Secret Sharing with Snitching

Stefan Dziembowski
University of Warsaw and IDEAS NCBR
Warsaw, Poland

Sebastian Faust
Technische Universität Darmstadt
Darmstadt, Germany

Tomasz Lizurej
University of Warsaw and NASK
Warsaw, Poland

Marcin Mielniczuk
University of Warsaw
Warsaw, Poland

Abstract

We address the problem of detecting and punishing shareholder collusion in secret-sharing schemes. We do it in the recently proposed cryptographic model called *individual cryptography* (Dziembowski, Faust, and Lizurej, Crypto 2023), which assumes that there exist tasks that can be efficiently computed by a single machine but distributing this computation across multiple (mutually distrustful devices) is infeasible.

Within this model, we introduce a novel primitive called *secret sharing with snitching* (SSS), in which each attempt to illegally reconstruct the shared secret S results in a proof that can be used to prove such misbehavior (and, e.g., financially penalize the cheater on a blockchain). This holds in a very strong sense, even if the shareholders attempt not to reconstruct the entire secret S but only learn some partial information about it. Our notion also captures the attacks performed using multiparty computation protocols (MPCs), i.e., those where the malicious shareholders use MPCs to compute partial information on S. The main idea of SSS is that any illegal reconstruction can be proven and punished, which suffices to discourage illegal secret reconstruction. Hence, our SSS scheme effectively prevents shareholders' collusion. We provide a basic definition of threshold (t-out-of-n) SSS. We then show how to construct it for $t = n$, and later, we use this construction to build an SSS scheme for an arbitrary t.

In order to prove the security of our construction, we introduce a generalization of the random oracle model (Bellare, Rogaway, CCS 1993), which allows modelling hash evaluations made inside MPC.

CCS Concepts

• **Security and privacy** → **Cryptography**; **Cryptography**.

Keywords

secret sharing, collusion prevention, front-running prevention

ACM Reference Format:
Stefan Dziembowski, Sebastian Faust, Tomasz Lizurej, and Marcin Mielniczuk. 2024. Secret Sharing with Snitching. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3658644.3690296

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA.

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690296

1 Introduction

Secret sharing [32] is a fundamental primitive in cryptography. It is used when sensitive information must be kept secret, e.g., for storing secret keys, passwords, or confidential documents. Beyond its basic applications, secret sharing has found numerous uses in cryptography, including multiparty computation protocols (MPC) [5, 9, 19, 35] (that allow a group of mutually distrusting parties to compute a function on secret inputs jointly), threshold cryptography [30], private information retrieval [11], and many more. While secret sharing comes in many different flavors, its basic form allows a dealer D to share a secret $S \in \{0,1\}^*$ among shareholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Given their shares, t shareholders (where $t \leq n$ is a parameter called a *threshold*) can recover the secret S by running the *reconstruction* algorithm. The two main properties of a secret sharing scheme (share, rec) are *correctness* and *security*. The former says that any set of t shareholders can reconstruct the shared secret. The latter guarantees that any shareholder set of at most $t-1$ shareholders learns no information about the secret S. Hence, the crucial requirement for the security of secret sharing is that no set of $t-1$ corrupt shareholders collaborates to reconstruct the secret illegally. By "illegally," we mean that shareholders reconstruct the secret (or partial information of it) outside of the situation that was agreed upon with the dealer. For example, if S is a blockchain transaction, then the dealer may want the shareholders to reconstruct it only after some time has passed.

Unfortunately, if t shareholders collude and send their shares to each other, security completely collapses. Even worse, in this case, the dealer D cannot determine that the collusion occurred (unless she has access to an unprotected communication channel between the shareholders), let alone prove this fact to any external judge. For several applications (we describe them later), this poses a significant problem, especially since the dishonest shareholders may systematically exploit the fact that they secretly reconstruct the shared secrets without the dealer noticing it. In this work, we ask the following intriguing question:

Can we design a secret sharing scheme that deters the shareholders from colluding by making such collusion provable to a judge?

We answer the question in the affirmative by introducing a new cryptographic primitive that we call *secret sharing with snitching (SSS)*. On a high level, our primitive achieves collusion-resilience by putting the shareholders in a situation akin to a prisoner's dilemma and exploiting the fact that they may have different economic interests.

A secret sharing with snitching scheme consists of a special procedure that allows one of the corrupt shareholders to "snitch" on the other shareholder and prove to an external judge J that the shareholders performed an illegal reconstruction of a secret S (see below for the explanation of the practical instantiations of the judge). More precisely, we have two new security requirements that we term *punishability* and *unframability*. Punishability guarantees that if a group of at least t shareholders engage in *any* successful illegal secret reconstruction protocol, then at least one of the shareholders can generate a valid proof of punishment for the judge. The unframability property protects any honest shareholder. More precisely, any adversary $\mathcal{M}$ that controls D and all the shareholders, except for some P_i, cannot wrongly accuse P_i in front of the judge. Note that in our solution, we are *not interested* in the case when there is one single adversary entity that controls at least t of the shareholders – indeed, in this case security is impossible as a single adversarial entity knows the secret and does not need to collude with any of the other shareholders.

In this work, we are interested in scenarios where *all* the shareholders can be corrupt. One could be tempted to think of the shareholders controlled by such a snitcher as "honest" (since they are loyally helping the dealer to catch and punish the other adversaries). However, typically, in cryptography, the term "honest party" is reserved to the one that strictly follows the protocol and does nothing more. As a consequence, an honest party can never successfully snitch, as it cannot obtain valid snitching data without illegal reconstruction. Since we needed to capture arbitrary illegal reconstruction strategies we define such a "loyal snitcher" as another adversarial party that, in the end, "betrays" other adversaries.

We believe that secret sharing with snitching has many practical applications. We describe them in Section 1.1.

Instantiating the judge. The role of the judge is to punish the misbehaving shareholders based on the evidence provided by the other parties. The most natural instantiation of the judge is a *smart contract*. Since smart contracts are placed on blockchains, their state is public (and so, assuming their existence does not trivialize the problem of constructing SSS). In our case, the concrete scenario would be as follows. (1) The dealer deploys a judge smart contract on the blockchain (or chooses an existing one), (2) the shareholders deposit coins in it for some time T, (3) once all the shareholders place their deposits, the dealer shares his secret among them. The secret should not be reconstructed until some time $T' \ll T$. If the contract receives a fraud proof against a shareholder P_i before time T', then (4) the coins of P_i are confiscated. Note that this requires a consensus between the parties about which smart contract they use and which blockchain they use. In practice, it is up to the dealer to communicate this information to the shareholders.

Our solution is not limited to blockchain applications. For example, one can apply it to any setting where the shareholders have a reputation (e.g., their reputation gradually grows with time based on their history of honest behavior). In this case, it is sufficient that the parties have access to a broadcast channel (or a "bulletin board"), and all the messages that are "sent to the judge" are just broadcast publicly. Since our judge is a deterministic machine with a public state, every party can locally simulate its actions based on these broadcasted messages and decide if a given shareholder is corrupt or honest. Note that by the properties of the broadcast primitive, all the honest parties will agree on the same corrupt shareholders.

Similarities with the blockchain models. Recall that in the standard Byzantine security of distributed systems (see, e.g., [13]), a "corruption threshold $t-1$" means that at least $n-(t-1)$ parties are always perfectly honest, i.e. they follow the protocol faithfully and leak no information, even if they could do it covertly and even if they could benefit financially from it. While this approach works very well in many settings (e.g. when the parties become malicious only when they are hacked), it is not well-suited for scenarios when the distributed algorithm is run between a group of *anonymous* parties, with no prior reputation. The most notable example of such a setting is the model used in several blockchain projects, starting from Nakamoto's Bitcoin work [28]. In such systems, the parties are called the *miners* or the *validators*. Due to their anonymity, unlike the standard distributed systems, it is unrealistic to assume perfect honesty from any fraction of the parties. More precisely, one typically thinks of a "corruption threshold" t as the upper bound on the size[1] of a malicious coalition of parties. Even though the original Nakamoto's paper did not have formal modeling (for more formal papers on this topic, see, e.g., [16, 18, 31]), it is clear that Bitcoin was designed to be *incentive-compatible* in the sense that at the intuitive level, the parties are incentivized to behave honestly. Hence, it is secure against any *selfish* behavior, or, more precisely, as long as the adversary does not control a very large fraction of the (computing power of) the parties, behaving honestly is the most profitable strategy.

Similar logic is used in the Proof-of-Stake (PoStake) blockchains (see, e.g., [8, 24]). In fact, making such protocols secure against selfish behavior was one of the main reasons why the standard consensus algorithms could not be used directly in this area. Several PoStake blockchains contain mechanisms to discourage selfish behavior. For example, in the Ethereum blockchain [8], the users are penalized for "mining" on multiple competing blockchain branches (the so-called "nothing at stake" attack, see, e.g., [8]). Such a mechanism would not be needed in settings where the majority of the parties are perfectly honest.

While incentivizing *correct* behavior of the selfish parties is often relatively simple (e.g., Ethereum discourages the "nothing at stake" attack by penalizing parties who sign contradictory statements), it is much less clear how to design systems that also incentivize maintaining *secrecy*. Our work addresses this problem.

A straw-man idea. Let us briefly discuss why achieving our notion of secret sharing with snitching is highly non-trivial. To this end, consider the following simple approach. To share S, the dealer D generates shares $S_1, \ldots, S_n$ using the standard Shamir's t-out-of-n secret sharing. It then signs each share S_i with its secret key sk and sends S_i together with the signature σ_i to the corresponding shareholder P_i. A naive idea could now be to define the proof of punishment as a share S_i signed by D (or more generally, one could think of a proof as a log of communication with the party holding S_i). Clearly, if a shareholder $\mathsf{P}_{\widehat{i}}$ (for $\widehat{i} \neq i$) can provide such a proof, then this shows that P_i illegally sent its share to $\mathsf{P}_{\widehat{i}}$. There are, however,

[1] Often measured in terms of computing power (the *Proofs-of-Work* blockchains), or financial resources (the *Proofs-of-Stake* blockchains).

multiple apparent problems with this approach. First, nothing forces P_i to run such a trivial collusion by just sending (S_i, σ_i) to $\mathsf{P}_{\widehat{i}}$. In fact, the same attack works if each P_i strips σ_i off its share and just sends S_i to the other colluding parties. One might hope this attack applies to our particular (naive) sharing scheme. Unfortunately, this is not the case. Even if the shares do not contain an obvious part to be "stripped off," the malicious shareholders can apply much more sophisticated strategies, especially since we must assume that the shareholders know the judge and choose their collusion method to prevent punishment. A particular subtle adversarial strategy is what we call an *MPC attack*, where the malicious shareholders run a multiparty-party (MPC) protocol [5, 9, 19, 35] in which each shareholder provides as input its secret share S_i, and the protocol securely computes the reconstruction function. In this way, the malicious shareholders may extract all the information about S that is relevant to them without learning the entire S or each other's shares. Hence, the knowledge of S or the other party's share cannot serve as collusion proof. The second problem of the straw man solution is that in our setting, a monolithic adversary controlling the dealer and one of the shareholders can trivially frame the other shareholder as it is aware of all information required to compute the proof of punishment.

Note that the MPC attack may look like something impossible to prevent, especially given the recent advances in the efficiency of the MPC protocols (see, e.g., [22]). In this paper, we challenge this by relying on the recently-introduced idea of *individual cryptography* [15] (similar ideas appeared independently in [23]). Informally, *individual cryptography* studies functions that are hard to implement in MPC or TEEs (i.e., "Trusted execution environments", see, e.g., [34]). This is done by designing "MPC-hard functions" which work by requiring a large number of hash evaluations. This choice comes from the fact that hashes (such as those from the SHA family) do not have an algebraic structure that could be exploited in an MPC attack. In the model of [15], it is assumed that a small number of hashes can actually be computed in an MPC, but the fast computation of a massive amount of hashes is infeasible (see Sect. 2 for the details).

Our contributions. As highlighted above, we introduce a new variant of secret sharing that we call *secret sharing with snitching (SSS)*. Our contributions can be summarized as follows.

(1) We define the notion of t-out-of-n secret sharing with snitching in the individual cryptography model of [15] (see Sect. 2). As already highlighted, this means that we base "MPC-hardness" on the assumption that computing massive amounts of hashes quickly in MPC is infeasible. At the same time, performing such computations individually can be done very efficiently.
(2) We construct (in Sect. 3) an n-out-of-n SSS and then, using this construction as a building block, a t-out-of-n SSS for an arbitrary t (see Sect. 4). We remark that starting with the construction of an n-out-of-n scheme serves two purposes. First, our n-out-of-n scheme offers better efficiency than what can currently be achieved by our generic construction of a t-out-of-n SSS. Second, our n-out-of-n scheme is an important building block for our construction of the t-out-of-n SSS and hence helps with modularization.
(3) We prove the security of our constructions. Our scheme is secure assuming that the number of hashes computed "in MPC" by the adversarial parties is a constant fraction of the number of hashes computed by the honest users of the protocol.[2]

1.1 Applications

Our solution can be used in scenarios where a client (acting as the dealer) needs to temporarily share its secret with external servers, with the intention that the secret become public at some point. The judge (say, a smart contract) comes with a condition when the secret is allowed to be reconstructed. Any attempt by malicious shareholders to reconstruct the secret before this condition is met allows the punishment of parties involved in the illegal reconstruction. Such punishment may, e.g., result in slashing some deposits that the shareholders had to put into the judge's smart contract, thus disincentivizing collusion among the shareholders. Hence, the client is guaranteed that if the servers attempt to reconstruct the key material illegally, they always risk punishment. On the other hand, thanks to the unframability property, an honest server that plays by the rules of the game never faces the risk of being falsely accused. We emphasize that our notion of secret sharing with snitching very much resembles how collusion is prevented in the real world. For example, illegal price agreements between companies or stock market manipulation is often contained in practice by putting high punishments when cheating is detected.

As a more concrete application, consider the *front-running attacks* [3, 14]. This class of powerful attacks is based on the fact that most of the blockchains are asynchronous, in the sense that a transaction *tx* that is sent to them becomes public *before* it is accepted into the blockchain. This means that powerful actors (e.g., the miners) can place their transactions *earlier* on the blockchain, even though they were produced *later* than *tx* (and possibly depending on it). Such attacks can lead to risk-free financial arbitrage and are routinely carried out in practice, leading to huge financial losses for honest blockchain users.

One way to thwart such attacks is the use of consortium-based blockchain solutions like the Shutter network[3] or i-TiRE [2]. The consortia are responsible for making sure that all the transactions that were submitted in a given time period are decrypted simultaneously, hence preventing the front-running problem. Such solutions are based on secret sharing and their security currently relies on strong honesty assumptions about the underlying consortia. In short, it is assumed that at most $t-1$ consortium members are corrupt, and the remaining ones are perfectly honest. This assumption is problematic in the blockchain settings, as nothing prevents t consortium members from secretly reconstructing user transactions and performing front-running covertly. SSS can be used in this setting to prevent such covert illegal reconstruction of secrets. More precisely, each user of the system would share his transaction with the consortium members using SSS, and the transactions would be reconstructed only after some time has passed (earlier reconstruction would be punished).

[2]The honest users are not required to compute hashes in MPC during reconstruction.
[3]See https://shutter.network.

Another class of applications are the online sealed-bid auctions, where the auctioneers should cast their bids without knowing the other bids. Yet another ones, are voting protocols, in order to prevent the tactical voting. In all of these cases, the bids/votes would be shared using SSS with a consortium of trustees, who would open them only once the auction/voting is over. Finally, one can use our solution to implement secure document discourse after an embargo period: a user that wants to publish a document after some embargo time, can use SSS to share it. In this way, she can be sure that it remains secret until the embargo is over, but then it gets published, even if she goes offline.

Note that all of the above examples can be generalized to a wider class of reconstruction conditions. For example, the reconstruction point can be defined not only depending on time but also on some event happening, e.g., the transactions should get reconstructed only when an exchange rate of some digital asset reaches a certain point, or when a document that was embargoed can be safely released, because some external entity agreed on it.

Another type of application where these ideas can be useful is the private-information retrieval schemes where two (or more) non-colluding servers allow more efficient solutions than single-server ones (see, e.g., [12]). We leave exploring these ideas as future work.

1.1.1 Comparison with alternative solutions. Let us also compare our solution with the alternative ones. An obvious alternative cryptographic approach to the front-running problem listed above is to use cryptographic commitments, the so-called *commit-reveal* technique. In practice, unfortunately, this solution does not work, since the users can always refrain from opening their transactions after seeing the openings of the other ones or submit several transactions, and only later decide which of them to open. In principle this could be addressed by forcing the users to put deposits (and slashing them if they do not open their commitments), but this solution is considered impractical as it affects the users experience, and exposes them to the risk of losing money if they accidentally lose the Internet connection. In the case of voting, requiring the users two interact with the system twice (for "commit" and for "reveal") may clearly be impractical.

Another class of alternative solutions is to use *time-based cryptography* [29], where a message is encrypted in such a way that decrypting it takes a substantial amount of sequential work. The main problem with such solutions is the need to estimate hardness of the underlying computational problems. If decrypting a puzzle takes time T on the honest users computers, then it is prudent to assume that the adversary can do it much faster (say: in time $T' \ll T$), due to the fact that she may have a better hardware or know better algorithms. This means that it is impossible to precisely estimate time when a given secret will be made public. For example, for the front-running prevention, this would mean that in the time period between T' and T no new message can be posted, but the old messages are still not known publicly. In case of voting, this would mean that the time between closing the voting, and announcing the result is long. This problem gets even larger if we allow the messages to be posted in relatively long period. For example, this is the case in the case of voting, as it would mean that the time between closing the vote and announcing the result is probably longer than the duration of the voting. Our solution does not have this problem, since the amount of work that needs to be done by the honest parties is much smaller than the work that has to be performed by the adversaries (if they want to avoid punishment then need to perform the same work as the honest parties, but in MPC).

1.2 Other related work

As discussed above, secret sharing comes in many flavors, and we only discuss the most relevant works here. Our model is a bit reminiscent of the work on *rational cryptography*, see, e.g., [17] and the rational secret sharing [21]. This notion studies how to incentivize rational shareholders to collaborate to reconstruct the secret. Hence, rational secret sharing is dual to our notion of secret sharing with snitching. The main difference is that in our modeling, we do not rely on game theory. Instead, we use techniques from individual cryptography. Another important line of related research is on collusion-free protocols [1, 25, 33]. However, these works do not consider the MPC attacks, one of the core problems we prevent in our work. Following a long line of work on traitor tracing, there has recently been some important work on lifting traitor tracing to the threshold setting [6, 7]. Both works consider a setting where a (sub)set of malicious shareholders cooperate to build a decryption/reconstruction box that contain partial information about the secret shares of the parties. The security property guarantees that such colluders can be traced via a tracing algorithm that has black box access to the decryption/reconstruction box. The main difference between traitor tracing threshold crypto schemes and our work is that [6, 7] do not protect against the MPC attacks. In particular, they consider that running an MPC protocol to compute a function depending on all secret shares is outside their model. On the positive side, by eliminating MPC attacks from their model, their security guarantees do not rely on the MPC hardness of certain functions. Recently, [20] consider how to de-incentivize collusion in multi-server PIR.

2 The definition of the Secret Sharing with Snitching

This section provides a formal description of our model and security definitions. Let 1^κ be a security parameter and let $H\colon \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ be a function modeled as a random oracle [4], denoted as Ω_H. Note that, similarly to [15], we consider hashes of short input (in practice, H could simply be a Merkle-Damgard hash compression function). See [15] for the discussion on this assumption.

The i-th bit of a binary string Y is denoted as $Y[i]$. Let e denote Euler's constant, and let $\exp(x) := e^x$. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. We will also use Shamir's secret sharing [32] as a building block. Let $(\mathbb{F}, \oplus, \times)$ be a finite field and let $t \leq n$. Shamir's t-out-of-n secret sharing is a pair of algorithms $(\mathsf{share}^{\mathsf{Shamir}}_{(t,n)}, \mathsf{rec}^{\mathsf{Shamir}}_{(t,n)})$ defined as follows. Let $1, \ldots, n$ be some distinct elements of $\mathbb{F}$. Then for $S \in \mathbb{F}$ we define $\mathsf{share}^{\mathsf{Shamir}}_{(t,n)}(S) := ((1, P(1)), \ldots, (n, P(n)))$, where P is a random polynomial over $\mathbb{F}$ with degree at most $t-1$ such that $P(0) = S$ and $\mathsf{rec}^{\mathsf{Shamir}}_{(t,n)}((i_1, X_1), \ldots, (i_t, X_t)) := P'(0)$, where P' is the polynomial of degree at most t interpolated at points $(i_1, X_1), \ldots, (i_t, X_t)$. It is well-known [32] that such interpolation is

always possible, while the set of at most $t-1$ shares $P(i)$ reveals no information about S.

2.1 Basic setting and terminology

Secret sharing with snitching (SSS) with respect to an oracle Ω_H is a tuple $\Pi := (\mathsf{J}, \mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n, \Omega_H^{\mathrm{MPC}})$ of interactive polynomial-time machines with access to Ω_H. Machines J and Ω_H^{MPC} are deterministic, and the others are randomized. Machine Ω_H^{MPC} is stateless. An SSS scheme Π is parametrized by three parameters: the *reconstruction threshold* $t \in \mathbb{N}$ (with $t \le n$), the *adversarial slow query budget* $\sigma \in \mathbb{N}$ (the notion of "slow queries" is defined below), and the length λ of the shared secret S (λ is a polynomial function of the security parameter). We will also say that Π is *t-out-of-n-SSS*.

We will refer to J as the *judge*, to D as the *dealer*, to the P_i's as the *shareholders*, and to Ω_H^{MPC} as the *multi-party random oracle*. The parties operate in a synchronous network, are connected by private channels, and have access to a broadcast channel. Time is divided into rounds, and the messages sent in round j arrive at the destination in round $j+1$. The judge's state is public (recall that we think of it as a smart contract or a public bulletin board, see Section 1): it is broadcast to all the parties in every round. We consider two types of adversarial models: the *monolithic* and the *distributed* one. In both of them, the adversary (or the "subadversaries", see below) can corrupt some parties. Once a (sub)adversary corrupts some party, it completely controls it: it learns P_j's entire state and history and receives and sends messages in P_j's name. A party that is not corrupt is *honest*. The judge J and the multi-party random oracle Ω_H^{MPC} are always honest. We explain the role and operation of Ω_H^{MPC} in Section 2.1.3. First, let us describe the different adversarial models. The first one, which we refer to as the *monolithic model*, is used in the unframability definition, while the second one, called the *distributed model*, is used in the punishability definition.

2.1.1 Monolithic adversarial model. The monolithic model is used in the definition of unframability of SSS. It is the standard cryptographic model, in which a *monolithic* active polynomial-time adversary $\mathcal{M}$ can adaptively corrupt parties from the set $\{\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n\}$. The adversary can access Ω_H freely. A monolithic adversary is rushing and controls the network.

2.1.2 Distributed adversarial model. For the punishability definition, 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 randomized polynomial-time machines called *subadversaries*. This model is a variant of the one introduced in Dziembowski et al. [15] (see also Section 2.4.1 for a comparison between our model and the model of [15]),

During the execution of a protocol, the subadversaries can communicate freely. Each subadversary can corrupt some shareholders (in this setting, both J and D are incorruptible[4]) and take control over it. Let C_j denote a set of parties that are corrupted by a subadversary $\mathcal{A}_j$. We will call such C_j, the *$\mathcal{A}_j$'s coalition*. We require that all the coalition 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 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 C_i is less than t. One can think of the coalitions as the Sybil identities of a subadversary or as a set of parties that ultimately trust and never betray each other.

The adversary $\mathcal{A}$ communicates with the outside world via one of the subadversaries, say $\mathcal{A}_1$, who can pass the received messages between the other subadversaries and the external entities. When we say that *a message m is sent to (or "sent by") $\mathcal{A}$*, we mean that it is sent to (or "sent by", resp.) $\mathcal{A}_1$. The subadversary $\mathcal{A}_1$ gets as input 1^κ and passes it to the other subadversaries. It also controls the network (we assume a rushing model).

2.1.3 The multi-party random oracle. In order to model the evaluations of H done over MPC, we introduce a natural generalization of the random oracle model of [4]. As mentioned before, we allow the honest parties and the adversary to evaluate a random hash function $H\colon \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$. We do it in two ways. First, each of them can directly access Ω_H and query it on the inputs of their choice. Second, we need to model the fact that both the honest parties and the adversary are allowed to compute a restricted number of hashes H in a distributed way using an MPC protocol. We do it by letting them access Ω_H via the multi-party random oracle Ω_H^{MPC} (see below for the details) and imposing a more restrictive bound on the number of times they can call Ω_H^{MPC}. The queries to Ω_H are called *fast*, while the queries to Ω_H^{MPC} are referred to as *slow*. The number of fast queries is not restricted (can be an arbitrary polynomial in κ), while the number of slow queries is bounded (for the adversary), or considered to be an important efficiency parameter (for the honest parties). This corresponds to the fact that the execution of an MPC protocol is slow and expensive, orders of magnitude slower than an individual evaluation of a hash function. We describe the interaction between the oracles and the other entities in the system below (see also Fig. 1 for its graphical presentation).

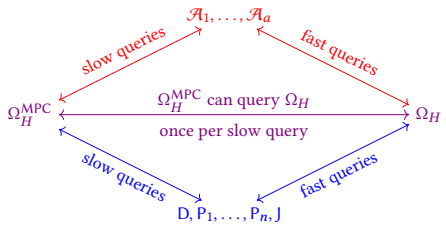

Figure 1: The interaction in the system.

The use of Ω_H^{MPC} by the honest parties. The honest parties query Ω_H^{MPC} as follows. Each party $\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n$ sends its input $x_{\mathsf{D}}, x_1, \ldots, x_n$ (respectively) to the oracle Ω_H^{MPC}. The multi-party random oracle performs deterministic precomputation φ on these inputs and queries Ω_H, obtaining $y = H(\varphi(x_{\mathsf{D}}, x_1, \ldots, x_n))$. Only one query to Ω_H is allowed for each call to Ω_H^{MPC}. Then, the multi-party random oracle passes the oracle response y to one of the subadversaries (say: to $\mathcal{A}_1$), who can deliver to each party either the actual value y or $\perp$, in which case we say that the computation failed. This models the fact that in the dishonest majority settings, the adversary can always abort the protocol and prevent some parties from learning

[4] We can assume that the dealer is incorruptible because the distributed adversarial model is used in the punishability definition only.

the output. In practice, such multi-party calls can be implemented using standard techniques for the MPC (see, e.g., [13]).

The use of Ω_H^{MPC} by the distributed adversary. In the distributed adversarial model, the subadversaries can access the Ω_H^{MPC} exactly like the honest parties, except we assume that the output is always delivered to them. More concretely, each subadversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$ sends its input $x_1, \ldots, x_a$ (respectively) to the oracle Ω_H^{MPC}. The oracle Ω_H^{MPC} queries Ω_H *once* and sends the output of this computation to all the subadversaries.

The execution of a protocol is divided into the *precomputation* epoch and the *online* epoch (which itself consists of three phases, see Section 2.2) happening one after another. The total number of slow queries answered in the online epoch is at most σ. The queries that exceed this quota are answered with $\perp$. On the other hand, the number of slow queries during the precomputation epoch is bounded only by the computing time of the adversary (i.e., it is polynomial in κ). This corresponds to the fact that the adversaries can have a long "precomputation period" before the protocol starts, and the MPC execution time is restricted only when the protocol execution takes place. As in [15], in order to reason about the information each subadversary submitted to the oracle, we define the *fast-oracle transcript of a subadversary $\mathcal{A}_j$* to be the sequence $\mathcal{T}_j$ of oracle inputs that Ω_H received from $\mathcal{A}_j$ (in the same order in which they were received).

Counting the number of slow queries. For the honest parties, the number of calls to the Ω_H^{MPC} will be an efficiency parameter #slow. For the adversary, the number of queries to Ω_H^{MPC} is bounded by the adversarial slow query budget parameter σ. In our constructions, we will have $\#\mathsf{slow} \ll \sigma$, i.e., the number of times the honest parties need to compute H in MPC will be much smaller than the bound σ of such MPC computations of the adversary. In particular, $\#\mathsf{slow} = \kappa$ in the n-out-of-n construction, and $\#\mathsf{slow} = n^2 \cdot \kappa$ in the t-out-of-n construction. At the same time, we have that $\kappa \geq 2\log_2 \sigma$, that is $\sigma \leq 2^{\kappa/2}$, therefore even σ that is exponential in κ can be accommodated.

2.2 Protocol execution

An SSS protocol Π is divided into three phases executed during the online epoch in the following order.

Sharing phase: In this phase, denoted $\Pi.\mathsf{share}$, all the parties take as input 1^κ, and the dealer D additionally takes as input a *secret* $S \in \{0,1\}^{\lambda(\kappa)}$. At the end of this phase, the judge may output error, in which case, the protocol terminates (and we say that the sharing *failed*). Otherwise, he outputs ok and the parties proceed to one of the next phases. During this phase, the dishonest shareholders may engage in an illegal reconstruction of a secret.

Punishment phase (of P_i snitched by $\mathcal{A}_j$): This phase is executed between the dealer D and the judge J. It is denoted $\Pi.\mathsf{punish}_{\oplus}^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_i}$ and is parametrized by a sub-adversary $\mathcal{A}_j$ (called a *snitcher*) and a shareholder P_i (called a *target*) who does not belong to $\mathcal{A}_j$'s coalition C_j. The dealer D receives the fast-oracle transcript $\mathcal{T}_j^{\mathsf{fast}}$ of the subadversary $\mathcal{A}_j$ (which may be empty if the parties in $\mathcal{A}_j$'s coalition did not compute any hashes) and may send a message to the judge J. At the end of this phase, the judge outputs $(\mathsf{punished}, \mathsf{P}_i)$ (where P_i is the target) or $\perp$. Note that the snitcher $\mathcal{A}_j$ does not explicitly participate in this phase. This is just a formal abstraction: in practice, it would be $\mathcal{A}_j$ who would deliver $\mathcal{T}_j^{\mathsf{fast}}$ to the judge.

Reconstruction phase: This phase is denoted $\Pi.\mathsf{reconstruct}$ and is executed between a set of shareholders of size t (the dealer D and the judge J do not participate in this phase). At the end of it, the participating parties output $\widehat{S} \in \{0,1\}^* \cup \{\perp\}$.

2.3 SSS properties

An SSS protocol should satisfy three requirements: correctness, punishability, and unframability that we describe below. They were introduced informally already in Sect. 1. Below, we define them more formally.

Correctness. The correctness requirement is standard and states that if all the parties are honest, then for every input $S \in \{0,1\}^{\lambda(\kappa)}$ with overwhelming (in κ) probabilitythe output $\widehat{S}$ of each honest party in the reconstruction phase is equal to S.

Punishability. The punishability property guarantees that if the shareholders engage in an illegal secret reconstruction protocol, then there exists a target P_i and a snitcher $\mathcal{A}_j$, (with $\mathsf{P}_i \notin C_j$), such that $\mathcal{A}_j$ can collaborate with the dealer D (as remarked above, technically, this means that the dealer simply receives the transcript $\mathcal{T}_j^{\mathsf{fast}}$ of $\mathcal{A}_j$) to generate a proof for the judge that will convince her to punish P_i. The probability that the punishment procedure works depends on the amount of information that the shareholders obtain about the shared secret measured in terms of the "distinguishing advantage". Thanks to this approach, if the malicious parties decide to illegally reconstruct the secret only with some probability p (and with probability $1-p$ they behave honestly), then the punishment happens with probability close to p.

More formally, let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ be an arbitrary distributed adversary and define

$$\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi; \kappa]\!] := |\Pr\big[\mathsf{Sharing}[\![\mathcal{A} \leftrightarrows \Pi, 0; \kappa]\!] = 1\big] - \tag{1}$$

$$\Pr\big[\mathsf{Sharing}[\![\mathcal{A} \leftrightarrows \Pi, 1; \kappa]\!] = 1\big]| \tag{2}$$

and let S be an arbitrary secret of length $\lambda(\kappa)$. Consider an execution of $\mathcal{A}$ against Π with security parameter 1^κ and the dealer's secret S, as defined in the $\mathsf{Snitching}[\![S, \mathcal{A} \leftrightarrows \Pi; \kappa]\!]$ experiment (cf. Fig. 2). Define the following event:

$$\begin{aligned}\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!] := \text{after the sharing phase, there exists} \\ \text{a target } \mathsf{P}_i \text{ and a snitcher } \mathcal{A}_j \text{ such that} \\ \mathsf{Snitching}[\![S, i, j, \mathcal{A} \leftrightarrows \Pi; \kappa]\!] = (\mathsf{punished}, \mathsf{P}_i).\end{aligned} \tag{3}$$

Then for all $S \in \{0,1\}^{\lambda(\kappa)}$ we have that $\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi; \kappa]\!] - \mathsf{negl}(\kappa) \leq \Pr\big[\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!]\big]$.

Unframability. The unframability property guarantees every honest shareholder P_{i^*}, that even if the dealer and all the remaining shareholders maliciously collaborate, they cannot *frame* P_{i^*}, i.e., with overwhelming probability they cannot trick the judge J into outputting a punishment message against P_{i^*}. More formally, for every $i^* \in \{1, \ldots, n\}$, every monolithic adversary $\mathcal{M}$ that does *not* corrupt P_{i^*} we have that in every execution of $\mathcal{M}$ against Π

Experiment Sharing$[\![\mathcal{A} \leftrightarrows \Pi, b; \kappa]\!]$

1. A pre-computation epoch is executed. During it, the subadversaries can submit an arbitrary number of slow queries to Ω_H.
2. $\mathcal{A}$ selects two secrets $S^0, S^1 \in \{0,1\}^{\lambda(\kappa)}$.
3. The sharing phase of Π is executed with the security parameter 1^κ and $S = S^b$. During this phase, the distributed adversary $\mathcal{A}$ attacks the protocol Π and possibly tries to obtain some information about the shared secret.
4. The output of the experiment is equal to the output of $\mathcal{A}$.

Experiment Snitching$[\![S, i, j, \mathcal{A} \leftrightarrows \Pi; \kappa]\!]$

1. A pre-computation epoch is executed. During it, the subadversaries can submit an arbitrary number of slow queries to Ω_H.
2. The sharing phase of Π is executed with the security parameter 1^κ and S being the dealer's secret. During this phase, $\mathcal{A}$ attacks the protocol Π.
3. The punishment phase of P_i snitched by $\mathcal{A}_j$ is executed.
4. The output of the experiment is defined to be the output of the judge J in Π at the end of the punishment phase.

Figure 2: The experiment Snitching$[\![S, \mathcal{A} \leftrightarrows \Pi; \kappa]\!]$**.**

on security parameter κ, and every $S \in \{0,1\}^{\lambda(\kappa)}$ we have that $\Pr[\mathsf{J} \text{ outputs } (\mathsf{punished}, \mathsf{P}_{i^*}) \text{ in the punishment phase}] \leq \mathsf{negl}(\kappa)$. Note that the assumption that $\mathcal{M}$ is monolithic only makes this requirement stronger, since this means that we do not impose any restrictions on how the hashes H are computed by the adversary.

2.4 Remarks on the definition

2.4.1 Comparison to Dziembowski et al. Our model is similar to the one of [15] in the sense that we also consider distributed adversaries that consist of "subadversaries". We also have two types of oracle queries: the expensive ("slow") ones, corresponding to the queries evaluated in MPC, and the cheap ("fast") ones that are computed individually by the subadversaries. The authors of [15] do not have the multi-party random oracle Ω_H^{MPC} since they do not need to model the access to the random oracle Ω_H from within an MPC protocol executed by the honest parties. From the security model point of view, this is mostly a syntactic difference. In [15], the authors also construct a secret sharing scheme called "individual secret sharing" (ISS), whose main distinguishing feature is that the reconstruction of a secret has to be performed individually. The main difference between ISS and SSS is that ISS does not have a punishment mechanism, particularly one that would prevent the framing of honest parties.

2.4.2 On the length of S. Our definition considers only SSS where the secrets are of fixed length. This technical requirement is needed to guarantee that S (used in the punishment experiment) has the same length as S_0 and S_1 chosen by $\mathcal{A}$ in the indistinguishability experiment. Since we want the snitching to work for any S, if we had not put any restriction on $|S|$, we would get to a definition that is impossible to satisfy, as the adversary's actions may depend on the length of the shared secret (which, in general, is impossible to hide from him).

2.4.3 The timeframe of punishment. Note that the fraud proofs are inherently meaningless anytime following a legal reconstruction. However, we do not impose any requirements on when the snitching has to happen, and it is up to the protocol designer to guarantee that the fraud proof is sent to the judge while the reconstruction is still considered illegal. Since the judge can be thought of as a smart contract, the timeframe can be a part of its public state and set in a trustless manner upon deployment.

2.4.4 The target in the punishment phase. In our definition of punishability, we only assume that there exists a party that one can snitch on. In other words, we do not assume *how* the snitching target is chosen, and this is left as an implementation detail. The actual algorithm for choosing the party being punished may vary depending on the desired guarantees and incentives. One might want to punish only a single shareholder or, in other cases, every shareholder but the snitcher. The bottom line is that the threat of punishment should discourage dishonest behavior, and the honest parties will not be punished.

2.4.5 Adaptivity in choosing the snitcher. Additionally, our definition of punishability allows for an adaptive choice of the *snitcher*. One could consider a non-adaptive model, in which the snitcher is chosen upfront before the execution of the protocol. Note that the non-adaptive choice of the snitcher would ensure *stronger* security guarantees, unlike in the case of adaptive corruption in multi-party protocols. A non-adaptive choice would correspond to an undercover agent being planted in a criminal group by the law enforcement, whereas our adaptive model corresponds to a criminal acting as a crown witness for incentives of any sort. Below, we explain why the non-adaptive model has some limitations.

First of all, it might turn out that every party involved in the scheme has a small chance of being able to snitch. As an example, consider an execution of the SSS protocol where each party P_i is corrupted by a different subadversary $\mathcal{A}_i$. The parties pick a leader $\mathsf{L} \in \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ uniformly at random and send all their data to the leader. Then the leader is the only one who can snitch, and every party can snitch with probability $1/n$, even though the distinguishing advantage of $\mathcal{A}$ equals 1. Moreover, a non-adaptively chosen party might not be able to snitch at all. Consider a variant of the example above, in which the leader L is a *fixed* party. Then, every party $\mathsf{P}_i \neq \mathsf{L}$ can snitch with probability 0. Last but not least, the adaptive model is enough for most applications of our protocol, as one should be able to provide incentives for the snitchers. One could, for instance, incentivize the snitching party by offering a part of the dishonest parties' deposit. A detailed study of possible cryptoeconomic incentives is, however, out of the scope of this paper.

2.4.6 Punishability and unframability implies secrecy. Note that the above definition does not contain any explicit secrecy requirement. We now informally argue that secrecy (against a group of at most $t-1$ malicious shareholders) is already implied by the other properties,

namely by the *unframability* and the *punishability*. For the sake of contradiction, suppose that a group C (such that $|C| \leq t-1$) of the shareholders gets enough information during the sharing procedure to win the distinguishing game with a non-negligible advantage ε (see Eq. (1) on Fig. 2). In other words, for a distributed adversary $\mathcal{A}_j$ corrupting coalition C, we have that $\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi; \kappa]\!] = \varepsilon$. Punishability implies that $\Pr\left[\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!]\right] \geq \varepsilon - \mathsf{negl}(\kappa)$ and the right-hand side is non-negligible. Suppose all the parties not in C remain honest (but curious). By the unframability of the protocol, they cannot be punished. So, the only parties that can be punished are members of C. The only non-trivial fast hash transcript is the one of $\mathcal{A}_j$, so $\mathcal{A}_j$ is the only possible snitcher. But we require that a subadversary never snitches on her coalition members. This yields a contradiction.

2.4.7 Verifiable SSS. To simplify the exposition and clearly present our main idea, we decided not to overload our definition with security requirements. Hence, we do not consider other properties commonly required from secret sharing, such as verifiability. Our definitions and both our schemes do not account for the guarantees of Verifiable Secret Sharing [10]. We consider the detailed analysis to be out of the scope of the paper, and we will thus not provide any formal definitions. However, we will sketch how one can extend the protocol to provide Verifiable Secret Sharing with Snitching. Recall that in the VSS, one needs to take into account that (1) the dishonest dealer might send inconsistent shares to the shareholders (e.g., shares that are not evaluations of a polynomial of degree t), and (2) the malicious shareholders can lie about their shares, preventing the reconstruction of S (even if the dealer was honest).

These problems can be addressed using zero-knowledge (ZK) techniques, especially their non-interactive versions, like zk-SNARKS. Observe that the sharing procedure requires a small number of hash computations (massive computations of H are present only in the reconstruction phase). Hence, we can instruct the dealer to prove in ZK to all the shareholders that the shares he shared are consistent (in particular, in the threshold case: that they interpolate a polynomial of degree t). Moreover, the dealer can sign every share (note that this would require using MPCs, since the shares are not known to the dealer) to prevent the problem of lying shareholders. We leave formalizing these solutions and finding more efficient and practical alternatives as future work.

3 The n-out-of-n SSS scheme

In this section, we will introduce a construction for secret sharing with snitching, where the dealer distributes shares to n parties and all n parties are needed for reconstructing the secret. In Sections 3-4, we use the following conventions. For a security parameter κ, define $G\colon \{0,1\}^\kappa \to \{0,1\}^{\lambda(\kappa)}$ as $G(X) := H(\text{“0”} \,||\, X) \,||\, \ldots \,||\, H(\text{“}\lceil \lambda(\kappa)/\kappa \rceil\text{”} \,||\, X)$, where “$k$” denotes the binary representation of a natural number (of length κ) and the last block $(H(\text{“}\lceil \lambda(\kappa)/\kappa \rceil\text{”} \,||\, X))$ is truncated to make $|G(X)| = \lambda(\kappa)$. We also use the following convention: for input X such that $|X| < \{0,1\}^{2\kappa}$ by writing $H(X)$ we mean $H\left(X \,||\, 0^{2\kappa - |X|}\right)$. We will assume that $\lambda(\kappa) \leq \kappa 2^\kappa$ so that this definition is well-formed.

3.1 Our construction

Our n-out-of-n scheme $\Pi_\oplus$ consists of three phases: $\Pi.\mathsf{share}_\oplus$, $\Pi.\mathsf{punish}_\oplus^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_i}$ and $\Pi.\mathsf{reconstruct}_\oplus$, presented on Figs. 3 to 5. Our construction uses an integer parameter $d := \lceil \log_2(\sigma/\kappa) \rceil + 4$ called *difficulty*. During $\Pi.\mathsf{share}_\oplus$ the dealer wishes to share a secret $S \in \{0,1\}^{\lambda(\kappa)}$.

$\Pi.\mathsf{share}_\oplus$ between $\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n$ and J

1. All parties take as input 1^κ. Additionally, the dealer D takes as input a secret $S \in \{0,1\}^{\lambda(\kappa)}$.
2. The dealer samples $Y^1, \ldots, Y^\kappa \leftarrow\!\$\, \{0,1\}^d$.
3. Each shareholder P_i samples $X_i^1, \ldots, X_i^\kappa \leftarrow\!\$\, \{0,1\}^\kappa$.
4. For $\ell := 1, \ldots, \kappa$, all the parties compute, using the multi-party random oracle, the following values:
$$Z^\ell := H\left((X_1^\ell \oplus \cdots \oplus X_n^\ell) \,||\, Y^\ell\right).$$
If any of the above MPC computations was not successful for some party, then the party aborts.
5. Each party $\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n$ sends $Z^1, \ldots, Z^\kappa$ to the judge J.
6. If the judge J did not receive $Z^1, \ldots, Z^\kappa$ from all the parties or if these messages are not identical, then he outputs error, and all the parties terminate.
7. Otherwise, the dealer D computes $R := H(Y^1[1], \ldots, Y^\ell[1])$ and sends $C := G(R) \oplus S$ to the shareholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$.

Figure 3: The sharing phase of the SSS scheme $\Pi_\oplus$.

$\Pi.\mathsf{punish}_\oplus^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_i}$ between D and J

1. The dealer receives the fast-oracle transcript $\mathcal{T}_j^{\mathsf{fast}}$ of $\mathcal{A}_j$.
2. For each query $(Q \,||\, Q') \in \{0,1\}^{2\kappa}$ in the transcript $\mathcal{T}_j^{\mathsf{fast}}$ (where $|Q| = |Q'| = \kappa$), the dealer checks if $H(Q \,||\, Y^\ell) = Z^\ell$ (for some ℓ). If yes, then he sends a message $(\mathsf{punish}, i, Q, Y^\ell, \ell)$ to the judge.
3. If indeed $H(Q \,||\, Y^\ell) = Z^\ell$ then the judge outputs $(\mathsf{punished}, \mathsf{P}_i)$.

Figure 4: The punishment of P_i snitched by $\mathcal{A}_j$ in $\Pi_\oplus$.

$\{0,1\}^{\lambda(\kappa)}$ that can be later reconstructed during $\Pi.\mathsf{reconstruct}_\oplus$. The main challenge is that if, on one hand, the shareholders attempt an illegal reconstruction, then at least one of them (the snitcher $\mathcal{A}_j$) should provide evidence for the dealer that suffices to punish one of the colluders P_i (recall that technically, the punishment is done in the $\Pi.\mathsf{punish}_\oplus^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_i}$ procedure by the dealer who uses the transcript $\mathcal{T}_j^{\mathsf{fast}}$ of $\mathcal{A}_j$ to produce the punishment proof). This is guaranteed by the *punishability* property. On the other hand if some shareholder P_i is honest, then even an adversary that corrupts the dealer and all the other shareholders should not be able to convince the judge to punish P_i. This is guaranteed by the *unframability*

$\Pi.\mathsf{reconstruct}_{\oplus}$ **between** $\mathsf{P}_1, \ldots, \mathsf{P}_n$

1. The shareholders appoint a chair, say P_1.
2. Each P_i sends $(X_i^1, \ldots, X_i^\kappa)$ to the chair P_1.
3. For $\ell := 1, \ldots, \kappa$, the chair computes $X^\ell := X_1^\ell \oplus \cdots \oplus X_n^\ell$.
4. For $\ell := 1, \ldots, \kappa$, the chair performs a brute-force search to find a value $Y^\ell \in \{0,1\}^d$ such that $H(X^\ell \,||\, Y^\ell) = Z^\ell$. If for some ℓ no such Y^ℓ is found, then P_1 outputs $\perp$.
5. The chair computes $R := H(Y^1[1], \ldots, Y^\ell[1])$ and sends $\widehat{S} := C \oplus G(R)$ to all the shareholders.
6. Each shareholder outputs $\widehat{S}$.

Figure 5: The reconstruction phase of the SSS scheme $\Pi_{\oplus}$.

property. Let us discuss how these two goals are achieved, starting with a toy solution, in which the dealer is assumed to be honest, i.e., the unframability property is trivially achieved.

Recall from the introduction that we need to protect against MPC attacks. Hence, the first idea is to use an MPC-hard function that the shareholders have to evaluate in order to reconstruct the secret S. MPC-hardness here means that to evaluate the MPC-hard function, the shareholders have to carry out a large number of hash evaluations, which cannot be done efficiently via an MPC protocol. In our solution, the MPC-hard function consists of finding a partial preimage of a hash function, i.e., for some random $X \in \{0,1\}^\kappa$ find $Y \in \{0,1\}^d$ such that $H(X \,||\, Y) = Z$. Given such an MPC-hard function, our toy solution proceeds as follows. First, the dealer samples $X_1, \ldots X_n \leftarrow\!\!\$\ \{0,1\}^\kappa$ and $Y \leftarrow\!\!\$\ \{0,1\}^d$. Then it computes $Z := H(X_1 \oplus \ldots \oplus X_n \,||\, Y)$ and the "ciphertext" $C := G(X_1 \oplus \ldots \oplus X_n \,||\, Y) \oplus S$.[5] Finally, it sends (X_i, C) to each shareholder P_i and Z to the judge. During the snitching phase $\Pi.\mathsf{punish}_{\oplus}^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_i}$, the judge will accept a fraud proof against some shareholder P_i if it receives some preimage of Z.

It is easy to see that in our toy solution, the shareholders can reconstruct S by computing $X := X_1 \oplus \ldots \oplus X_n$ and then letting one of the shareholders (called the chair) solve the MPC-hard function by finding Y such that $H(X \,||\, Y) = Z$. Hence, *correctness* of the scheme is satisfied. Let us next take a look at the *punishability* property. Since the distributed adversary only has a limited budget of queries to the multi-party random oracle, there are only two ways to compute Y and recover S. First, the adversary may get lucky and manage to invert H using slow queries only, that is, use the correct value of Y when evaluating $H(X \,||\, Y)$ via the multi-party random oracle. Since the corresponding query is slow, the adversary cannot be punished, thus breaking the punishability property. While in our toy solution, this "bad case" may indeed happen with noticeable probability, it can easily be addressed by increasing the number of hashes to invert (see the discussion below). Second, the adversary may ask one of the subadversaries $\mathcal{A}_j$ to reconstruct X individually and then find Y using fast queries to the hash oracle Ω_H. But in this case, $\mathcal{A}_j$ knows (X, Y) such that $H(X \,||\, Y) = Z$ and can send a valid fraud proof to the judge. Thus, except for the "bad case" mentioned above, if the corrupted shareholders attempt to learn any information about Y, then there must always exist one party that can send a fraud proof to the judge. This implies our *punishability* property. The above toy solution has a fundamental drawback. Since the dealer knows the pair (X, Y), it can easily frame any honest shareholder, thus breaking the *unframability* property. In order to address this issue, we proceed as follows. Instead of letting the dealer choose $X_1, \ldots, X_n \leftarrow\!\!\$\ \{0,1\}^\kappa$ on its own, we let each shareholder choose X_i themselves and ask the parties to jointly compute $Z := H(X_1 \oplus \ldots \oplus X_n \,||\, Y)$ via the multi-party random oracle.[6] Since $X_1 \oplus \ldots \oplus X_n$ is unknown to any adversary who corrupts the dealer and all but one shareholder, the adversary cannot punish any honest shareholder, thus guaranteeing the *unframability* property.

Finally, let us discuss how we deal with the "bad case" mentioned above. Because of our choice of parameters, the adversary may get lucky and evaluate $H(X \,||\, Y)$ for the correct Y using the multi-party random oracle. This would break the *punishability* property as now no individual shareholder knows Y. In order to address this issue, we essentially repeat the above protocol κ times and accept any of the preimages as a fraud proof. More precisely, each shareholder P_i chooses κ values X_i^ℓ, and the dealer chooses κ values Y^ℓ. Next, the parties compute the $Z^\ell := H(X_1^\ell \oplus \ldots \oplus X_n^\ell \,||\, Y^\ell)$ using the multi-party random oracle and send the Z_ℓ values to the judge. Since now we have multiple Y^ℓ's, we can simply let the input of G be their xor, and hence adding "$X_1 \oplus \cdots \oplus X_n$" to the input of G is not needed (cf. Footnote 5). Snitching works if any of the shareholders sends a preimage for any of the Z^ℓ. This concludes the high-level description of our protocol. We notice that on a technical level, the analysis is a bit more delicate as we have to ensure that when the colluding parties learn only some (partial) information about S with some probability p, then with the same probability there is one party that can snitch on any of the colluders. We now proceed with the formal proof.

3.2 Security Analysis

In this section, we provide a security analysis of our scheme by proving the following theorem.

Theorem 3.1. *Let σ be a function of κ such that $\kappa \geq 2\log_2 \sigma$. Then the scheme $\Pi_{\oplus}$ (constructed on Figs. 3 to 5) with $d := \lceil \log_2(\sigma/\kappa) \rceil + 4$ is a secure n-out-of-n SSS scheme with a slow query budget σ.*

Proof. In order to prove the theorem, we need to argue that $\Pi_{\oplus}$ is correct, punishable, and unframable. This is done below.

3.2.1 Correctness. Suppose that all the parties are honest. This implies that all the input variables C, R, X_i^ℓ, for $i \in [n]$, $j \in [\kappa]$ used in the reconstruction algorithm were computed according to the instructions in the sharing algorithm. Thus, all the values X^ℓ in Step 3 of the reconstruction algorithm are correct. We only need to show that the values Y^ℓ computed in the Step 4 of the reconstruction procedure are computed correctly. For a fixed $\ell \in [\kappa]$, consider the function $h_\ell \colon \{0,1\}^d \to \{0,1\}^\kappa \colon y \mapsto H(X^\ell \,||\, y)$. In the random oracle model, we have that $\Pr_y\left[h_\ell(y) = Z^\ell\right] = 1/2^\kappa$. Therefore,

[5] The input "$X_1 \oplus \ldots \oplus X_n$" is added to G avoid reconstructing S by simply finding $Y \in \{0,1\}^d$ by brute force.

[6] Notice that this is possible in our model because we allow the honest parties to make a small number of hash function evaluations in MPC. In practice, this means that the multi-party random oracle is realized via a suitable MPC protocol.

there are no collisions of h_ℓ at Z^ℓ with probability equal to $(1 - 1/2^\kappa)^{2^d-1}$. By Bernoulli's inequality, we have that $(1 - 1/2^\kappa)^{2^d} \geq 1 - 2^d/2^\kappa$, which is overwhelming as long as $2^{d-\kappa} = \mathsf{negl}(\kappa)$, which is clearly the case given that $d = \lceil \log_2 \sigma - \log_2 \kappa \rceil + 2$ and $\log_2 \sigma < \kappa/2$. In order to account for all ℓ, we apply the union bound and the probability remains negligible.

3.2.2 Punishability. Fix some distributed adversary $\mathcal{A}$. In this part of the proof we show that

$$\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi_\oplus; \kappa]\!] - \mathsf{negl}(\kappa) \leq \Pr\big[\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!]\big]. \quad (4)$$

First, consider the $\mathsf{Sharing}[\![\mathcal{A} \leftrightarrows \Pi, b; \kappa]\!]$ experiment. For each ℓ, let $Y^\ell[i]$ denote the i-th bit of Y^ℓ and let $\mathsf{Broken}_\oplus$ denote the event that the adversary submitted to Ω_H a query $(Y^1[1], \ldots, Y^\kappa[1])$. From the properties of the random oracle, it follows that as long as $\mathsf{Broken}_\oplus$ did *not* occur, from the adversary's point of view, the value of R is completely uniform, and hence $C = G(R) \oplus S$ perfectly hides S. Therefore, we have

$$\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi_\oplus; \kappa]\!] \leq \Pr[\mathsf{Broken}_\oplus]. \quad (5)$$

Now consider the $\mathsf{Snitching}[\![S, \mathcal{A} \leftrightarrows \Pi_\oplus; \kappa]\!]$ experiment. We now show the following upper bound on $\Pr[\mathsf{Broken}_\oplus]$ in this experiment:

$$\Pr[\mathsf{Broken}_\oplus] \leq \Pr\big[\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!]\big] + \mathsf{negl}(\kappa). \quad (6)$$

Observe that as long as $\mathsf{Broken}_\oplus$ did not occur, the Snitching experiment does not depend on the choice of S, and hence the probability of $\mathsf{Broken}_\oplus$ in the Sharing and in the Snitching experiments are equal, hence (5) and (6) will together imply (4). Thus, what remains to show is Eq. (6).

The proof of Eq. (6). Let us first introduce some notation. Recall that the queries to the oracle have a form (Q, mode), where $Q \in \{0,1\}^{2\kappa}$ and $\mathit{mode} \in \{\mathsf{fast}, \mathsf{slow}\}$. We will consider two different types of queries to the oracle Ω_H. First, we call a query a *scratch (for ℓ)* if it has the form $((X^\ell \,||\, \cdot), \cdot)$ (where "$\cdot$" denotes an arbitrary value). Second, we call a query *successful* if it has the form $((X^\ell \,||\, Y^\ell), \cdot)$ such that $H(X^\ell \,||\, Y^\ell) = Z^\ell$. To simplify our analysis, we introduce the following bad events. (1) bad_1 := "a scratch query for some ℓ was submitted to Ω_H during the pre-computation phase"; (2) bad_2 := "there exist two distinct indices $\ell, \ell' \in \{1, \ldots, \kappa\}$ such that $X^\ell = X^{\ell'}$"; (3) bad_3 := "an adversary and the honest parties found a collision in H, i.e., during the execution of the protocol, two different queries sent to Ω_H resulted in the same output." Let us analyze the probability that bad_i happens. For bad_1 observe that X^ℓ values are chosen uniformly at random from $\{0,1\}^\kappa$, and hence it can be guessed correctly with probability at most $2^{-\kappa}$. For this reason, the probability that a poly-time adversary submits X^ℓ to Ω_h during the pre-computation phase is negligible in κ. Regarding bad_2 observe that if bad_2 does not happen then a given query can be a scratch for at most one ℓ. It is clear that for any $\ell \neq \ell'$ the probability that $X^\ell = X^{\ell'}$ is negligible in κ and hence $\Pr[\mathsf{bad}_2] \leq \mathsf{negl}(\kappa)$. Finally, the probability $\Pr[\mathsf{bad}_3]$ is negligible in κ because of the collision resistance of the random oracle. Define $\mathsf{bad} := \bigcup_i \mathsf{bad}_i$. By union bound we have $\Pr[\mathsf{bad}] \leq \mathsf{negl}(\kappa)$. Hence, without loss of generality, in the rest of the analysis, we assume that bad did not occur. We now show the following technical claim:

Claim 1. *Suppose the adversary made at most $\kappa \cdot 2^{d-4}$ scratch queries to the oracle. Then, the probability that at least $3\kappa/4$ queries are successful is negligible in κ.*

Proof. Suppose that the adversary made at least $3\kappa/4$ successful queries. Since H was computed by the multi-party random oracle, thus, the adversary has no information about the Y^ℓ. Moreover, since H is a random function, thus without loss of generality we can assume that $\mathcal{A}$ submits to the oracle scratches in a fixed order. Say: for every X^ℓ she submits them in order $(X^\ell || \text{"0"}), (X^\ell || \text{"1"}), \ldots, (X^\ell || \text{"}2^d\text{"})$ (where "m" denotes the binary representation of m of length d).

Let $\mathcal{L} \subseteq [\ell]$ be the set of indices such that Y^ℓ is a binary representation of a number that is less than 2^{d-2}, i.e., those ℓ where the adversary is "lucky" in the sense that she will find Y^ℓ with at most 2^{d-2} scratches. Since we assumed that bad_2 did not occur, thus the sets of scratches for each index ℓ are pairwise disjoint. For every $\ell \in \mathcal{L}$, the adversary can be lucky and get a successful query already for one scratch. On the other hand, for every $\ell \notin \mathcal{L}$ the adversary needs to make at least 2^{d-2} scratches in order to have a successful query. Since the total number of successful queries is at least $3\kappa/4$, thus the total number of scratches is at least

$$\begin{aligned} & |\mathcal{L}| \cdot 1 + (3\kappa/4 - |\mathcal{L}|) \cdot 2^{d-2} \\ \geq \quad & (3\kappa/4 - |\mathcal{L}|) \cdot 2^{d-2}. \end{aligned} \quad (7)$$

On the other hand, we assumed that the number of scratches is at most $\kappa \cdot 2^{d-4}$. We hence get that Eq. (7) is at most $\kappa \cdot 2^{d-4}$, which can be rearanged to

$$|\mathcal{L}| \geq \kappa/2. \quad (8)$$

Since we assumed that the event bad_3 holds, thus there are no collions in the hash function, and therefore for each ℓ the probability that $\ell \in \mathcal{L}$ is exactly equal to $1/4$ and these events are independent.[7] Let S_ℓ be a binary variable equal to 1 iff and only if $\ell \in \mathcal{L}$. Clearly $|\mathcal{L}| = \sum_\ell S_\ell$, and by the Chernoff bound[8] we get the following bound on Eq. (8):

$$\Pr[|\mathcal{L}| \geq \kappa/2] \leq \exp(-\kappa/8) = \mathsf{negl}(\kappa).$$

This finishes the proof of the claim. □

We now go back to the proof of punishability. Let $\mathcal{G}$ be the event that at least $3\kappa/4$ queries are successful, and let $\mathcal{H}$ be the event that there was no fast scratch query. We are interested in the joint event $\mathcal{G} \wedge \mathcal{H}$. This is the event that the adversary makes at least $3\kappa/4$ successful queries, and all of them are done via slow queries. Since we defined the difficulty as $d := \lceil \log_2(\sigma/\kappa) \rceil + 4$, hence $\sigma \leq \kappa \cdot 2^{d-4}$. Thus, using the assumption that bad_1 did not occur (i.e. no slow scratch query was submitted in the pre-computation phase), and the claim from above, we obtain:

$$\Pr[\mathcal{G} \text{ and } \mathcal{H}] \leq \mathsf{negl}(\kappa). \quad (9)$$

It is easy to see that if less than $3\kappa/4$ queries were successful, then the probability that the adversary guesses the entire $(Y^1[1], \ldots,$

[7] Recall that the order in which the adversary makes is fixed and hence the probability that the adversary "hits" the value Y^ℓ in her first 2^{d-2} queries is $1/4$.

[8] We use the following version of this bound (see, e.g., [27, Theorem 4.5]). 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 $\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/2}$.

$Y^\kappa[1]$) correctly is negligible in κ, and hence

$$\Pr[\neg\mathcal{G} \text{ and } \mathsf{Broken}_\oplus] \le \mathsf{negl}(\kappa). \tag{10}$$

Combining (9) and (10) we obtain $\Pr[\mathcal{H} \text{ and } \mathsf{Broken}_\oplus] \le \mathsf{negl}(\kappa)$, and consequently $\Pr[\mathsf{Broken}_\oplus] \le \Pr[\neg\mathcal{H}] + \mathsf{negl}(\kappa)$. On the other hand, if $\mathcal{H}$ did not occur, then there was at least one fast scratch query, i.e., query starting with X^ℓ (for some ℓ). Since $H(X^\ell \,||\, Y^\ell) = Z^\ell$, thus in the punishing phase, the party who made that query has $X^\ell \,||\, Y^\ell$ in her fast transcript, and consequently (if she is the snitcher), then the dealer sends $(\mathsf{punish}, i, X^\ell, Y^\ell, \ell)$ to the judge. Note that, since we assumed that bad_3 did not occur, thus there are collisions among the Z^ℓ, so the party can infer ℓ from Z^ℓ. The judge, in turn, accepts this message and outputs $(\mathsf{punished}, \mathsf{P}_i)$. Therefore, (6) (and consequently (4)) is proven.

3.2.3 Unframability. Recall that, for a fixed honest shareholder P_{i^*}, the "punish" message submitted to the judge J by the dealer D, necessary to frame the shareholder P_{i^*}, is a string $Q \in \{0,1\}^\kappa$ such that $H(Q \,||\, Y^\ell) = Z^\ell$ for some $\ell = 1, \ldots, \kappa$. Recall that Z^ℓ is computed using a multi-party random oracle as $Z^\ell := H\left((X_1^\ell \oplus \cdots \oplus X_n^\ell) \,||\, Y^\ell\right)$. Since the X_{i^*} is chosen at random by P_{i^*}, thus $X^\ell := X_1^\ell \oplus \cdots \oplus X_n^\ell$ is uniformly random. Moreover, since Z_ℓ is computed using the multi-party random oracle, thus X^ℓ is uniformly random from the point of the adversary. Therefore, his probability of guessing any Q such that $H(Q \,||\, Y^\ell) = Z^\ell$ is negligible in κ. □

3.3 Discussion

3.3.1 Efficiency analysis. In our analysis, we focus on the number of slow and fast queries to Ω_H as our main measure of efficiency. Unlike the previous work on Individual Cryptography [15], our scheme assumes that the honest parties can compute a few hashes in MPC while still relying on the hardness of computing hashes in MPC for the proof of security. In the sharing phase, the number of slow queries made by the parties is κ; Additionally, the dealer makes an extra fast query. The reconstruction procedure requires an expected $\kappa 2^{d-1}$ queries to Ω_H, which will be fast in the case of the honest chair in the reconstruction procedure and slow in the case of a distributed adversary, as even a single fast scratch attempt would already render the adversary punishable. This means that the ratio between the number of queries that must be computed by the distributed adversary and the parameter #slow in the honest execution is roughly $2^{d-1} \approx 2^{\log_2 \sigma - \log_2 \kappa + 3} = 8\sigma/\kappa$. If we take $\kappa = 2\log_2 \sigma$ (cf. the statement of Theorem 3.1), we obtain the ratio to be $4\sigma/\log_2 \sigma$. Finally, let us notice, that the judge requires only a single query to the oracle for verification of the fraud proof.

Since we do not consider the randomness necessary to implement a multi-party random oracle, it is easy to see that each shareholder requires κ^2 random bits and the dealer needs κd random bits, which is the randomness complexity of the protocol. It is also easy to see that the size of the output is $\Theta(\kappa^2)$ per party.

4 The t-out-of-n SSS scheme

In this section, we construct a t-out-of-n SSS scheme (for any $t \le n$) from the 2-out-of-2 SSS scheme $\Pi_\oplus$ constructed in the previous section (we assume that $\Pi_\oplus$ works for secrets of length κ) and a t-out-of-n Shamir's secret sharing scheme $\left(\mathsf{share}_{(t,n)}^{\mathsf{Shamir}}, \mathsf{rec}_{(t,n)}^{\mathsf{Shamir}}\right)$.

By $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$-*instance of* $\Pi_\oplus$ we mean an instance of $\Pi_\oplus$ with P_α and P_β taking the roles of the shareholders P_1 and P_2, respectively. The judge in such an instance is denoted J_α^β.

4.1 Our construction

Similarly to the n-out-of-n case, our t-out-of-n scheme $\Pi_{(t,n)}$ consists of phases: $\Pi.\mathsf{share}_{(t,n)}$, $\Pi.\mathsf{punish}_{(t,n)}^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_\gamma}$, and $\Pi.\mathsf{reconstruct}_{(t,n)}$, presented on Figs. 6 to 8. The main idea behind this construction is

$\Pi.\mathsf{share}_{(t,n)}$ between $\mathsf{D}, \mathsf{P}_1, \ldots, \mathsf{P}_n$ and J

1. All parties take as input 1^κ. Additionally, the dealer D takes as input a secret $S \in \{0,1\}^{\lambda(\kappa)}$.
2. The dealer samples a random $R \leftarrow\!\!\$\ \{0,1\}^\kappa$.
3. The dealer D runs a Shamir secret sharing procedure $(S_1, \ldots, S_n) \leftarrow\!\!\$\ \mathsf{share}_{(t,n)}^{\mathsf{Shamir}}(R)$.
4. For $\alpha := 1$ to n:
   (1) The dealer D runs a t-out-of-n Shamir secret sharing procedure $(S_\alpha^1, \ldots, S_\alpha^n) \leftarrow\!\!\$\ \mathsf{share}_{(t,n)}^{\mathsf{Shamir}}(S_\alpha)$.
   (2) For $\beta := 1$ to n the parties $\mathsf{D}, \mathsf{J}, \mathsf{P}_\alpha$, and P_β run the $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$-instance of $\Pi_\oplus$, that is, they execute the $\Pi.\mathsf{share}_\oplus$ phase with P_α and P_β taking the roles of the shareholders P_1 and P_2, respectively, J taking the role of the judge, D taking the role of the dealer with secret input S_α^β. We will also denote the judge corresponding to the $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$-instance (who is being run by J as a subprocedure) as J_α^β.
5. If any of the $\Pi.\mathsf{share}_\oplus$ procedures above failed, then the dealer outputs error and the protocol terminates.
6. Otherwise, the dealer D sends $C := G(R) \oplus S$ to the shareholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$.

Figure 6: The sharing phase of the SSS scheme $\Pi_{(t,n)}$.

as follows. Just as in the n-out-of-n scheme, the dealer first shares a random secret R with the shareholders and then uses it to encrypt the message X using the pad $G(R)$.

Naively, one could attempt to simply run a t-out-of-n sharing on R, obtaining the shares $R_1, \ldots, R_n$ and then share both R_i and R_j between P_i and P_j using the 2-out-of-2 sharing $\Pi.\mathsf{share}_\oplus$ for each $i, j \in [n]$. However, this does not provide protection against coalitions of size as low as 2, since every party only needs to contact one party to recover its share. Once the shares have been recovered, t parties can easily collude to recover the secret R without risking to get punished. Therefore, we add another layer of secret sharing to ensure that every party has to reconstruct at least $t-1$ instances of $\Pi_\oplus$. Under the assumption that each coalition is of size less than t, this implies that an adversary will have to reconstruct at least t instances of $\Pi_\oplus$ between the shareholders who do not share a coalition.

In our solution, each pair $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$ of shareholders executes an instance of $\Pi_\oplus$ *twice*: once P_α and P_β playing the roles of P_1 and P_2 (respectively), and the second time with the roles reversed. The punishment procedure (in Fig. 7) takes this into account by running

$\Pi.\mathsf{punish}^{\mathcal{A}_j \rightsquigarrow \mathsf{P}_\gamma}_{(t,n)}$ **between** D **and** J

1. The dealer receives the fast-oracle transcript $\mathcal{T}_j^{\mathsf{fast}}$ of $\mathcal{A}_j$.
2. For each $\alpha \in \{1, \ldots, n\}$ the following is executed:
   (1) The dealer runs the $\Pi.\mathsf{punish}_\oplus$ procedure of P_i in the $(\mathsf{P}_\alpha, \mathsf{P}_\gamma)$-instance on transcript $\mathcal{T}_j^{\mathsf{fast}}$. If this subprocedure produces a message $(\mathsf{punish}, 2, Q, Y^\ell, \ell)$ to the judge, then the dealer sends $(\mathsf{punish}, \alpha, \gamma, 2, Q, Y^\ell, \ell)$ to the judge J.
   (2) For each message $(\mathsf{punish}, \alpha, \gamma, 2, Q, Y^\ell, \ell)$ received from the dealer, the judge J sends $(\mathsf{punish}, 2, Q, Y^\ell, \ell)$ to J^γ_α.
   (3) If J^γ_α outputs $(\mathsf{punished}, \mathsf{P}_2)$, then J outputs $(\mathsf{punished}, \mathsf{P}_\gamma)$.
3. Symmetrically, for each $\beta \in \{1, \ldots, n\}$ the following is executed:
   (1) The dealer runs the $\Pi.\mathsf{punish}_\oplus$ procedure of P_i in the $(\mathsf{P}_\gamma, \mathsf{P}_\beta)$-instance on transcript $\mathcal{T}_j^{\mathsf{fast}}$. If this subprocedure produces a message $(\mathsf{punish}, 1, Q, Y^\ell, \ell)$ to the judge, then the dealer sends $(\mathsf{punish}, \gamma, \beta, 1, Q, Y^\ell, \ell)$ to the judge J.
   (2) For each message $(\mathsf{punish}, \gamma, \beta, 1, Q, Y^\ell, \ell)$ received from the dealer, the judge J sends $(\mathsf{punish}, 1, Q, Y^\ell, \ell)$ to J^β_γ.
   (3) If J^β_γ outputs $(\mathsf{punished}, \mathsf{P}_1)$, then J outputs $(\mathsf{punished}, \mathsf{P}_\gamma)$.

Figure 7: The punishment of P_γ snitched by $\mathcal{A}_j$ in $\Pi_{(t,n)}$.

$\Pi.\mathsf{reconstruct}_{(t,n)}$ **between** $\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_t}$

Let $I = \{i_1, \ldots, i_t\}$.
1. For $\alpha, \beta \in I$, the parties P_α and P_β run $\Pi.\mathsf{reconstruct}_\oplus$ on the $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$-instance and obtain S^β_α.
2. For each $\alpha \in I$, the parties P_i for $i \neq \alpha$ send their values of S^i_α to P_α, who in turn reconstructs S_α from $S^{i_1}_\alpha, \ldots, S^{i_t}_\alpha$.
3. The parties choose a chair $\mathsf{P} \in \{\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_t}\}$. Each party P_i for $i \in I$ send its share S_i to P.
4. The chair reconstructs R from $S_{i_1}, \ldots, S_{i_t}$, computes $\widehat{S} := G(R) \oplus C$ and sends it to the P_i for $i \in I$.
5. Each party outputs $\widehat{S}$.

Figure 8: The reconstruction phase of the SSS scheme $\Pi_{(t,n)}$.

two "symmetric" procedures described in Steps 2 and 3 (think of them as going through the column and through the row of the matrix on Fig. 9). If the dealer discovers fraud proof in any of these steps, he forwards it to the judge (labeling it appropriately). Note that in our model, the hash transcripts are not bound to specific $\Pi_\oplus$ instances, and hence, the dealer receives just one transcript that needs to be scanned for each instance independently. In practice, this can be optimized by performing just one scan that looks for the fraud proofs in parallel. We choose not to do it in order to keep our scheme modular.

4.2 Security analysis

Theorem 4.1. *Consider fixed n and t and let σ be a function of κ such that $\kappa \geq 2\log_2(\sigma/t)$. Let $\Pi_\oplus$ be the SSS protocol (constructed on Figs. 3 to 5) with $d := \lceil \log_2(\sigma/(\kappa t)) \rceil + 4$. [9] Then the scheme $\Pi_{(t,n)}$ (constructed on Figs. 6 to 8) is a secure t-out-of-n SSS scheme with the slow query budget σ.*

Proof. In order to prove the theorem, we need to argue that Π is correct, punishable, and unframable.

4.2.1 Correctness. 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^β_α in the reconstruction algorithm are reconstructed to values S^β_α 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.

4.2.2 Punishability. Fix a distributed adversary $\mathcal{A}$ and assume that each coalition consists of less than t parties. We need to show that: $\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi_{(t,n)}; \kappa]\!] - \mathsf{negl}(\kappa) \leq \Pr\big[\mathsf{Punished}[\![\mathcal{A}, S; \kappa]\!]\big]$. First, consider the $\mathsf{Sharing}[\![\mathcal{A} \leftrightarrows \Pi_{(t,n)}, b; \kappa]\!]$ experiment. Similarly, as in the proof of punishability of the $\Pi_\oplus$ scheme, 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: $\mathsf{Adv}[\![\mathcal{A} \leftrightarrows \Pi_{(t,n)}; \kappa]\!] \leq \Pr[\mathsf{Broken}]$. Similar to what we did in the proof of Theorem 3.1 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 X^ℓ used in the instances of $\Pi_\oplus$, which we will furthermore assume. From now on, let us assume that none of these unlikely events have happened.

For any $\alpha, \beta \in [n]$, let R^β_α be the value R computed in the sharing algorithm of the $\Pi_\oplus$ scheme (cf. Fig. 3) and let $\mathsf{Broken}^\beta_\alpha$ be the event that the preimage of R^β_α was submitted to the oracle. In other words, $\mathsf{Broken}^\beta_\alpha$ means the event $\mathsf{Broken}_\oplus$ occurred in the $(\mathsf{P}_\alpha, \mathsf{P}_\beta)$-instance of $\Pi_\oplus$ and let $\mathsf{Fast}^\beta_\alpha$ be the event that there was a fast scratch query in that instance. Furthermore, let Broken_α be the event that $\mathsf{Broken}^\beta_\alpha$ occurred for α, β such that P_α and P_β do not share a coalition. Finally, let $\mathsf{Broken}(t)$ be the event that Broken_α occurred for at least t distinct values of α. 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 3.2.2, still holds even if the adversary controls some parties who are not shareholders but might participate in a reconstruction attempt.

[9] This value of d corresponds to instantiating $\Pi_\oplus$ with a slow query budget $\sigma' := \sigma/t$ per instance, cf. Theorem 3.1.

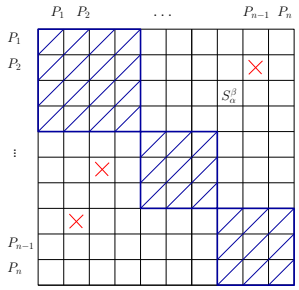

Figure 9: An example of illegal reconstruction in the threshold scheme. The blue rectangles on the diagonal describe coalitions. Within each coalition, the parties can freely recover the S^{β}_{α}. The red cross marks denote the events $\mathsf{Broken}^{\beta}_{\alpha}$.

Note that clearly, the sets of scratches are made in different instances of the $\Pi.\mathsf{share}_{\oplus}$ procedure are with overwhelming probability disjoint since the X^{ℓ}_i are sampled freshly. 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^{β}_{α}, i.e., $\Pr[\mathsf{Broken}(t)] \leq \Pr\left[\exists\alpha\exists\beta\colon \mathsf{Fast}^{\beta}_{\alpha}\right] + \mathsf{negl}(\kappa)$. It is easy to see that the above still holds in the $\mathsf{Snitching}[\![S, \mathcal{A} \leftrightarrows \Pi_{(t,n)}; \kappa]\!]$ experiment, as argued in Section 3.2.2 and that $\mathsf{Fast}^{\beta}_{\alpha}$ implies that either of the parties P_{α} and P_{β} is punishable.

4.2.3 Unframability. Fix an honest party P_{i^*}. Suppose that a monolithic adversary $\mathcal{M}^0$ corrupts the dealer D and some shareholders $\mathsf{P}_{j_1}, \ldots, \mathsf{P}_{j_m}$. 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 X^{ℓ} 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. □

4.3 Discussion

4.3.1 Efficiency analysis. We will neglect the constants in the following analysis and focus on the asymptotic properties. Since the threshold scheme does not explicitly compute hashes and the execution of $\Pi_{(t,n)}$ requires $\Theta(n^2)$ executions of $\Pi_{\oplus}$, each of the shareholders P_i have to make $\Theta(n\kappa)$ slow queries, whereas the dealer D needs to make $\Theta(n^2\kappa)$ slow queries, as per Section 3.3.1. Since the reconstruction of each 2-out-of-2 instance requires an expected $\kappa 2^{d-1}$ queries to Ω_H, and $2^{d-1} = 8\sigma/(\kappa \cdot t)$, the full reconstruction in the t-out-of-n scheme requires $\Theta(t\sigma)$ queries. In an honest reconstruction, all of them will be fast. In an adversarial setting, the number of slow queries will depend on the maximum size of the coalition k. We will analyze two extreme cases: $k = 1$ and $k = t - 1$. In any case, the adversary needs to reconstruct $\Theta(t^2)$ instances of $\Pi_{\oplus}$. If $k = 1$, then all of them have to be slow, which yields $\Theta(t\sigma)$ slow queries. If $k = t - 1$, then the adversary can use fast queries for the instances shared by the coalition members and only needs to reconstruct t of them using slow queries, which amounts to $\Theta(\sigma)$ slow queries. The ratio between the number of queries in the dishonest execution and the parameter #slow in the honest execution is $\Theta(t/n \cdot \sigma/\kappa)$ for $k = 1$ and $\Theta(1/n \cdot \sigma/\kappa)$ for $k = t - 1$. Assuming $\kappa = 2\log_2 \sigma$, as previously, this amounts to $\Theta(t/n \cdot \sigma/\log_2 \sigma)$ and $\Theta(1/n \cdot \sigma/\log_2 \sigma)$, respectively.

4.3.2 The parameter σ. The punishability would still hold for the threshold scheme, even if we used the underlying scheme $\Pi_{\oplus}$ with any parameter σ', such that $\sigma/t \leq \sigma' \leq \sigma$. Choosing $\sigma' = \sigma/t$ is optimal in the sense that it enables lower values of κ for a given slow query bound σ or, equivalently, improves the maximum number of slow queries σ that a fixed security parameter κ protects against.

5 Conclusions and open problems

Our novel primitive of *secret sharing with snitching* (SSS) provides a way to discourage collusion in secret-sharing schemes by introducing a threat of punishment for the colluding parties. We base our construction on the assumption that computing hashes in MPC is moderately hard, i.e., that massive hash computations in MPC are infeasible while allowing the parties to compute a few hashes in MPC. We also introduced a generalization of the random-oracle model, which allows modeling hash evaluations made inside MPC. We leave it as an open question to what extent concrete hash functions.

6 Acknowledgments

This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666), by an NCN Grant 2019/35/B/ST6/04138, and by the Copernicus Awards (agreement no. COP/01/2020). In addition, 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 the Copernicus Award (INST 18989/419-1). We would also like to thank the anonymous ACM CCS reviewers for their helpful comments.

References

[1] Joël Alwen, Abhi Shelat, and Ivan Visconti. 2008. 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 (Lecture Notes in Computer Science, Vol. 5157)*, David A. Wagner (Ed.). Springer, 497–514. https://doi.org/10.1007/978-3-540-85174-5_28

[2] Leemon Baird, Pratyay Mukherjee, and Rohit Sinha. 2022. i-TiRE: Incremental Timed-Release Encryption or How to use Timed-Release Encryption on Blockchains?. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 235–248. https://doi.org/10.1145/3548606.3560704

[3] Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. 2021. SoK: Mitigation of Front-running in Decentralized Finance. *IACR Cryptol. ePrint Arch.* (2021), 1628. https://eprint.iacr.org/2021/1628

[4] Mihir Bellare and Phillip Rogaway. 1993. 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*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM, 62–73. https://doi.org/10.1145/168588.168596

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. 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*, Janos Simon (Ed.). ACM, 1–10. https://doi.org/10.1145/62212.62213

[6] Dan Boneh, Aditi Partap, and Lior Rotem. 2023. Accountability for Misbehavior in Threshold Decryption via Threshold Traitor Tracing. Cryptology ePrint Archive, Paper 2023/1724. https://eprint.iacr.org/2023/1724 https://eprint.iacr.org/2023/1724.

[7] Dan Boneh, Aditi Partap, and Lior Rotem. 2024. Traceable Secret Sharing: Strong Security and Efficient Constructions. Cryptology ePrint Archive, Paper 2024/405. https://eprint.iacr.org/2024/405 https://eprint.iacr.org/2024/405.

[8] Vitalik Buterin. 2022. *Proof of stake: The making of Ethereum and the philosophy of blockchains.* Seven Stories Press.

[9] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, Janos Simon (Ed.). ACM, 11–19. https://doi.org/10.1145/62212.62214

[10] Benny Choc, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. VERIFIABLE SECRET SHARING AND ACHIEVING SIMULTANEITY IN THE PRESENCE OF FAULTS.. In *Annual Symposium on Foundations of Computer Science (Proceedings)*. 383–395.

[11] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *J. ACM* 45, 6 (1998), 965–981. https://doi.org/10.1145/293347.293350

[12] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 44–75. https://doi.org/10.1007/978-3-030-45721-1_3

[13] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. 2015. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB&isbn=9781107043053

[14] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 910–927. https://doi.org/10.1109/SP40000.2020.00040

[15] Stefan Dziembowski, Sebastian Faust, and Tomasz Lizurej. 2023. 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 (Lecture Notes in Computer Science, Vol. 14082)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 547–579. https://doi.org/10.1007/978-3-031-38545-2_18

[16] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102. https://doi.org/10.1145/3212998

[17] Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. 2013. Rational Protocol Design: Cryptography against Incentive-Driven Adversaries. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 648–657. https://doi.org/10.1109/FOCS.2013.75

[18] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, 281–310. https://doi.org/10.1007/978-3-662-46803-6_10

[19] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, Alfred V. Aho (Ed.). ACM, 218–229. https://doi.org/10.1145/28395.28420

[20] Tiantian Gong, Ryan Henry, Alexandros Psomas, and Aniket Kate. 2023. More is Merrier: Relax the Non-Collusion Assumption in Multi-Server PIR. arXiv:2201.07740 [cs.CR] https://arxiv.org/abs/2201.07740

[21] Joseph Y. Halpern and Vanessa Teague. 2004. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, László Babai (Ed.). ACM, 623–632. https://doi.org/10.1145/1007352.1007447

[22] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. 2019. SoK: General Purpose Compilers for Secure Multi-Party Computation. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 1220–1237. https://doi.org/10.1109/SP.2019.00028

[23] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. 2023. Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets. *IACR Cryptol. ePrint Arch.* (2023), 44. https://eprint.iacr.org/2023/044

[24] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[25] Matt Lepinski, Silvio Micali, and Abhi Shelat. 2005. Collusion-free protocols. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, Harold N. Gabow and Ronald Fagin (Eds.). ACM, 543–552. https://doi.org/10.1145/1060590.1060671

[26] Easwar Vivek Mangipudi, Donghang Lu, Alexandros Psomas, and Aniket Kate. 2023. Collusion-Deterrent Threshold Information Escrow. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*. IEEE, 584–599. https://doi.org/10.1109/CSF57540.2023.00010

[27] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, USA.

[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[29] R. L. Rivest, A. Shamir, and D. A. Wagner. 1996. *Time-lock Puzzles and Timed-release Crypto*. Technical Report. USA.

[30] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. 1994. How to share a function securely. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, Frank Thomson Leighton and Michael T. Goodrich (Eds.). ACM, 522–533. https://doi.org/10.1145/195058.195405

[31] Okke Schrijvers, Joseph Bonneau, Dan Boneh, and Tim Roughgarden. 2016. Incentive Compatibility of Bitcoin Mining Pool Reward Functions. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9603)*, Jens Grossklags and Bart Preneel (Eds.). Springer, 477–498. https://doi.org/10.1007/978-3-662-54970-4_28

[32] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613. https://doi.org/10.1145/359168.359176

[33] Abhi Shelat. 2010. 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*, Moshe Dror and Greys Sosic (Eds.). ACM, 91:1. https://doi.org/10.1145/1807406.1807497

[34] Wikipedia contributors. 2023. Trusted execution environment — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Trusted_execution_environment&oldid=1153071242 [Online; accessed 5-May-2023].

[35] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 160–164. https://doi.org/10.1109/SFCS.1982.38