pi_{\oplus}$ be the underlying $(2,2)$-SSS protocol. Fix an honest party P_{i^*}. Suppose that a monolithic adversary $\mathcal{M}^0$ corrupts the dealer D and some shareholders $\mathsf{P}_{j_1},\ldots,\mathsf{P}_{j_m}$ different from P_{i^*}. The punishment algorithm of the t-out-of-n SSS scheme runs the punishment subprocedure of the underlying $\Pi_{\oplus}$ scheme. Notice that, with overwhelming probability, all the values Y_j in the instances of $\Pi_{\oplus}$ are distinct, so we might only consider the instances of $\Pi_{\oplus}$ in which P_{i^*} participates. For any instance in which P_{i^*} participates, the party will not be framed due to the unframability of the $\Pi_{\oplus}$ scheme and the claim follows from the union bound. □

Achieving strong punishability. Given the (t,n)-SSS scheme Π, we show how to adapt it to a (t,n)-SSSS scheme Π' with the same technique as in Section 7.

Denote Π_i^j as the instance between i and j. Observe that the punishment phase of Π is essentially restricted to *one* $(2,2)$ instance Π_i^j in the subroutine J_i^j of J' for some $i \neq j$. We apply the same changes as in Section 7 to each instance Π_i^j. In particular, we sample separate secret keys and public keys for each instance in the setup_i^j phase. We also make the same modification to the punishment phase. We modify the proof system PS from Section 7 such that, to punish party P_i, the dealer needs to prove knowledge of a valid witness *for any instance* Π_i^j with $j \in [n] \setminus \{i\}$. First the dealer sends a commitment c to $\mathcal{W}_i^j$ if they want to perform a real punishment or a commitment to sk_i^j if they want to self-punish on behalf of P_i for some

$j \in [n] \setminus \{i\}$, respectively. Once all parties have opened their commitments to their shares of all $\mathcal{U}_i^*$ and the judge has reconstructed all $\mathcal{U}_i^*$, the dealer sends a proof in PS that he either knows a decommitment $\mathcal{W}_i^j$ to c such that $\mathcal{W}_i^j$ is a large enough subset of one $\{\mathcal{U}_i^j\}_{j\in[n]\setminus\{i\}}$ or that he knows any secret key sk_i^j of the party P_i.

Theorem 5. *Suppose that (1) the relation* R *is a hard relation, (2) the proof system* PS *is correct, knowledge-sound and zero-knowledge, (3) the commitment scheme is hiding and perfectly binding and (4)* Π *is a* (t, n)*-SSS-scheme with weak punishability as constructed above with slow-query budget* σ*. Then the protocol* $\Pi' = (\mathsf{J}', \mathsf{D}', \mathsf{P}'_1, \ldots, \mathsf{P}'_n)$ *is a secure* (t, n)*-SSSS-scheme with slow-query budget* σ*.*

Correctness and unframability can be argued identically to Section 7. The strong punishability proof is very similar to the proof of strong punishability in Section 7. Assume that punishment happens in instances Π_i^*. The only difference is that the "irrelevant" instances $\Pi_{i'}^{j'}$ for $i' \neq i$ are also part of the view of the arbiter Arb. It is easy to see though that Arb's view of all instances $\Pi_{i'}^{j'}$ are even *identically* distributed between the executions of $\mathsf{Snitching}$ with $\mathcal{B}_1$ and $\mathcal{B}_2$. Hence, $\mathcal{B}_2$ still has the same non-negligible probability of exploiting the arbiter in Case 2 of the strong punishability proof in Section 7.

9 Open problems

Mutually assured destruction. Our construction protects against insurance-type attacks, which are not subject to griefing and involve no risk on to the party being compensated. One attack that the adversary could attempt is a "Mutually Assured Destruction" (MAD) attack, i.e., letting the arbiter slash *all* adversarial parties whenever a punishment is output by the judge. Our construction does not protect against such attacks, and we leave the construction of a scheme that would be secure against such attacks as an open problem. While MAD attacks are generally less powerful, as any colluder could deviate from the collusion strategy to grief all colluders (and cause everyone to get slashed), we still consider it an interesting open problem to formally model them and design collusion countermeasures that hold up against MAD attacks.

Non-interactive punishment. In our scheme, the punishment phase is interactive, as all shareholders are forced to send decommitments (U_i, R_i). Ideally, we would like to have a (strong) SSS scheme with a non-interactive punishment phase.

Primitives beyond secret sharing. In order to prevent colluders from secretly reconstructing the secret in MPC, we have designed a secret sharing scheme with an "MPC-hard" (non-linear) reconstruction algorithm. This, however, is detrimental, whenever we want to extrapolate the powerful anti-collusion guarantees provided by SSS to more advanced primitives in distributed cryptography like threshold decryption [16, 34] or threshold signatures [35, 2, 15], because constructions usually rely on the linearity of the underlying secret sharing scheme. As a result, it seems infeasible to construct collusion-resilient primitives by merely replacing the underlying secret sharing scheme with some SSSS scheme. There seems to be an inherent incompatibility between efficient distributed computation as required by advanced threshold primitives and the need for MPC-hard reconstruction as required by SSSS. Hence, we believe that designing advanced threshold primitives with SSSS-level anti-collusion guarantees is an important direction for future research.

Game-theoretic modeling. Finally, a formal game theoretic analysis of our scheme, which is considered out of scope for this paper, would definitely be of interest to the cryptographic community.

Acknowledgments. This work has been partially funded by the German Research Foundation (DFG) CRC 1119 CROSSING (project S7), by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant CRYPTOLAYER-101044770), and by the European Research Council under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

References

[1] Alwen, J., Shelat, A., and Visconti, I. "Collusion-Free Protocols in the Mediated Model". In: *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*. Ed. by D. A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 497–514. DOI: 10.1007/978-3-540-85174-5_28.

[2] Bacho, R. and Loss, J. "On the adaptive security of the threshold BLS signature scheme". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 193–207.

[3] Bellare, M. and Rogaway, P. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by D. E. Denning et al. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[4] Ben-Or, M., Goldwasser, S., and Wigderson, A. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by J. Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[5] Boneh, D. and Franklin, M. K. "An Efficient Public Key Traitor Tracing Scheme". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 338–353. DOI: 10.1007/3-540-48405-1_22.

[6] Boneh, D. and Naor, M. "Traitor tracing with constant size ciphertext". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. CCS '08. Alexandria, Virginia, USA: Association for Computing Machinery, 2008, pp. 501–510. ISBN: 9781595938107. DOI: 10.1145/1455770.1455834.

[7] Boneh, D. and Naor, M. "Traitor tracing with constant size ciphertext". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. CCS '08. Alexandria, Virginia, USA: Association for Computing Machinery, 2008, pp. 501–510. ISBN: 9781595938107. DOI: 10.1145/1455770.1455834.

[8] Boneh, D., Partap, A., and Rotem, L. "Accountability for Misbehavior in Threshold Decryption via Threshold Traitor Tracing". In: *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII*. Ed. by L. Reyzin and D. Stebila. Vol. 14926. Lecture Notes in Computer Science. Springer, 2024, pp. 317–351. DOI: 10.1007/978-3-031-68394-7_11.

[9] Boneh, D., Partap, A., and Rotem, L. "Traceable Secret Sharing: Strong Security and Efficient Constructions". In: *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part V*. Ed. by L. Reyzin and D. Stebila. Vol. 14924. Lecture Notes in Computer Science. Springer, 2024, pp. 221–256. DOI: 10.1007/978-3-031-68388-6_9.

[10] Boneh, D., Sahai, A., and Waters, B. "Fully Collusion Resistant Traitor Tracing with Short Ciphertexts and Private Keys". In: *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*. Ed. by S. Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 573–592. DOI: 10.1007/11761679_34.

[11] Boneh, D. and Waters, B. "A fully collusion resistant broadcast, trace, and revoke system". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 211–220. ISBN: 1595935185. DOI: 10.1145/1180405.1180432.

[12] Canetti, R. "Universally Composable Security". In: *J. ACM* 67.5 (2020), 28:1–28:94. DOI: 10.1145/3402457.

[13] Chor, B. et al. "Tracing traitors". In: *IEEE Trans. Inf. Theory* 46.3 (2000), pp. 893–910. DOI: 10.1109/18.841169.

[14] Cleve, R. "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)". In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*. Ed. by J. Hartmanis. ACM, 1986, pp. 364–369. DOI: 10.1145/12130.12168.

[15] Crites, E., Komlo, C., and Maller, M. "Fully adaptive schnorr threshold signatures". In: *Annual International Cryptology Conference*. Springer. 2023, pp. 678–709.

[16] Desmedt, Y. "Threshold cryptosystems". In: *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1992, pp. 1–14.

[17] Dziembowski, S., Faust, S., and Lizurej, T. "Individual Cryptography". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14082. Lecture Notes in Computer Science. Springer, 2023, pp. 547–579. DOI: 10.1007/978-3-031-38545-2_18.

[18] Dziembowski, S. et al. "Secret Sharing with Snitching". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*. Ed. by B. Luo et al. ACM, 2024, pp. 840–853. DOI: 10.1145/3658644.3690296.

[19] Fiat, A. and Tassa, T. "Dynamic Traitor Tracing". In: *J. Cryptol.* 14.3 (2001), pp. 211–223. DOI: 10.1007/S00145-001-0006-7.

[20] Fischer, M. J., Lynch, N. A., and Merritt, M. "Easy Impossibility Proofs for Distributed Consensus Problems". In: *Distributed Comput.* 1.1 (1986), pp. 26–39. DOI: 10.1007/BF01843568.

[21] Goldreich, O. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. ISBN: 0-521-79172-3. DOI: 10.1017/CBO9780511546891.

[22] Gong, T. et al. "More is Merrier: Relax the Non-Collusion Assumption in Multi-Server PIR". In: *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 4348–4366. DOI: 10.1109/SP54263.2024.00095.

[23] Goyal, R., Koppula, V., and Waters, B. "Collusion Resistant Traitor Tracing from Learning with Errors". In: *SIAM J. Comput.* 49.5 (2020). DOI: 10.1137/18M1197825.

[24] Goyal, V., Song, Y., and Srinivasan, A. "Traceable Secret Sharing and Applications". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*. Ed. by T. Malkin and C. Peikert. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 718–747. DOI: 10.1007/978-3-030-84252-9_24.

[25] Katz, J. and Lindell, Y. *Introduction to Modern Cryptography, Third Edition*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2020. ISBN: 9781351133029.

[26] Kurosawa, K. and Desmedt, Y. "Optimum Traitor Tracing and Asymmetric Schemes". In: *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*. Ed. by K. Nyberg. Vol. 1403. Lecture Notes in Computer Science. Springer, 1998, pp. 145–157. DOI: 10.1007/BFB0054123.

[27] Lepinski, M., Micali, S., and Shelat, A. "Collusion-free protocols". In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. Ed. by H. N. Gabow and R. Fagin. ACM, 2005, pp. 543–552. DOI: 10.1145/1060590.1060671.

[28] Mitzenmacher, M. and Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. USA: Cambridge University Press, 2005. ISBN: 0521835402.

[29] Naor, M. and Pinkas, B. "Efficient trace and revoke schemes". In: *Int. J. Inf. Sec.* 9.6 (2010), pp. 411–424. DOI: 10.1007/S10207-010-0121-2.

[30] Naor, M. and Pinkas, B. "Threshold Traitor Tracing". In: *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*. Ed. by H. Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Springer, 1998, pp. 502–517. DOI: 10.1007/BFB0055750.

[31] Nishimaki, R., Wichs, D., and Zhandry, M. "Anonymous Traitor Tracing: How to Embed Arbitrary Information in a Key". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II. Ed. by M. Fischlin and J. Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 388–419. DOI: 10.1007/978-3-662-49896-5_14.

[32] Shamir, A. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176.

[33] Shelat, A. "Collusion-free protocols". In: *Proceedings of the Behavioral and Quantitative Game Theory - Conference on Future Directions, BQGT '10, Newport Beach, California, USA, May 14-16, 2010*. Ed. by M. Dror and G. Sosic. ACM, 2010, 91:1. DOI: 10.1145/1807406.1807497.

[34] Shoup, V. and Gennaro, R. "Securing threshold cryptosystems against chosen ciphertext attack". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1998, pp. 1–16.

[35] Stinson, D. R. and Strobl, R. "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates". In: *Australasian Conference on Information Security and Privacy*. Springer. 2001, pp. 417–434.