Individual Cryptography*

Stefan Dziembowski[1,2][0000−0002−6914−6425],
Sebastian Faust[3][0000−0002−8625−4639], and
Tomasz Lizurej[1,2][0000−0001−8563−4325]

[1] University of Warsaw
[2] IDEAS NCBR
[3] TU Darmstadt

Abstract. We initiate a formal study of *individual cryptography*. Informally speaking, an algorithm Alg is *individual* if, in every implementation of Alg, there always exists an individual user with full knowledge of the cryptographic data S used by Alg. In particular, it should be infeasible to design implementations of this algorithm that would hide S by distributing it between a group of parties using an MPC protocol or outsourcing it to a trusted execution environment.
We define and construct two primitives in this model. The first one, called *proofs of individual knowledge*, is a tool for proving that a given message is fully known to a single ("individual") machine on the Internet, i.e., it cannot be shared between a group of parties. The second one, dubbed *individual secret sharing*, is a scheme for sharing a secret S between a group of parties so that the parties have no knowledge of S as long as they do not reconstruct it. The reconstruction ensures that if the shareholders attempt to collude, one of them will learn the secret entirely. Individual secret sharing has applications for preventing collusion in secret sharing. A central technique for constructing individual cryptographic primitives is the concept of MPC hardness. MPC hardness precludes an adversary from completing a cryptographic task in a distributed fashion within a specific time frame.

1 Introduction

Multiparty computation (MPC) [6, 12, 20] is a powerful cryptographic technique that enables parties to evaluate any function securely in the presence of an adversary. It guarantees that nothing is revealed about the honest parties' inputs except what can be learned from the function's output. MPC protocols have found countless applications in cryptography and are one of the main tools for achieving privacy. In addition, MPC technology is becoming widely available in practice, e.g., for machine learning and blockchain applications. While MPCs are traditionally used for the "good" with their increasing availability in practice, there is a danger that an adversary misuses this technology to carry out malicious tasks. Let us illustrate this with two examples.

Identity sharing over the Internet. Imagine a service provider $\mathcal{S}$ that maintains a system in which individual users $\mathcal{U}$ can open accounts by paying a subscription fee. To lower this fee, the malicious users decide to open a *single* account and to share the credentials S to it between each other. In this paper, we are interested in situations when these users are individuals $\mathcal{A}_1, \ldots, \mathcal{A}_a$ connected via the Internet that do not trust each other, i.e., we are *not* concerned about scenarios in which the credentials are shared between different devices that belong to a single person, between members of one family, or between devices that are located physically very close to each other (so the network connection speed does not matter).

* This result is part of a project that received funding from the European Research Council (ERC) under the European Union's *Horizon 2020* and *Horizon Europe* research and innovation programs (grants PROCONTRA-885666 and CRYPTOLAYER-101044770). This work was also partly supported by the National Science Centre, Poland, under research project No. 463393, by the German Research Foundation (DFG) via the DFG CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

Suppose that to discourage such users from simply sharing S in plaintext, the service provider integrates ad-hoc countermeasures such that knowledge of S suffices to damage the account significantly. For example, if the "account" is a cryptocurrency address, then the knowledge of S should suffice to drain the account from all the coins. Alternatively, if it is an online storage system, then knowledge of S should permit deleting all the files. While these countermeasures should suffice to discourage the users from sharing S in plaintext, they are not sufficient to protect against a more sophisticated *identity sharing attack*, where the malicious users share S using secret sharing and jointly emulate a single "virtual" user $\mathcal{U}$ using an MPC protocol to authenticate with $\mathcal{S}$.

The identity-sharing attack is similar to the "identity-lease attack" recently introduced by Puddu et al. [29]. In an identity-lease attack, the attacker's primary goal is to temporarily outsource ("lease") its identity to a third party, who can control it for a specific time or purpose. As discussed in [29], this may, for instance, be problematic in electronic voting, where identity leasing can be used to sell votes. While Puddu et al. rely on a trusted execution environment for their attack, it is possible to carry out the same attack using an MPC committee. Here, the committee holds secret shares of the user's identity and is queried by the third party to carry out the desired task (e.g., vote for a certain party in an electronic election).

Collusion in secret sharing. Threshold secret sharing is a fundamental primitive in cryptography. It allows an honest dealer to share a secret S among a set of parties $\mathcal{P}_1, \ldots, \mathcal{P}_a$ such that any subset of $t-1$ parties learns nothing about S, while a subset of t parties suffices to recover S. To this end, we typically require that at most $t-1$ parties are malicious, while the remaining parties are honest and do not collude. However, the latter may be unrealistic, particularly when collusion can go undetected and thus cannot easily be financially punished. For example, in standard secret sharing schemes, nothing prevents a set of t parties from simply exchanging their shares to recover the secret S. As discussed in the next paragraph, a trivial attempt to de-incentivize such an attack can be broken by an MPC protocol.

Consider a setting where the dealer wishes that the shareholders only release the secret after a certain time T. This setting is often referred to as "encryption to the future" and has gained some interest due to applications for blockchains [10, 22]. To prevent the simple collusion attack from above, the honest dealer shares $S' := (S, X)$, where if some party $\mathcal{P}_i$ gets to know X before time T, she is able to punish all other shareholders (e.g., this may be done via a smart contract on a blockchain).[4] While this approach de-incentivizes our naive collusion attack from above, it can easily be attacked using an MPC protocol. More concretely, instead of just recovering the entire S' by exchanging their secret shares, the parties run an MPC, which allows them to stripe off X before learning S.

MPC hardness. By closely examining the two previous examples, we make the following crucial observation. Both attacks rely on the assumption that a distributed adversary can efficiently evaluate the cryptographic task via the MPCs. Hence, to thwart these attacks, our key idea is to make these cryptographic tasks *MPC-hard*. Informally, we say that a task/function is MPC-hard if executing it securely in a distributed way takes a significant amount of time. This implies that if a cryptographic task is MPC-hard, then in order to run it efficiently, the parties need to execute it *individually*. We formalize this new notion through a concept that we call *individual cryptography*.

Let us take a look at how MPC hardness may help us to prevent the previous two attacks. In the case of the identity sharing example, the *distributed adversary*, i.e., a tuple $\mathcal{A}_1, \ldots, \mathcal{A}_a$ of interactive machines (also called the *sub-adversaries*), uses an MPC protocol to ensure that no party individually knows the credentials to authenticate with a service provider $\mathcal{S}$. Suppose the service provider will only accept an authentication attempt if it is completed within a certain time frame. The distributed adversary now has two options. Either it runs the authentication process via the MPC. This, however, will fail due to MPC hardness. Alternatively, the adversary may ask one of the sub-adversaries, say $\mathcal{A}_j$, to execute the task *individually*, in which case $\mathcal{A}_j$ has to know the credential S entirely.[5] We formalize this concept via a new primitive that we call *Proof of*

[4] We note that a similar (but more involved) idea has been used in a recent work of Mangipudi et al. to construct a collusion-deterrent threshold information escrow [27].

[5] Recall that in this case, the sub-adversary $\mathcal{A}_j$ can take full control over the account of the user, which was something that a malicious user tries to avoid.

Individual Knowledge (PIK). Informally, a PIK guarantees that the prover must know the entire secret if it wants to get accepted by a verifier. We will provide further details on PIKs in Sect. 1.2. In the secret sharing example, the distributed adversary uses the MPC to compute some non-trivial function of the shared secret S before time T. Here, we require that the reconstruction algorithm of the secret sharing scheme is MPC hard. Informally, this guarantees that the distributed adversary cannot complete the reconstruction before some time T unless one of the sub-adversaries say $\mathcal{A}_j$, runs the reconstruction individually. We call such a secret sharing scheme an *individual secret sharing (ISS)* and present more details on it in Sect. 5.

Attacks via the trusted execution environment (TEE). An alternative way to perform the aforementioned attacks is to use trusted execution environments (such as Intel's SGX) to accelerate the MPC (see, e.g., [2]) or to outsource secret storage in a way similar to the one described in [29]. To make our schemes secure against such attacks, we need to make some additional assumptions about what kind of fast computation is infeasible in TEEs. One option is to assume that the honest users are equipped with hardware similar to Bitcoin mining rigs that can compute a massive amount of hashes in parallel (note that our PIK protocol is based on massive parallel computation, while the ISS uses sequential computation). Another option is to come up with new hash functions that are infeasible to compute quickly on TEEs (e.g., due to large memory requirements). We leave it as future work to design such hash functions.

1.1 Informal description of our model

As standard in cryptographic modeling, we have to describe the adversarial model (i.e., the adversaries' abilities) and specify what it means for a cryptographic task to be secure. Let us start with the adversarial model.

The adversarial model. As our MPC-hard functions will be based on massively evaluating a hash function, we give the distributed adversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$ access to a special oracle Ω_H that allows evaluating a fixed input-length hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$. The oracle accepts queries of the form $(x, mode)$, where $x \in \{0,1\}^\lambda$ and $mode \in \{\mathsf{fast}, \mathsf{slow}\}$. If $mode = \mathsf{fast}$, then a query is called *fast*. It is called *slow* otherwise. Let us give some intuition behind these two modes.

The fast queries are hash function evaluations that a sub-adversary $\mathcal{A}_b$ runs locally. We assume that these evaluations can be done very fast (orders of magnitude faster than slow queries). For example, a party may execute them using a specially designed ASIC, such as used in the context of Bitcoin mining. We assume that the number of fast queries is only bounded by the running time of the adversary, which is polynomial time. On the other hand, the slow queries model an evaluation of the hash function using an MPC protocol. In particular, this means that the sub-adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_a$ can learn $H(x)$ without knowing x. Since the evaluation of a hash function using MPC technology is conceivable much slower than using an ASIC, we assume that the budget of the adversary for such queries is comparably small, i.e., bounded by some parameter σ.

In addition to bounding the number of slow queries that the sub-adversaries may ask for, in our PIK application, we put an additional restriction on the sub-adversaries. We require that they run in at most ρ communication rounds. Notice that we do not require any bounds on the communication complexity as at the end of each round, we allow the sub-adversaries to share any information that they currently possess.

Attacks exploiting the hash function structure. We need to stress that in the model above, the hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ needs to be chosen carefully, and it is unrealistic to assume that λ is large. It is also important that H does not have a structure that the adversary could exploit. A natural choice for H is a compression function of a popular hash function (for example, in SHA-256, the compression function is of a type $H : \{0,1\}^{728} \to \{0,1\}^{256}$).

We illustrate this by the following example. Suppose H is a Merkle-Damgard-type hash function, and let $G : \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ be the compression function that H is built from, i.e., for messages of a form $(S_1 \,||\, S_2 \,||\, W)$ (with $|S_1| = |S_2| = |W| = \kappa$) we have $H(S_1 \,||\, S_2 \,||\, W) = G(G(G(IV \,||\, S_1) \,||\, S_2) \,||\, W)$ (where $IV \in \{0,1\}^\kappa$ is some fixed initial value, and for simplicity, we omit the last block encoding message length).

Now, suppose that we want to verify if a message S consisting of two blocks (S_1, S_2) (both of length κ) is stored on one machine by forcing the prover to compute a large number of hashes of a form $H(S, W)$ (for multiple W's), which is the case for our PIK construction (see below). Taking into account the structure of H, two sub-adversaries: $\mathcal{A}_1$ and $\mathcal{A}_2$ can now "split" the computation as follows: the $\mathcal{A}_1$ first computes $h = G(IV, S_1)$, and then sends h to $\mathcal{A}_2$ who can compute $H(S_1 || S_2 || W)$ for an arbitrary number of different W's *without* knowing S_1, just by using that fact that $H(S_1 \,||\, S_2 \,||\, W) = G(G(h \,||\, S_2) \,||\, W)$. Hence, if H is a Merkle-Damgard type of hash function, the assumption that the input of every fast query is known to the adversary would be unrealistic. The same applies to sponge-based hash functions and to any other hash functions that read their input blocks in an online way and compress them to a shorter state.

Security against distributed adversaries. By using MPC-hardness, we want to enforce that the distributed adversary must run the cryptographic task individually. More concretely, if the adversaries manage to complete the cryptographic task within some specified time bound (seconds in the case of PIK and hours in the case of ISS), then one of the adversaries, say $\mathcal{A}_j$, must know some secret information S completely. Hence, we need some formal method to model "knowledge".

The question of formalizing knowledge has a long history in cryptography, and in particular, it has been studied in the context of "proofs of knowledge" [3], "knowledge of exponent" [14], or "plaintext-awareness" [5]. None of these approaches considers attacks by a distributed adversary. The most relevant to ours is the model of [5], which considers an adversary with access to a hash function (modeled as a random oracle). It is assumed that if an adversary $\mathcal{A}$ evaluated H on some input x, then $\mathcal{A}$ knows the input x and the corresponding output $H(x)$. Technically: the (input, output) pairs are later given to an algorithm $\mathcal{E}$ called "knowledge extractor". If, based on these tuples, the knowledge extractor outputs some message S, then we assume that "$\mathcal{A}$ knows S" (since $\mathcal{A}$ could have computed S herself just by observing the oracle queries and the replies to them).

In our case, we use the concept of a knowledge extractor but slightly adjust it in the following way. First, we will consider knowledge extractors $\mathcal{E}_b$ for each of the different adversaries $\mathcal{A}_b$. Second, each such knowledge extractor takes as input the transcript of queries $\mathcal{T}_b^{\mathsf{fast}}$ that $\mathcal{A}_b$ has made to the oracle Ω_H only in *mode* = fast. Put differently: queries made by $\mathcal{A}_b$ in *mode* = slow (recall that these queries model MPC evaluations of H with a potentially unknown input) are not given to the knowledge extractor $\mathcal{E}_b$. Finally, we say that an adversary $\mathcal{A}_j$ individually knows a secret S if there exists an efficient knowledge extractor $\mathcal{E}_j$ such that $S \in \mathcal{E}_j(\mathcal{T}_b^{\mathsf{fast}})$.

Allowing pre-computation of the hash function. To model realistic attacks, we do not make any assumptions about how much time the distributed adversary has before the protocol starts. In particular, we allow the sub-adversaries to perform any distributed computation that involves the hash function H before the beginning of the protocol. This will be reflected by assuming that the attack works in two phases: in the first one, called *pre-processing* phase, the adversary will not be restricted in the number of slow hash queries that she can evaluate. Such a restriction will only apply in the second, *online*, phase. We protect our protocols against such pre-computation attacks by making all hashes depend on random values that are unknown before the online phase.

1.2 Informal description of PIK

As outlined in the introduction, a PIK protocol should allow the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that a secret S (that they both know) is stored on one machine and known to the machine owner. This is done by forcing $\mathcal{P}$ to quickly reply to challenges Z from $\mathcal{V}$ in a way that proves that $\mathcal{P}$ performed intensive computation on the entire value of S. Since $\mathcal{P}$ measures the response time, it will notice any response delays that are due to the fact that S is, in fact, distributed between different "sub-adversaries", and it is not stored on one machine. This distinction is made by performing a large amount of "hash computations" on S, which in practice, for example, can be carried out via a highly optimized ASIC.

As outlined in the previous subsection, two main parameters that characterize the adversary are σ – the number of hashes that can be computed in such a way that no sub-adversary learns their inputs since they are computed using MPC (in our model, this corresponds to the "slow" queries to the oracle), and ρ – the

number of communication rounds between the $\mathcal{A}_b$'s. In Sect. 1.1, we already explained the idea behind the slow queries. The bound on the number of communication rounds is relatively mild in practice. Recall that $\mathcal{A}_b$'s are connected over the Internet, and hence assuming that no more than 1000 rounds of communications per second (say) are executed between them is reasonable.

Our scheme is described formally in Sect. 4. The reader may, in particular, look at the diagram in Fig. 2 – we will refer to it while presenting our solution informally below. Let us start with describing a simple scheme for proving knowledge of a message consisting of one block (i.e., messages of a form $S = (S_1)$, where $S_1 \in \{0,1\}^\lambda$), and assuming that (a) the sub-adversaries cannot execute any slow queries, and (b) the sub-adversaries cannot communicate during protocol execution. In this case, the following idea works: the verifier sends a challenge Z to the prover, and the prover has to respond with $h = H(Z \,||\, S_1)$. Since no slow queries are allowed, and the sub-adversaries cannot communicate. Thus one sub-adversary, say $\mathcal{A}_b$, needs to know (Z, S_1). Let us now show how to remove our artificial assumptions in the above example.

Allowing slow queries. Recall that above, we assumed that the sub-adversaries could not perform any slow queries (i.e., no H can be computed using MPCs). We eliminate this restriction in the following way: namely, we force the prover to perform multiple computations of hashes on different nonces to find a nonce that leads to a hash starting with ζ zeros. This ensures that to convince the verifier, there must be an individual adversary $\mathcal{A}_b$ that makes a large number of fast queries that contain as input S_1. Notice that our approach is very similar to Bitcoin's puzzles, except that we do it κ times to reduce the variance in finding a solution. The nonce that is used in the ith puzzle is denoted with W^i.

Longer messages S. Let us now discuss how to eliminate the assumption that S just consists of one block. Let $S = (S_1, \ldots, S_n)$ where $n \geq 1$. Our idea is straightforward (see also the first column on Fig. 2, ignoring the values at the beginning of the hashed blocks): we apply the construction for a single block iteratively, i.e., for the ith nonce W we let $Q_1 := H(Z \,||\, W)$, and then for each $j = 2, \ldots, n+1$ we let $Q_j := H(S_j \,||\, Q_{j-1})$.

Allowing communication between the sub-adversaries. Note that the above construction is insecure if there are no restrictions on the communication between the $\mathcal{A}_b$'s. Indeed, imagine two sub-adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ and suppose S has two blocks $S = (S_1, S_2)$, with $\mathcal{A}_1$ holding S_1 and $\mathcal{A}_2$ holding S_2. Then these adversaries can break the above PIK as follows: $\mathcal{A}_1$ computes a massive number of hashes $Q_2 := H(S_1 \,||\, H(Z, W))$ (for different values of W) using fast queries to Ω^{fast} and sends the Q_2's to $\mathcal{A}_2$. Sub-adversary $\mathcal{A}_2$ processes every Q_2 by computing $Q_3 := H(S_2 \,||\, Q_2)$ in order to find Q_2 such that Q_3 starts with ζ zeros. Once such Q_2 is found, she communicates it to $\mathcal{A}_2$, who checks which W this Q_2 corresponds to and sends this Q to the verifier.

The above example shows that we need to restrict communication between the sub-adversaries. As discussed in Sect. 1.1 we choose to do it by putting a bound ρ on the number of rounds (an alternative approach would be to bound the communication size, but it seems more challenging to work with in practice). In our construction, we use this assumption by requiring that the prover needs to compute $d = \rho + 1$ iterations of the procedure outlined above (cf. Fig. 2).

Dealing with artificial fast queries. While from the above, it is clear that one of the sub-adversaries $\mathcal{A}_b$ has to know the entire S, it is not immediately clear how to build an efficient knowledge extractor. One challenge here is that the adversary may try to "confuse" us by making "useless" fast queries to Ω_H. We address this challenge by adding 2-bit flags to the inputs on which the hash function is evaluated. This enables the knowledge extractor to identify starting points for efficiently extracting the most likely values for S.

1.3 Informal description of ISS

Our individual secret sharing scheme is based on the following simple idea. Let $P_1, \ldots, P_a$ be the parties among which the secret S is shared, and let t be the threshold denoting the minimal number of parties needed to reconstruct the secret. In order to share S in an "individual" way, we first share a random value $X \leftarrow\!\!\$\, \{0,1\}^\kappa$ using a standard t-out-of-a secret sharing. We then compute a long chain of hashes H on

X. Let K be the output of this computation, and let η be the length of the chain. We then use K to encrypt our secret S using a standard symmetric encryption scheme. The shares of S are the shares of X plus the resulting ciphertext C. Reconstruction works in a natural way: t parties first reconstruct X (using the standard reconstruction procedure), and then one of them computes K. Afterwards, S is computed by decrypting C using key K.

The parameter η is chosen in such a way that it is feasible to compute K from X in plain (e.g. it takes 10 minutes in hardware), but computing it using MPC is infeasible within the time in which we want S to remain secret. In the security proof we use the fact that if a long chain of hashes was computed by the adversary, then a noticeable fraction of intermediate results of this computation had to be computed using the fast queries. For the details see Sect. 5.

1.4 Related work

Using cryptography for malicious purposes has been studied before, most notably in the context of "Cryptovirology" proposed by Young and Yung [31]. The approach of [31] focuses on the malicious use of public-key encryption and not the MPCs. Hence, our paper can be viewed as a natural continuation of the approach of [31] (with MPCs being more "advanced" primitives than the public-key encryption schemes). The idea of preventing leaking secrets has been studied extensively in a context such as traitor-tracing, e.g., it [13, 26]. To our knowledge, none of these works considers a distributed adversary.

On a higher level, our paper is also related to papers that look at defining the notion of "identity" in cryptography. In particular, it has some similarities to Position-Based Cryptography [11], where an identity of a user is defined by its geographic location. This approach is also based on measuring the prover's response time. Still, it is assumed that the communication is not done over the Internet but via physical signals (the whole approach is based on the fact that the speed of electromagnetic signals is fixed). Another difference is that the users in [11] do not have secret credentials but are identified by their geographic location. As already mentioned, our proof techniques are similar to those used in the context of space-bounded cryptography, in particular in the construction of schemes that are secure based on assumptions about the restricted memory of the adversary and whose security relies on hash functions, see, e.g., [1, 16, 17, 19]. For the differences between our model and the ones used in this area, see Sect. 1.1.

Concurrent work. In a very recent concurrent work, Kelkar et al. [25] introduce the concept of *complete knowledge*. A proof of complete knowledge (PoCK) guarantees that a single party has complete control/knowledge of its secret. This is very similar to our notion of PIK. We emphasize that while both works aim at similar goals, there are significant differences. Their construction of a PoCK directly achieves a zero-knowledge property. In contrast, our basic PIK construction does not have this property (we outline a generic transformation of any PIK into a zkPIK in Sect. 4.4). In addition, they also present an implementation. On the other hand, their work is very much application-oriented. It lacks a formal model that takes into account the many subtleties that we attempt to model with the concept of individual cryptography (e.g., possible communication, fast/slow queries, and taking into account how hash functions are constructed in practice). In addition, they do not have a construction of individual secret sharing, which shows that our modeling has applications beyond proof of knowledge.

Acknowledgments. We would like to thank the anonymous Crypto reviewers for their helpful comments, especially for pointing out to use the fact the need to model the pre-processing attacks.

2 Preliminaries

A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive integer c there exists an integer N such that for every $n > N$ we have $|f(n)| \leq n^{-c}$. A sequence of events has an *overwhelming probability* if the probability of their negations is negligible. We will use the following standard fact.

Lemma 1. *For every $p < 1$ we have that $(1-p)^{1/p} \leq e^{-1}$.*

A pair of algorithms $(\mathsf{Enc} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*, \mathsf{Dec} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*)$ is a *CPA-secure symmetric encryption scheme* (see, e.g., [24]) if for every $K, M \in \{0,1\}^*$ we have that $\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$. Moreover, we require that for every poly-time machine $\mathcal{D}_1^{\mathsf{cpa}}$, that takes as input 1^κ, and outputs $S^0, S^1, Y \in \{0,1\}^*$ (such that $|S^0| = |S^1|$) an every every poly-time machine $\mathcal{D}_2^{\mathsf{cpa}}$ it holds that $\Pr\left[\mathcal{D}_2(Y, \mathsf{Enc}(K, S^0)) = 1\right] - \Pr\left[\mathcal{D}_2(Y, \mathsf{Enc}(K, S^1)) = 1\right] \leq \mathsf{negl}(\kappa)$, where $K \leftarrow\!\!\$\, \{0,1\}^\kappa$. We will also use the following lemma, whose proof appears in Appx. A.1.

Lemma 2. *Let $\kappa, \zeta \in \mathbb{N}$ be arbitrary parameters (where ζ can be a function of κ). Let $U_1, \ldots, U_C$ be independent random variables distributed over $\{0,1\}$ and such that for each i we have that $\Pr[U_i = 1] = 2^{-\zeta}$. Let $U := U_1 + \cdots + U_C$. We have that (a) if $C = \kappa \cdot 2^{\zeta+1}$ then $\Pr[U \leq \kappa] \leq \mathsf{negl}(\kappa)$ and (b) if $C \leq \kappa \cdot 2^{\zeta-1}$ then $\Pr[U \geq \kappa] \leq \mathsf{negl}(\kappa)$.*

3 The model

This section provides more details on the model already informally introduced in Sect. 1.1. All protocols are executed in an asynchronous model. Every protocol is parameterized by a security parameter 1^κ and a hash function H that is modeled differently in the honest and adversarial executions. In the honest execution, the parties access H in a black-box way (via a standard random oracle [4]), except for Sect. 4.4, where a "circuit access" to H is needed (in the construction of zkPIK).

In the adversarial model, the malicious parties access H via an interactive machine Ω_H, which chooses a random function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ (where λ and κ are parameters of the model) and interacts with parties $\mathcal{A}_1, \ldots, \mathcal{A}_a$ by accepting queries of a form (x, mode), where $x \in \{0,1\}^*$ and $\mathit{mode} \in \{\mathsf{fast}, \mathsf{slow}\}$. If $\mathit{mode} = \mathsf{fast}$, then a query is called *fast*. It is called *slow* otherwise. Each query coming from a party $\mathcal{A}_b$ is answered to $\mathcal{A}_b$ with $H(x)$ (we also say that $\mathcal{A}_b$ *evaluated* H on input x). The execution of a protocol is divided into the *pre-computation* phase and the *online* phase, happening one after another. We say that Ω_H *is σ-bounded* if the total number of slow queries answered in the online phase is at most σ. The queries that exceed this quota are answered with $\perp$. The total number of fast queries is only bounded by the time complexity of the adversaries (i.e., it is polynomial in κ).

The main idea is that the fast queries are "cheap" and the participants will be allowed to send much more of them than the "expensive" slow queries (which correspond to queries computed using MPC/TEE techniques). On the other hand, when analyzing what the adversarial parties learned from the execution (i.e. when defining the "knowledge extractors", see below), only the fast queries will count. Namely, only the inputs to such queries will be considered known to the querying party. The distinction between the pre-computation phase and the online phase serves to model the fact that before the protocol starts, the sub-adversaries have an unbounded (but polynomial) time and can execute any distributed protocol. Note that the *mode* flag is only used for "accounting" purposes: the actions Ω_H do not depend on the value of this flag, except when defining the available budget of queries.

At the end of the execution of a protocol, we look at the information each party received as a result of the fast oracle queries. We define the *local hash transcript of a party $\mathcal{A}_b$* to be the sequence $\mathcal{T}_b$ of hash inputs that Ω_H received from $\mathcal{A}_b$ (in the same order in which they were received). Let $\mathcal{T}_b^{\mathsf{fast}}$ be the sub-sequence of $\mathcal{T}_b$ containing only the inputs corresponding to fast queries (call it *local fast-hash transcript of a party $\mathcal{A}_b$*). A *(knowledge) extractor* $\mathcal{E}$ is a deterministic poly-time machine that takes $\mathcal{T}_b^{\mathsf{fast}}$ as input and produces as output a finite set $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}}) \subset \{0,1\}^*$. The extractors have block-box access to H (via the same oracle Ω_H, only using the fast queries).

For future reference, also define the *global hash transcript* to be the sequence $\mathcal{T}$ of hash inputs that Ω_H received (in the same order in which they were received). Let the *global fast-hash transcript* be the sub-sequence $\mathcal{T}^{\mathsf{fast}}$ of $\mathcal{T}$ containing only the inputs corresponding to fast queries.

4 Proofs of Individual Knowledge

We now provide formal details of the definition and the construction that were informally presented in Sect. 1.

4.1 Definition

A *proof of individual knowledge (PIK)* is a protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ (also denoted $(\mathcal{P} \leftrightarrows \mathcal{V})$), which are probabilistic poly-time machines with access to a hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$. The prover and the verifier take as input a pair $(1^\kappa, S)$, where 1^κ is a security parameter and $S \in \{0,1\}^*$. For $\eta_\mathcal{P}, \eta_\mathcal{V} \in \mathbb{N}$, we say that the prover and the Verfier are $\eta_\mathcal{P}$- and $\eta_\mathcal{V}$-*bounded* (respectively) if the prover and the Verfier make at most $\eta_\mathcal{P}$ and $\eta_\mathcal{V}$ evaluations of H (respectively). We require that the protocol is *complete*, i.e., if both parties are honest (and their inputs are as above), then with overwhelming probability, the verifier outputs yes (in which case we also say that $\mathcal{V}$ *accepts*).

The second required property is *soundness*. Define a (ρ,σ)-*distributed adversary* to be a tuple $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of poly-time interactive *sub-adversaries*. The honest verifier $\mathcal{V}$ receives $(1^\kappa, S)$ as input and interacts with $\mathcal{A}_1$ that also receives $(1^\kappa, S)$ as input. Intuitively, $\mathcal{A}_1$ plays the role of the (malicious) prover from the point of view of the verifier $\mathcal{V}$. Initially, the sub-adversaries can run a per-computation phase, where they can send an arbitrary number of slow queries to the oracle Ω_H. Then, the protocol starts, and the sub-adversaries enter the online phase in which they can make at most σ queries to the oracle Ω_H. The adversaries run in at most ρ rounds, where each of them has the following form: (1) each sub-adversary $\mathcal{A}_b$ performs some local computation, at the end of which $\mathcal{A}_b$ outputs a string Str_b, and (2) each Str_b is delivered to every other sub-adversary $\mathcal{A}_{\widehat{b}}$. We emphasize that we do not restrict the size of the communicated messages.

Consider an execution of a distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ against an honest verifier (on input $(1^\kappa, S)$). Define $\mathsf{exec}((\mathcal{A}_1, \ldots, \mathcal{A}_a) \leftrightarrows \mathcal{V}); 1^\kappa; S)$ to be equal to $(\mathcal{T}^{\mathsf{fast}}_1, \ldots, \mathcal{T}^{\mathsf{fast}}_a, Out_\mathcal{V})$, where each $\mathcal{T}^{\mathsf{fast}}_b$ is the local fast-hash transcript of $\mathcal{A}_b$ and $Out_\mathcal{V} \in \{\mathsf{yes}, \mathsf{no}\}$ is the output of $\mathcal{V}$. We now have the following.

Definition 1. *We say that $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ is a* PIK protocol sound against (ρ,σ)-distributed adversary *if there exists knowledge extractors $\mathcal{E}_1, \ldots, \mathcal{E}_a$ such that for every (ρ,σ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ and every $S \in \{0,1\}^*$ we have that*

$$\Pr[Out_\mathcal{V} = \mathsf{yes} \text{ and } S \notin \mathcal{E}_1(\mathcal{T}^{\mathsf{fast}}_1) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}^{\mathsf{fast}}_a)] \leq \mathsf{negl}(\kappa), \tag{1}$$

where $(\mathcal{T}^{\mathsf{fast}}_1, \ldots, \mathcal{T}^{\mathsf{fast}}_a, Out_\mathcal{V}) \leftarrow\!\!\$\ \mathsf{exec}((\mathcal{A}_1, \ldots, \mathcal{A}_a) \leftrightarrows \mathcal{V}); 1^\kappa; S)$. We say that $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ has extraction efficiency $(\alpha_\mathsf{O}, \alpha_\mathsf{T}, \alpha_\mathsf{S})$ *if $|\mathcal{E}_1(\mathcal{T}^{\mathsf{fast}}_1) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}^{\mathsf{fast}}_a)| \leq \alpha_\mathsf{O}$, and $\mathcal{E}$ operates in time at most α_T and uses space at most α_S. The parameters $\alpha_\mathsf{O}, \alpha_\mathsf{T}$ and α_S can be functions of some other parameters in the system.*

The idea behind α_O is that $\alpha_\mathsf{O} - 1$ is the number of "false positives", i.e., values in $\mathcal{E}_1(\mathcal{T}^{\mathsf{fast}}_1) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}^{\mathsf{fast}}_a)$ that are not equal to S. Obviously, the smaller α_O is, the better. In Sect. 4.4, we describe a method that, with high probability, allows to reduce the number of such false positives to 0 in many applications.

4.2 Construction

In this section, we present our construction of a PIK protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$, which was already informally described in Sect. 1.2. The protocol is parameterized by a "moderate hardness parameter" $\zeta \in \mathbb{N}$ and a security parameter 1^κ. It uses a hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ with $\lambda \geq 2\kappa$. In practice, H could be a compression function of a popular hash function. The protocol participant takes as input $S \in \{0,1\}^*$ and 1^κ. We assume that $|S|$ is a multiple of $\lambda - \kappa - 2$ (if it is not the case, then one can pad S with zeros). Let $S_1, \ldots, S_n$ be such that $S = (S_1 \,||\, \cdots \,||\, S_n)$ where $|S_1| = \cdots = |S_n| = \lambda - \kappa - 2$.

Our PIK protocol is depicted in Fig. 1. It uses a sub-routine $\mathsf{scratch}$ (see also Fig. 2 for a graphical representation of the computation of $\mathsf{scratch}$). The protocol starts with the verifier sending a random challenge $Z \in \{0,1\}^\kappa$. Then, the goal of the prover is to find κ nonces $W^1, \ldots, W^\kappa \in \{0,1\}^{\lambda-\kappa-2}$ that convince the verifier that the prover did a substantial amount of work for the challenge Z. This is done by requiring that the output of every $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. We have the following definition.

Definition 2. *We call a given* $\mathsf{scratch}(S, Z, W)$ successful *if its output starts with ζ zeros (cf. Step 2b on Fig. 1 (b)).*

This is similar to the Bitcoin mining procedure, except that we require κ nonces to be found (in Bitcoin $\kappa = 1$) in order to reduce the variance of the success probability. The main idea behind $\mathsf{scratch}$ is that it forces the prover to sequentially compute d times H on every block S_j of S. We also force the prover to compute $\mathsf{scratch}(S, Z, W)$ on a large number of W's. The search for the W^i's can be fully parallelized. The only non-parallelizable part is the $\mathsf{scratch}$ procedure, which takes as input (S, Z, W), where S is the message, Z is the challenge, and W is the nonce.

The inputs of H in $\mathsf{scratch}$ start with two bits: 00 indicates that the hash is computed on (Z, W) (where Z is a challenge and W is a nonce), 01 indicates that the first block (S_1) is hashed (together with some Q), and 10 indicates that the hashed value S_j is one of the subsequent blocks (i.e., $j > 1$). Labels 00, 01, and 10 are used to help the extractor find S (e.g. if the extractor sees that some $(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ was hashed then the extractor guesses that $\widehat{S}_1$ is the first block of S and starts searching for a hash of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$ with $\widehat{Q}_2 = H(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ (see Proof of Thm. 1 for the details). Note that in total $\mathsf{scratch}$ computes $nd+1$ hashes H. Security of $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ is stated in the following theorem.

Theorem 1. *Let κ, n, d be as above and let $\eta_{\mathcal{P}} := (nd+1)\cdot 2^{\zeta+1}\cdot\kappa$ and $\eta_{\mathcal{V}} := (nd+1)\cdot\kappa$. Assume $\sigma \leq \kappa\cdot 2^{\zeta-3}$ and $\rho \leq d-1$. Then $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ from Fig. 1 is a PIK protocol with $\eta_{\mathcal{P}}$-bounded prover and $\eta_{\mathcal{V}}$-bounded verifier that is sound against a (ρ,σ)-distributed adversary with extraction efficiency $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, where*

$$\mathbb{E}[\alpha_{\mathsf{O}}] = 2^{-\zeta}\cdot\ell,\ \mathbb{E}[\alpha_{\mathsf{T}}] = O(\ell_b),\ \textit{and}\ \mathbb{E}[\alpha_{\mathsf{S}}] = O(2^{-\zeta}\cdot\ell_b\cdot|S|).$$

Above, ℓ_b is the number of hashes computed by a sub-adversary $\mathcal{A}_b$, and $\ell := \ell_1 + \cdots + \ell_a$. The unit of time is a computation of H.

The proof of Thm. 1 is presented in Sect. 4.3. Before proceeding to it, let us comment on the parameters in the lemma statement. When it comes to the parameters of the honest prover and verifier, the most important ones are those denoting the "budget" for the hash computations, i.e., $\eta_{\mathcal{P}} = (nd+1)\cdot 2^{\zeta+1}\cdot\kappa$ and $\eta_{\mathcal{V}} = (nd+1)\cdot\kappa$ respectively. Note that each computation of the $\mathsf{scratch}$ procedure requires $(nd+1)$ hash computations. The verifier needs to do such a computation κ times; hence, she needs to perform only $(nd+1)\cdot\kappa$ hashes. For the prover, observe that each scratch attempt succeeds with probability $2^{-\zeta}$ (by "succeeding" we mean finding a value that starts with ζ zeros). Since the prover needs to be successful κ times, she needs, on average, $(nd+1)\cdot 2^{\zeta}\cdot\kappa$ scratch attempts. We set $\eta_{\mathcal{P}}$ to be the double of this parameter in order to make the probability that he is successful (see Def. 2) less than κ times exponentially small.

Observe also that if we substitute $2^{-\zeta}$ with $2\cdot\kappa\cdot(nd+1)/\eta_{\mathcal{P}}$ then the formula $2^{-\zeta}\cdot\ell$ (appearing both in the bounds on $\mathbb{E}[\alpha_{\mathsf{O}}]$ and $\mathbb{E}[\alpha_{\mathsf{S}}]$) can be rewritten as: $2\cdot\kappa\cdot(nd+1)\cdot\ell/\eta_{\mathcal{P}}$. It is interesting to look at the last factor, i.e., $\ell/\eta_{\mathcal{P}}$. Recall that ℓ is the number of hashes computed by the adversary $\mathcal{A}$. On the other hand, $\eta_{\mathcal{P}}$ is the maximal number of hashes computed by the honest prover. Hence, $\ell/\eta_{\mathcal{P}}$ corresponds to the multiplicative advantage of the adversary with respect to the honest prover, or, in other words, it answers the question "How much more computing power does the adversary have compared to the honest prover?". In our theorem, the bounds on α_{O} and α_{S} grow linearly with $\ell/\eta_{\mathcal{P}}$. Intuitively, this comes from the fact that a powerful adversary can always "mislead" the extractor by executing a large number of $\mathsf{scratch}$ procedures on $\widehat{S} \neq S$.

4.3 Proof of Thm. 1

Completeness. Let us start with proving completeness. We first upper-bound the probability that the protocol halts in Step 2 due to the fact that the budget for H computations was exhausted. Note that this budget allows the prover to evaluate $\mathsf{scratch}$ $\lfloor\eta_{\mathcal{P}}/(nd+1)\rfloor = 2^{\zeta+1}\cdot\kappa$ times (since each $\mathsf{scratch}$ execution takes $nd+1$ hash evaluations). For $i = 1, \ldots, 2^{\zeta+1}\cdot\kappa$, let $U_i \in \{0,1\}$ be equal to 1 if and only if the ith $\mathsf{scratch}$ execution was successful, i.e., it produced an output that starts with κ zeros. Set $U = U_1 + \cdots + U_{2^{\zeta+1}\cdot\kappa}$. For simplicity of the analysis assume that $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ does not stop once all the W^j's are found. Clearly, the U_i's are independent. Moreover, we have that each $\Pr[U_i = 1] = 2^{-\zeta}$ (the probability that a random string starts with ζ zeros) and hence each $\mathbb{E}[U_i] = 2^{-\zeta}$. Therefore by Lemma 2 (Point (a)), we get that $\Pr[U < \kappa] \leq \mathsf{negl}(\kappa)$. Therefore with overwhelming probability the prover finds κ values W such that $\mathsf{scratch}(S, Z, W)$ starts with

Procedure $\mathsf{scratch}(S, Z, W)$

1. Assume $S = (S_1 \,||\, \cdots \,||\, S_n)$, where each $|S_j| = \lambda - \kappa - 2$.
2. For $k = 1$ to d do:
   For $j = 1$ to n do:
$$Q_j^k := \begin{cases} H(00 \,||\, Z \,||\, W) & \text{if } k = 1 \text{ and } j = 1 \\ H(10 \,||\, S_n \,||\, Q_n^{k-1}) & \text{if } k \neq 1 \text{ and } j = 1 \\ H(01 \,||\, S_1 \,||\, Q_1^k) & \text{if } j = 2 \\ H(10 \,||\, S_{j-1} \,||\, Q_{j-1}^k) & \text{if } j > 2 \end{cases}$$
3. Output $H(10 \,||\, S_n \,||\, Q_n^d)$.

(a)

Protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$

The protocol is parameterized with a "moderate hardness" parameter $\zeta \leq \kappa$. It is executed between the $\eta_{\mathcal{P}}$-bounded prover $\mathcal{P}$ and the $\eta_{\mathcal{V}}$-bounded verifier $\mathcal{V}$. Both parties take as input $(1^\kappa, S)$, where $S = (S_1, \ldots, S_n)$.

1. The verifier $\mathcal{V}$ chooses a random *challenge* $Z \in \{0,1\}^\kappa$ and sends it to the prover $\mathcal{P}$.
2. The prover $\mathcal{P}$ does the following **parallel search** across different values of $i = \{1, \ldots, \kappa\}$ and *nonces* $W^i \in \{0,1\}^{\lambda-\kappa-2}$:
   (a) Let $Q^i := \mathsf{scratch}(S, Z, W^i)$.
   (b) If Q^i starts with ζ zeros then record W^i and **stop the parallel search**.
   The above search is done as long as the prover did not exhaust her budget for computing hashes (recall that she can make at most $\eta_{\mathcal{P}}$ of them). If this happens before the search is over, then $\mathcal{P}$ outputs $\perp$ to $\mathcal{V}$, who also outputs $\perp$, and then both halt. Otherwise, we proceed to the next step.
3. The prover sends $(W^1, \ldots, W^\kappa)$ to the verifier.
4. Upon receiving $(W^1, \ldots, W^\kappa)$ the verifier outputs yes if for *all* $i \in \{1, \ldots, \kappa\}$ it holds that the output of $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. Otherwise, the verifier outputs no.

(b)

Fig. 1. Proof of individual knowledge (PIK) $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ that satisfies Def. 1 (see Thm. 1). The main procedure is depicted in point (b). It uses a sub-routine $\mathsf{scratch}$ depicted in point (a).

ζ zeros without exhausting her hash budget, and hence she does not output $\perp$. It is also easy to see that the Verifer always accepts in such a case (since she just repeats the same $\mathsf{scratch}$ computation as the prover for the W^i values that she received in Step 4, and $\eta_{\mathcal{V}}$ hash evaluations are needed for this computation).

Soundness. To show soundness, let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ be a (ρ, σ)-distributed adversary. Consider an execution of $\mathcal{A}$ against an honest verifier on input $(1^\kappa, S)$ and let Z be the challenge that the verifier sends in Step 1. Recall that the adversary has access to a σ-bounded oracle Ω_H that she can use to evaluate hash function H. For an arbitrary $\widehat{S}$ (necessarily equal to S), such that $\widehat{S} = (\widehat{S}_1 \,||\, \cdots \,||\, \widehat{S}_n)$ (with each $\widehat{S}_j \in \{0,1\}^{\lambda-\kappa-2}$) and an arbitrary $\widehat{Q}_1 \in \{0,1\}^\kappa$ a *trace on* $\mathcal{T}$ *for* $\widehat{S}$ *(starting with* $\widehat{Q}_1$*)* is a sequence $(i_1, \ldots, i_n)$ such that

- $\mathcal{T}[i_1] = (01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ and
- for every $m = 2, \ldots t$ we have: $\mathcal{T}[i_1] = (10 \,||\, \widehat{S}_m \,||\, H(\mathcal{T}[i_{m-1}]))$.

Intuitively a trace is a sequence of indices on $\mathcal{T}$ that "look like" an attempt to evaluate a column on Fig. 2 for *some* $\widehat{S}$. For any nonce W, an *S-scratch on* $\mathcal{T}$ *for* W is a sequence

$$i_0 \,||\, (i_1^1, \ldots, i_n^1) \,||\, \cdots \,||\, (i_1^d, \ldots, i_n^d), \tag{2}$$

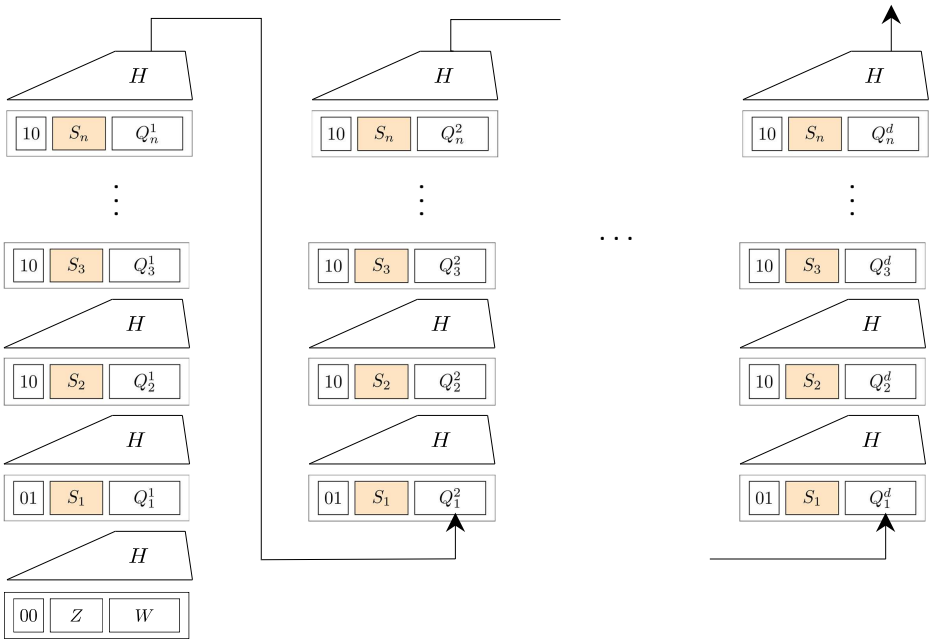

Fig. 2. A diagram representing an execution of $\mathsf{scratch}((S_1, \ldots, S_n), Z, W)$ procedure from Fig. 1.

where $\mathcal{T}[i_0] = (00 \,||\, Z \,||\, W)$ and each $(i_1^k, \ldots, i_n^k)$ is a trace for S staring either with $H(\mathcal{T}[i_0])$ (if $k = 1$) or with $H(10 \,||\, S_n \,||\, H(\mathcal{T}[i_n^{k-1}]))$ (otherwise). In other words: S-scratch is a sequence of indices on $\mathcal{T}$ that correspond to an attempt to evaluate the entire diagram on Fig. 2 for $\widehat{S} = S$.

Claim 1. For a hash transcript $\mathcal{T}$ consider two distinct nonces W and $\widehat{W}$. Let I and $\widehat{I}$ be S-scratches on $\mathcal{T}$ for W and $\widehat{W}$, respectively. Then with overwhelming probability, we have that I and $\widehat{I}$ are disjoint, i.e., there does not exist an index i that appears in both I and $\widehat{I}$.

Proof (sketch). Let Q_i^k's and $\widehat{Q}_i^k$'s be the values of the variables in the $\mathsf{scratch}$ procedure when run on input (S, Z, W) and $(S, Z, \widehat{W})$, respectively (see Fig. 1 or the diagram on Fig. 2). Suppose there exists an index i that appears in both I and $\widehat{I}$. Obviously $\mathcal{T}[i]$ cannot be of a form $(00 \,||\, Z \,||\, W)$, since $W \neq \widehat{W}$. Hence $\mathcal{T}[i]$ must have a form $(b_0 \,||\, b_1 \,||\, S_j \,||\, Q)$. This means that $\mathcal{T}[i]$ is an output of a sequence of hashes starting with $H(00 \,||\, Z \,||\, W)$ *and* simultaneously, it is an output of a sequence of hashes starting with $H(00 \,||\, Z \,||\, \widehat{W})$. Using this information, one can efficiently find a collision in H. Since the probability of finding collisions in H is negligible, the probability that such W and $\widehat{W}$ can be found has to be negligible. □

We will need the following definition.

Definition 3. *Let W be a nonce such that there exists an S-scratch on $\mathcal{T}$ for W. We call W* fast *if for every S-scratch $(i_1, \ldots, i_m)$ on $\mathcal{T}$ for W each $\mathcal{T}[i_j]$ is a fast query. Otherwise, we call it* slow.

We will also use the following simple fact that states that with overwhelming probability, the adversary has to compute the Q_j^k variables in the same order as the honest party executing the $\mathsf{scratch}$ procedure.

Claim 2. Fix some $Z \in \{0,1\}^\kappa$ and $W \in \{0,1\}^{\lambda-\kappa-2}$ and let S be as above. Let Q_j^k's be the variables computed in the $\mathsf{scratch}(S, Z, Q)$ procedure (see Fig. 1 (a)). For a global hash transcript $\mathcal{T}$

– let i_0^1 be the first position in $\mathcal{T}$ on which $(00 \,||\, Z \,||\, W)$ appears, and
– for $k \in \{1, \ldots, d\}$ and $j \in \{1, \ldots, n\}$ let i_j^k be the first position in $\mathcal{T}$ such that $\mathcal{T}[i_j^k]$ ends with Q_j^k.

Then with overwhelming probability the i_j^k's when sorted lexicographically by (k, j) are monotonically increasing, i.e.,

$$\forall_{\substack{(k_0,j_0)\\(k_1,j_1)}} \; k_0 < k_1 \vee ((k_0 = k_1) \wedge (j_0 < j_1)) \text{ implies that } i_{j_0}^{k_0} < i_{j_1}^{k_1}. \tag{3}$$

Proof (sketch). Let $\prec$ be the strict lexicographic order on (k, j)'s used in Eq. (3). Suppose that there exists $(k_0, j_0) \prec (k_1, j_1)$ such that $i_{j_0}^{k_0} \geq i_{j_1}^{k_1}$. Clearly, unless a collision in H is found (which happens with negligible probability), all the Q_j^k's are distinct, and hence we can assume that $i_{j_0}^{k_0} > i_{j_1}^{k_1}$. Without loss of generality assume that (k_0, j_0) immediately precedes (k_1, j_1) in the "$\prec$" order. Observe that $Q_{j_1}^{k_1}$ is an output of a hash function H when evaluated on an input that contains $Q_{j_0}^{k_0}$ (with a convention that $Q_0^1 := 00 \,||\, Z \,||\, W$). But $i_{j_0}^{k_0} > i_{j_1}^{k_1}$ means that $Q_{j_1}^{k_1}$ was submitted to the oracle *before* the adversary learned $Q_{j_0}^{k_0}$. Since the outputs of a random oracle are uniform over $\{0,1\}^\kappa$, this happens with negligible probability. This finishes the proof of the claim. □

Claim 3. Suppose the number of fast W's is at most $\kappa \cdot 2^{\zeta-3}$. Then the probability that $\mathcal{V}(1^\kappa, S)$ accepts on input S is negligible.

Proof (sketch). Recall that we assumed that $\sigma \leq \kappa \cdot 2^{\zeta-3}$ and in Claim 1 we have shown that for distinct W and $\widehat{W}$ the S-scratches for W and $\widehat{W}$ are disjoint. Since $Z \in \{0,1\}^\kappa$ is sampled by the prover in the online phase, thus with overwhelming probability $(00 \,||\, Z \,||\, W)$ is sent to the oracle in the online phase. By Claim 2, this means that with overwhelming probability, all of the S-scratches on T for W contain only queries sent to the oracle in the online phase. Thus, the total number of slow W's is at most $\kappa \cdot 2^{\zeta-3}$. Adding the fast ones, we get that the total number C of W's for which an S-scratch in $\mathcal{T}$ exists is at most $\kappa \cdot 2^{\zeta-2}$. Let $W^1, \ldots, W^C$ be these nonces.

By Claim 2, with overwhelming probability, the adversary *first* needs to compute the entire S-scratch before learning whether it was successful or not. For each $i \in \{1, \ldots, C\}$ let $U_i \in \{0,1\}$ be equal to 1 if and only if $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. Clearly, the U_i's are independent and $\Pr[U_i = 1] = 2^{-\zeta}$. Let $U = U_1 + \cdots + U_C$. Using Lemma 2, we get that $\Pr[U \geq \kappa/2] \leq \mathsf{negl}(\kappa)$. Recall that $\mathcal{V}$ accepts only if she receives $(W^1, \ldots, W^\kappa)$ such that each $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. From the analysis above, we get that with overwhelming probability, there exists $\kappa - \kappa/2 = \kappa/2$ values W_i such that the corresponding value $(10 \,||\, S_n \,||\, Q_n^d)$ is not in $\mathcal{T}$, or, in other words, H was not evaluated on it. Clearly, the probability that $H(10 \,||\, S_n \,||\, Q_n^d)$ starts with ζ zeros for all such W_i's is equal to $(2^{-\zeta})^{\kappa/2} = 2^{-\zeta \cdot \kappa/2} \leq \mathsf{negl}(\kappa)$. □

Before presenting our extractor $\mathcal{E}$, consider the following natural construction idea. By Claim 3, if the adversary was successful, then $\mathcal{T}^{\mathsf{fast}}$ needs to contain at least $\kappa \cdot 2^{\zeta-3}$ S-scratches of a computation of $\mathsf{scratch}(S, Z, W)$ (for some W's). Each such an S-scratch contains $d = \rho + 1$ traces for S, where ρ is the maximal number of communication rounds. Therefore every S-scratch contains a trace computed in a single round. As we show formally later (see Claim 4), all the queries corresponding to such a trace had to come from a single adversary. Hence, it should be possible to find it by scanning all $\mathcal{T}_b^{\mathsf{fast}}$'s.

More concretely, consider an extractor that scans the transcript $\mathcal{T}_b^{\mathsf{fast}}$ in order to find elements of a form $(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ (for some $\widehat{S}_1$ and $\widehat{Q}_1$). Each such element is potentially part of an attempt by $\mathcal{A}_b$ to compute a scratch path on a message starting with $\widehat{S}_1$. The extractor stores $\widehat{S}_1$ and $\widehat{Q}_2 := H(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$. It then continues scanning the transcript to find elements of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$. Suppose $\mathcal{E}$ found such an element, and let $Q_3 := H(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$. It then looks for elements of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_3)$, and so on, until it finds n strings: $(\widehat{S}^1, \ldots, \widehat{S}^n)$. It then outputs $(\widehat{S}^1 \,||\, \cdots \,||\, \widehat{S}^n)$.

The above approach essentially works, although it needs some modifications. The main challenge is that the adversary can make "fake" calls to Ω_H, whose goal would be just to mislead the extractor. Hence, a hash transcript can contain $\widehat{S}$-scratches for some other $\widehat{S}$'s (possibly sharing a prefix with S). We deal with

these problems by exploiting the fact that in a successful execution, the adversary still needs to perform a large number ($\kappa \cdot 2^{\zeta-3}$) of $\mathsf{scratch}$ executions on real S. Our solution is presented in Fig. 3. The $\mathcal{E}$ procedure maintains a set $\mathcal{F}$ containing information that can lead to finding the solution S. The values are added to this set only with a certain probability ($2^{-\zeta}$). This probability is chosen so that the most likely *some* S-scratch is included in $\mathcal{F}$, yet, it reduces the size of $\mathcal{F}$ significantly, leading to better parameters. In particular, it prevents the size of $\mathcal{F}$ from becoming too large if the adversary makes lots of "fake" hash queries described above.

Knowledge extractor $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$

1. Let $\ell_b := |\mathcal{T}_b^{\mathsf{fast}}|$.
2. Let $\mathcal{F}$ be a set that is initially empty. Set $\mathcal{F}$ contains pairs of a form $(\widehat{Q}, \widehat{S})$, where $\widehat{Q} \in \{0,1\}^\kappa$ is a hash output, and $\widehat{S} \in \{0,1\}^*$.
3. For each $i = 1, \ldots, \ell_b$ do
   (a) If $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(01||\widehat{S}||\widehat{Q})$ (for some $\widehat{S}$ and $\widehat{Q}$) then with probability $2^{-\zeta}$ add a pair $(H(01||\widehat{S}||\widehat{Q}), \widehat{S})$ to $\mathcal{F}$ (if it is not stored there yet).
   Note that $(01 \,||\, \widehat{S} \,||\, \widehat{Q})$ can appear more than once in $\mathcal{T}_b^{\mathsf{fast}}$. Since we want the probability of being added to $\mathcal{F}$ to be the same for every string $(01 \,||\, \widehat{S} \,||\, \widehat{Q})$ (no matter how often it appears in $\mathcal{T}_b^{\mathsf{fast}}$), we perform this random choice by evaluating some function $\varphi : \{0,1\}^* \to \{0,1\}$ (treated as a random oracle) such that $\Pr[\varphi(x) = 1] = 2^{-\zeta}$. We add $(H(01 \,||\, \widehat{S} \,||\, \widehat{Q}), \widehat{S})$ to $\mathcal{F}$ if and only it $\varphi(01 \,||\, \widehat{S} \,||\, \widehat{Q}) = 1$.
   (b) If $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(10 \,||\, \widehat{S} \,||\, \widehat{Q})$ (for some $\widehat{S}$ and $\widehat{Q}$) and that there exists a pair $(\widehat{Q}, \widehat{S}') \in \mathcal{F}$ (for some $\widehat{S}' \in \{0,1\}^*$) then add $(H(10 \,||\, \widehat{S} \,||\, \widehat{Q}), (\widehat{S}' \,||\, \widehat{S}))$ to $\mathcal{F}$.
   Additionally, if $|(\widehat{S}' \,||\, \widehat{S})| = n$ then **output** $(\widehat{S}' \,||\, \widehat{S})$ (but do not terminate),

Fig. 3. The knowledge extractor for our PIK protocol. It preoceeds in an online fashion, reading $\mathcal{T}_b^{\mathsf{fast}}$ and outputting during the execution the "candidate" values for S. The output of $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$ is the set of all values that were output during the execution of $\mathcal{E}$.

Claim 4. Consider an execution of $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ against an honest verifier $\mathcal{V}$ on input $(1^\kappa, S)$. For each b, let $\mathcal{T}_b^{\mathsf{fast}}$ be the fast-hash transcript of $\mathcal{A}_b$. Suppose $\mathcal{V}(1^\kappa, S)$ accepts. Then with overwhelming probability for some b, the string S is among the values output by the knowledge extractor $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$.

Proof. Consider a global hash transcript $\mathcal{T}^{\mathsf{fast}}$. Let W be a fast W (see Def. 3). Recall that, since the adversary can evaluate H on the same value multiple times, there can be multiple S-scratches on $\mathcal{T}^{\mathsf{fast}}$ for W. Let $\mathsf{first\text{-}scratch}(W, \mathcal{T})$ denote the S-scratch composed of the *first* positions on $\mathcal{T}^{\mathsf{fast}}$ for W where the given value appears on $\mathcal{T}^{\mathsf{fast}}$. More formally, let $\mathsf{first\text{-}scratch}(W)$ be the S-scratch $(i_1, \ldots, i_m)$ for W such that for every i_j is the least index i such that $\mathcal{T}^{\mathsf{fast}}[i_j] = \mathcal{T}^{\mathsf{fast}}[i]$, i.e., we have:

$$\forall_{j \in \{1,\ldots,m\}} \; \forall_{i < i_j} \; \mathcal{T}^{\mathsf{fast}}[i] \neq \mathcal{T}^{\mathsf{fast}}[i_j]. \tag{4}$$

Recall that ρ is the number of rounds, and we assumed that $\rho \leq d - 1$. Therefore for every fast W and $\mathsf{first\text{-}scratch}(W)$ of form as in Eq. (2), there needs to exist a round and an index k such that hashes of values $\mathcal{T}[i_1^k], \ldots, \mathcal{T}[i_n^k]$ were all sent to Ω_H in a single round. Observe that the knowledge of each $\mathcal{T}[i_j^k]$ is needed to compute the next value $\mathcal{T}[i_{j+1}^k]$ and, because of Eq. (4) $H(\mathcal{T}[i_j^k])$ was not computed in the previous round. Since the output of a hash can be guessed with probability $2^{-\kappa}$, thus with overwhelming probability, all the values $\mathcal{T}[i_1^k], \ldots, \mathcal{T}[i_n^k]$ had to be submitted to Ω_H by single sub-adversary $\mathcal{A}_b$. Hence there must exist a trace $i_1^k, \ldots, i_n^k$ in $\mathsf{first\text{-}scratch}(W)$ that was computed by a single sub-adversary $\mathcal{A}_b$.

By Claim 3, with overwhelming probability, the total number of fast W's is greater than $\kappa \cdot 2^{\zeta-3}$. Hence, at least $\kappa \cdot 2^{\zeta-3}$ traces for S were computed by a single $\mathcal{A}_b$. The probability that each trace is added to $\mathcal{F}$ is

$2^{-\zeta}$. Hence, the probability that at least one trace for S is output by *some* $\mathcal{A}_b$ is at least $1-(1-2^{-\zeta})^{\kappa\cdot 2^{\zeta-3}}$, which, by Lemma 1, is at least $1-e^{-\kappa/8} \geq 1-\mathsf{negl}(\kappa)$. □

It remains to analyze the extraction efficiency of the extractor. Consider a run on $\mathcal{E}$ on input $\mathcal{T}_b$. We first show the following bound on the expected output size of $\mathcal{F}$:

$$\mathbb{E}[|\mathcal{F}|] \leq 2^{-\zeta} \cdot \ell_b \tag{5}$$

(where ℓ_b is the number of fast hash queries of $\mathcal{A}_b$). Recall that new values are added to $\mathcal{F}$ by scanning all ℓ_b positions on $\mathcal{T}_b^{\mathsf{fast}}$ in the following way:

1. if $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(01 \,||\, \widehat{S} \,||\, \widehat{Q})$ then a new value is added to $\mathcal{F}$ with probability at most $2^{-\zeta}$ and
2. if $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(10 \,||\, \widehat{S} \,||\, \widehat{Q})$ then a new value is added if $\widehat{Q}$ is a result of a chain of hashes that started with hashing some $(01 \,||\, \widehat{S} \,||\, \widehat{Q}')$ and $\varphi(01 \,||\, \widehat{S} \,||\, \widehat{Q}') = 1$. Since this happens with probability 2^{ζ} we get that the probability that $(10 \,||\, \widehat{S} \,||\, \widehat{Q})$ is added to $\mathcal{F}$ is $2^{-\zeta}$.

This proves (5). Since the values that are output by the extractor are a subset of $\mathcal{F}$, and $\ell = \ell_1 + \cdots + \ell_b$, thus we get that $\mathbb{E}[\alpha_{\mathsf{O}}] = \mathbb{E}\big[\mathcal{E}_1(\mathcal{T}_b^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_b^{\mathsf{fast}})\big] \leq 2^{-\zeta} \cdot \ell$.

To analyze the time and space complexity of $\mathcal{E}$ observe that $\mathcal{E}$ is an online algorithm that reads $\mathcal{T}_b^{\mathsf{fast}}$ once. Assume that $\mathcal{F}$ is maintained using Cuckoo Hashing [28], where in every pair $(\widehat{Q}, \widehat{S})$, the element $\widehat{Q}$ is treated as the key, and $\widehat{S}$ is the stored value. Thanks to this, looking up and inserting the values in $\mathcal{F}$ takes constant time. Thus, the expected time complexity α_{T} of each $\mathcal{E}$ is $O(\ell_b)$, and the expected space complexity α_{S} is $O(2^{-\zeta} \cdot \ell_b \cdot |S|)$. This concludes the proof. □

4.4 Extensions

This section describes possible extensions of the basic PIK protocol from Sect. 4. We leave the formalization of these extensions for future work as formally modeling attacks in the network settings is often highly non-trivial. For example, it requires us to consider different attack scenarios (e.g., man-in-the-middle, replay attacks, etc.) and typically involves a detailed description of the attack model (including modeling the network and the environment).

PIK as building block. As mentioned in Sect. 1, PIK is a primitive that will usually not be used in a "standalone" way but rather as a part of a more extensive system. The main reason for this is that the messages sent by the prover may provide some information about S. Indeed, the definition of PIK does not mention any privacy guarantees for the prover, neither against an external attacker nor against the verifier (who actually, in our setting from Def. 1, knows S entirely). In this section, we describe two extensions of PIK that address this problem. The first one ("Encrypted and Authenticated PIK") answers the issue of security against an external adversary, and the second one provides a version of PIK ("zero-knowledge PIK") that works against a verifier that does *not* know S. This version of PIK does not reveal any information about S to the verifier (or any other party).

Encrypted and Authenticated PIK (eaPIK). *Encrypted and Authenticated PIK (eaPIK)* is a version of PIK where the communication between the prover and the verifier is secured using a symmetric key K. Let us describe this solution as a general transformation of any PIK protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$. The main idea is quite straightforward: we let every message[6] in the protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ be encrypted and authenticated with a fresh key K that is shared between the prover and the verifier, sampled independently from S. Note that since we have a setup phase (for establishing S), thus assuming that in this phase, we also generate K is very mild. To summarize: the client and the server share the following:

[6] It is easy to see that in case of our protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ (see Fig. 1) the only message that may contain sensitive information about S is $(W^1, \ldots, W^\kappa)$ sent by the prover to the verifier in Step 3. Hence, it is enough if only this message is encrypted.

– key K for encrypting messages during the execution of the PIK protocol and
– secret S that is used as the input of both parties in PIK.

Of course, as long as the prover is honest (and hence: she stores S on a single machine), she easily convinces the verifier that she knows S. On the other hand, by the PIK properties, a dishonest prover cannot distribute S among different sub-adversaries. It is important to note that the PIK protocol only guarantees that S cannot be distributed, and K is not protected similarly. Hence, any potential application accessing the server should require the knowledge of S only, and it should not be assumed that K is stored on a single machine.

ZK-Proofs of Individual Knowledge. Let us now describe a solution for a setting where S is known only to the prover. In this case, PIK makes sense only if some publicly-available information on S is known. We model this in the following standard way. Suppose L is an NP-language characterized by some NP-relation $\varphi : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. The verifier holds some public information $pub \in \{0,1\}^*$, while the prover has an NP-witness $S \in \{0,1\}^*$ such that $(S, pub) \in \mathsf{L}$. A natural example of L is the language of all secret keys in some public-key encryption scheme, with pub being the public key and S being the corresponding private key. The goal of the prover is to convince the verifier that she knows S such that $\varphi(S, pub) = 1$ and that this S is stored on an individual machine. Moreover, this should be done in zero-knowledge, i.e., without revealing additional information to the verifier about S. We call a protocol that satisfies these requirements a *ZK-proof of individual knowledge (zkPIK)* for relation φ.

Let us now outline how to transform a public-coin PIK protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ into a zkPIK, where by "public-coin PIK" we mean protocols where the messages of the verifier are drawn at random, independently from S. It is easy to see that our $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ protocol (see Fig. 1) is public-coin since the only message that the verifier sends is the random challenge Z in Step 1. Our protocol works as follows. The prover and the verifier execute $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ with the following modification: the prover, instead of sending her messages in clear, commits to them using a cryptographic commitment scheme [9], and later proves in zero-knowledge [21] that she committed to messages that would make the verifier in protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ accept. Note that the zero-knowledge proof can be executed after the message exchange is finished; hence, the time needed to execute it does not influence the time bounds of the original PIK protocol. Clearly, this proof can also be executed in a non-interactive way [8], e.g., using one of the Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) schemes (see, e.g., [7]). Note that this solution assumes a generic use of zero-knowledge protocols.

Reducing the number of candidate messages. Recall that one of the parameters in PIK is the maximal size α_{O} of the set of "candidate secrets $\widehat{S}$", that besides the real value, S contains a lot of "false positives". Moreover, α_{O} can be quite large in our construction. A simple technique for eliminating the false positives is to let the verifier publish a hash $h = H'(S)$ (where H' is some hash function different from the one used in constructing PIK). Once h is known, everyone (including the prover) can check for every candidate $\widehat{S}$ if $H'(\widehat{S}) = h$, and if it does not hold, eliminate this $\widehat{S}$ from the candidate set. It is easy to see that $\widehat{S} = S$ passes this test, while (assuming no collisions in H are found) all $\widehat{S}$'s such that $\widehat{S} \neq S$ get eliminated. This solution can be proven secure in the random oracle model as long as the entropy of S (from the point of view of an external adversary) is large enough to guarantee that the adversary cannot guess S. This is the case, e.g., if S is a uniform random key.

Let us also discuss how h can be published. One option is to let h be sent to the verifier by the prover during the execution of the protocol. In the case of the eaPIK protocol, h would be encrypted by key K. Note that we do not even need the assumption that S has high entropy since it remains hidden from the external attacker. In the case of zkPIK, h could just be a part of the public key.

5 Individual Secret Sharing

In this section, we define and construct a secret sharing scheme that is "individually secure" in the following sense. Any secret S shared between the parties can be either reconstructed entirely by a single individual

party or it remains completely hidden. In other words: it is impossible to reconstruct the secret "partly" or to convert its shares to some shares from a standard secret sharing scheme (so that the parties could, e.g., perform some MPC computation on such standard shares). We call our scheme *individual secret sharing* (for a formal definition, see Def. 4). It is constructed using another primitive that we also define and construct, called individual *random* secret sharing scheme (IRSS, see Def. 5). The difference between ISS and IRSS is that in the latter, the shared secret is chosen randomly by the sharing procedure. Our schemes are passively secure, i.e., we do not address the problem of secure reconstruction in case some of the parties are actively cheating. A scheme that addresses this in the blockchain settings has been recently proposed in [18].

5.1 Standard Secret Sharing

Recall that secret sharing [30] is a protocol that allows a *dealer* to share a secret S between a group of parties. Let $a, t \in \mathbb{N}$ be such that $t \leq a$. A *(t,a)-secret sharing ((t,a)-SS)* is a pair of poly-time algorithms $(\mathsf{share}, \mathsf{rec})$ such that share is a randomized algorithm that takes as input $(S, 1^\kappa)$ (where $S \in \{0,1\}^*$ and 1^κ is a security parameter), and outputs a sequence $(S_1, \ldots, S_a, pub) \in (\{0,1\}^*)^{a+1}$ and rec takes as input a subset $\{S_{i_1}, \ldots, S_{i_t}\} \subseteq \{S_1, \ldots, S_a\}$ of size t and pub and outputs $\widehat{S} \in \{0,1\}^* \cup \{\perp\}$. Each S_i is called a *share of S*. The value pub is not a part of the standard secret sharing definition. We will use it to denote a public value (in our case: a ciphertext encrypted with a shared key) known to every party (think of it as included in every share or published by the dealer on some public bulletin board).

We require that if $\{S_{i_1}, \ldots, S_{i_t}\}$ and pub come from the output of $\mathsf{share}(S)$, then $\mathsf{rec}(\{S_{i_1}, \ldots, S_{i_t}, pub\}) = S$. The security of a (t,a)-SS is defined as follows. Let a *distinguisher* be a pair $(\mathcal{D}_1, \mathcal{D}_2)$ of poly-time machines such that:

- machine $\mathcal{D}_1$ takes as input 1^κ and outputs $S^0, S^1, Y \in \{0,1\}^*$ (with $|S^0| = |S^1|$) and
- machine $\mathcal{D}_2$ takes as input $Y \in \{0,1\}^*$, a subset of shares, and $pub \in \{0,1\}^*$ and outputs a bit.

We require that for every subset $\{i_1, \ldots, i_{t-1}\} \subseteq \{1, \ldots, a\}$ and every distinguisher $(\mathcal{D}_1, \mathcal{D}_2)$ it holds that

$$\begin{aligned} &\Pr\Big[\mathcal{D}_2(Y, \{S^0_{i_1}, \ldots, S^0_{i_{t-1}}\}, pub) = 1\Big] \\ -&\Pr\Big[\mathcal{D}_2(Y, \{S^1_{i_1}, \ldots, S^1_{i_{t-1}}\}, pub) = 1\Big] \leq \mathsf{negl}(\kappa), \end{aligned} \tag{6}$$

where for $b \in \{0,1\}$ we have $(S^b_1, \ldots, S^b_a, pub) \leftarrow\!\!\$\ \mathsf{share}(S^b, 1^\kappa)$. Note that the machine $\mathcal{D}_2$ learns pub "for free". The role of Y is to "pass" the information from $\mathcal{D}_1$ to $\mathcal{D}_2$ — one can think of $(\mathcal{D}_1, \mathcal{D}_2)$ as a single entity operating in two phases, and Y as the state of this entity between these phases.

We also define a *random* secret sharing to be a secret sharing scheme where the secret S is not chosen by the dealer but is selected randomly by the share algorithm together with the shares of S. Formally, a *(t,a)-random secret sharing ((t,a)-RSS)* is a pair of poly-time algorithms $(\mathsf{share}, \mathsf{rec})$ such that share is a randomized algorithm that takes as input 1^κ and outputs a sequence $(S, (S_1, \ldots, S_a))$, where $S \in \{0,1\}^\kappa$ and rec is the same as rec in the SS definition (except that there is no "pub" input). We require for $(S, (S_1, \ldots, S_a)) \leftarrow\!\!\$\ \mathsf{share}(1^\kappa)$ we always have $\mathsf{rec}(\{S_{i_1}, \ldots, S_{i_t}\}) = S$ and that for every $\{i_1, \ldots, i_{t-1}\} \subseteq \{1, \ldots, a\}$ and every poly-time distinguisher $\mathcal{D}$ that takes as input a secret S and a set of $t-1$ shares and outputs a bit, it holds that

$$|\Pr\big[\mathcal{D}(S^0, \{(S^0_{i_1}, \ldots, S^0_{i_1}\}) = 1\big] - \Pr\big[\mathcal{D}(S^1, \{S^0_{i_1}, \ldots, S^0_{i_1}\}) = 1\big]| \leq \mathsf{negl}(\kappa), \tag{7}$$

where $(S^0, (S^0_1, \ldots, S^0_a)) \leftarrow\!\!\$\ \mathsf{share}(1^\kappa)$ and $S^1 \leftarrow\!\!\$\ \{0,1\}^\kappa$. It is easy to see that every random secret sharing scheme can be converted into a standard secret sharing by using the random key S to encrypt the shared messages and making pub equal to the resulting ciphertext. We will use this technique is Sect. 5.4.

Examples. Let $(\mathbb{F}, \oplus, \times)$ be a finite field and let t, a be some parameter with $t \leq a$. The following is an (a,a)-random secret sharing: $\mathsf{share}_\oplus(1^\kappa) := (S_1, \ldots, S_a)$, and $\mathsf{rec}_\oplus(S_1, \ldots, S_a) := S_1 + \cdots + S_a$. Another

standard example is the Shamir's (t, a)-random secret sharing [30] $(\mathsf{share}_{\mathsf{Shamir}}, \mathsf{rec}_{\mathsf{Shamir}})$ defined as follows. Let $1, \ldots, a$ be some distinct elements of $\mathbb{F}$. Then $\mathsf{share}_{\mathsf{Shamir}}(1^\kappa) := ((1, P(1)), \ldots, (a, P(a)))$, where P is a random polynomial over $\mathbb{F}$ with degree at most $t-1$ and $\mathsf{rec}_{\mathsf{Shamir}}((i_1, X_1), \ldots, (i_t, X_t)) := P(0)$, where P is the polynomial of degree at most t interpolated at points $(i_1, X_1), \ldots, (i_t, X_t)$. It is easy to see that both of these schemes satisfy our definition in a very strong sense, namely Eq. (7) holds even if we substitute "$\mathsf{negl}(\kappa)$" with 0 (note that this also explains why the share algorithms do not depend on 1^κ).

5.2 ISS and IRSS Definitions

In this section, we present the definition of our main primitive, individual secret sharing (ISS), as well as the auxiliary notion of individual random secret sharing (where the shared secret is chosen randomly by the sharing algorithm). The main idea of the ISS definition is that we require that the scheme should be secure in the standard sense (in the random oracle model), and additionally, we require that for any distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ (with each poly-time machine $\mathcal{A}_b$ holding a share S_i of a secret S) that learns some information about the shared secret at least one $\mathcal{A}_b$'s learns the secret S entirely. As in Sect. 4, we consider an distributed adversary that operates in a *pre-processing phase*, and then in the *main phase*. The adversary interacts with an oracle Ω_H. She is σ-bounded, meaning that she can send at most σ slow queries to the oracle in the online phase. Unlike, in Sect. 4, in this definition, we do not make any restriction on the number of rounds in which the distributed adversary operates.

More concretely, suppose that $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ can distinguish between the shares of two secrets S^0 and S^1 with advantage $p_{\mathcal{A}}$. We then require that there exists an extractor $\mathcal{E}$ that extracts S entirely from the fast hash transcript of one of the $\mathcal{A}_i$'s with probability proportional to $p_{\mathcal{A}}$ (i.e., $\xi \cdot p_{\mathcal{A}}$, where ξ is the parameter of the scheme). Additionally, $\mathcal{E}$ gets as input string pub, which is assumed to be public according to the secret sharing definition (see Sect. 5.1). The "distinguishing game" is denoted as ISS and presented in Fig. 4 (a). Similarly to what we had in the PIK definition (see Def. 1), we can assume that the game is played by one of the adversaries, say $\mathcal{A}_1$. Our definition uses the notion of "extraction efficiency" in the same way as it was done in the PIK definition.

Definition 4. *For $\xi \in [0, 1]$ a (t, a, σ, ξ)-*individual secret sharing *((t, a, σ, ξ)-ISS) is a (t, a)-SS scheme with both share and rec having access to a random oracle $\Omega_{H:\{0,1\}^\lambda \to \{0,1\}^\kappa}$, and security holding against $(\mathcal{D}_1, \mathcal{D}_2)$ that have access to Ω_H. Consider a distributed adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ and the $\mathsf{ISS}(\sigma, b, \mathcal{A})$ experiment from Fig. 4 (a). Let*

$$p_{\mathcal{A}} := |\Pr[\mathsf{ISS}(\sigma, 0, \mathcal{A}) = 1] - \Pr[\mathsf{ISS}(\sigma, 1, \mathcal{A}) = 1]|. \tag{8}$$

We require that for every $\mathcal{A}$ there exists a poly-time extractor $\mathcal{E}$ such that for both $b \in \{0, 1\}$

$$\Pr\left[S^b \in \mathcal{E}(\mathcal{T}_1^{\mathsf{fast}}, pub) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a^{\mathsf{fast}}, pub)\right] \geq \xi \cdot p_{\mathcal{A}} - \mathsf{negl}(\kappa), \tag{9}$$

where pub comes from the $\mathsf{ISS}(\sigma, b, \mathcal{A})$ experiment and $\mathcal{T}_1^{\mathsf{fast}}, \ldots, \mathcal{T}_a^{\mathsf{fast}}$ are the fast hash transcripts of $\mathcal{A}_1, \ldots, \mathcal{A}_a$ (respectively) in this experiment. We say that $(\mathsf{share}, \mathsf{rec})$ *has* extraction efficiency *$(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$ if $|\mathcal{E}_1(\mathcal{T}_1^{\mathsf{fast}}, pub) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_a^{\mathsf{fast}}, pub)| \leq \alpha_{\mathsf{O}}$, and $\mathcal{E}$ operates in time at most α_{T} and uses space at most α_{S}. The parameters $\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}$ and α_{S} can be functions of some other parameters in the system.*

We also have the following.

Definition 5. *A (t, a, σ)-*random individual secret sharing *((t, a, σ)-ISS) is a (t, a)-RSS scheme with both share and rec having access to a random oracle $\Omega_{H:\{0,1\}^\lambda \to \{0,1\}^\kappa}$, and security holding against $\mathcal{D}$ that has access to Ω_H. Consider a distributed adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ and the $\mathsf{IRSS}(\sigma, b, \mathcal{A})$ experiment from Fig. 4 (b). Let*

$$p_{\mathcal{A}} := |\Pr[\mathsf{IRSS}(\sigma, 0, \mathcal{A}) = 1] - \Pr[\mathsf{IRSS}(\sigma, 1, \mathcal{A}) = 1]|. \tag{10}$$

We require that for every $\mathcal{A}$ there exists an extractor $\mathcal{E}$ such that

$$\Pr\left[S^0 \in \mathcal{E}(\mathcal{T}_1^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a^{\mathsf{fast}})\right] \geq p_{\mathcal{A}} - \mathsf{negl}(\kappa), \tag{11}$$

where $\mathcal{T}_1^{\mathsf{fast}}, \ldots, \mathcal{T}_a^{\mathsf{fast}}$ are the fast hash transcripts of $\mathcal{A}_1, \ldots, \mathcal{A}_a$ (respectively) in the $\mathsf{IRSS}(\sigma, b, \mathcal{A})$ experiment. The extraction efficiency is defined as in Def. 4.

Note that unlike Def. 4, this definition does not use the ξ parameter. This is done for the sake of simplicity, since, as it turns out, our construction (see Sect. 5.3) works without the need for this parameter. The extractors do not take *pub* as input, as in the *random* secret sharing definition this value does not exist.

$\mathsf{ISS}(\sigma, b, \mathcal{A})$

1. The sub-adversaries perform the pre-computation phase in which they can send an unbounded number of queries (including the slow ones) to the oracle Ω_H.
2. $\mathcal{A}_1$ chooses (S^0, S^1) (with $|S^0| = |S^1|$)
3. For $b \in \{0,1\}$ let $(S_1^b, \ldots, S_a^b, pub) \leftarrow\$ \mathsf{share}(S^b, 1^\kappa)$.
4. Each $\mathcal{A}_i$ receives (S_i^b, pub),
5. The sub-adversaries $\mathcal{A}_i$ engage in an interactive protocol with access to σ-bounded oracle Ω_H, at the end of which, $\mathcal{A}_1$ outputs $\widehat{b} \in \{0,1\}$. The experiment outputs $\widehat{b}$.

(a)

$\mathsf{IRSS}(\sigma, b, \mathcal{A})$

1. The sub-adversaries perform the pre-computation phase in which they can send an unbounded number of queries (including the slow ones) to the oracle Ω_H.
2. Let $(S^0, (S_1^0, \ldots, S_a^0)) \leftarrow\$ \mathsf{share}(S^0, 1^\kappa)$ and let $S^1 \leftarrow\$ \{0,1\}^\kappa$.
3. Each $\mathcal{A}_i$ receives S_i^0.
4. Proceed as in Step 5 in the $\mathsf{ISS}(\sigma, b, \mathcal{A})$ experiment above.

(b)

Fig. 4. (a) The ISS experiment and (b) the IRSS experiment. Above, $\mathcal{A}$ is a distributed adversary consisting of sub-adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_a$.

5.3 IRSS Construction

In this section, we construct an IRSS scheme $(\mathsf{share}, \mathsf{rec})$. Our scheme is presented in Fig. 5. It uses an arbitrary (standard) (t, a)-random secret sharing scheme $(\mathsf{share}_{\mathsf{std}}, \mathsf{rec}_{\mathsf{std}})$ as a building block. It is parameterized by the maximal number σ of slow queries available to the distributed adversary and a parameter $\eta > 2 \cdot \max(\kappa, \sigma)$. Its main idea is fairly simple: we first use a standard random secret sharing (RSS) scheme to share a random secret X. The shares in our individual random secret sharing are the same as those in the RSS, while the shared secret is equal to $\mathsf{iter}^{1 \mapsto \eta}(X)$. Here, $\mathsf{iter}^{1 \mapsto \eta}(X)$ is a procedure that simply applies H to itself η times, adding the index j to each hash (see Fig. 4 (a)). Intuitively, the security of our scheme is based on the fact that computing iter in a reasonable amount of time is impossible without one of the parties learning one of the intermediate values used in this computation. This is formalized in Lemma 3 below.

Lemma 3. *For any (t, a) suppose $(\mathsf{share}_{\mathsf{std}}, \mathsf{rec}_{\mathsf{std}})$ is a (t, a)-random secret sharing scheme. Then for any σ the scheme $(\mathsf{share}, \mathsf{rec})$ from Fig. 5 is a (t, a, σ)-individual random secret sharing with extraction efficiency $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, where $\mathbb{E}[\alpha_{\mathsf{O}}] \leq \kappa \cdot \ell / \eta$, $\mathbb{E}[\alpha_{\mathsf{T}}] \leq \ell$ (where ell is the number of fast hashes computed by $\mathcal{A}$ and time complexity is measured in the number of H evaluations), and $\alpha_{\mathsf{S}} = O(\kappa)$.*

Proof (sketch). We first show that $(\mathsf{share}, \mathsf{rec})$ is a (t, a)-RSS, i.e., that for any poly-time distinguisher $\mathcal{D}$ (having access to Ω_H) Eq. (6) holds. From the security of the underlying RSS scheme $(\mathsf{share}_{\mathsf{rnd}}, \mathsf{rec}_{\mathsf{rnd}})$ we know that X is computationally indistinguishable from random, given the knowledge of $t - 1$ shares. Hence the probability that $\mathcal{A}$ queries the oracle on X is negligible, and thus X^1 is uniform from $\mathcal{A}$'s point of view.

Procedure $\mathsf{iter}^{\alpha\mapsto\eta}(X)$

1. Let $X^{a-1} := X$.
2. For $j = \alpha$ to β let $X^j := H((j-1) \,||\, X^{j-1})$.
3. Output X^η.

(a)

Protocol $(\mathsf{share}, \mathsf{rec})$

The protocol is parameterized by a parameter σ denoting the maximal number of slow queries available to the distributed adversary and a parameter $\eta > 2 \cdot \max(\kappa, \sigma)$.

$\mathsf{share}(1^\kappa)$:

1. Let $(X, (S_1, \ldots, S_a)) \leftarrow\$ \mathsf{share}_{\mathsf{std}}(1^\kappa)$.
2. Output $(S, (S_1, \ldots, S_a))$, where $S = \mathsf{iter}^{1\mapsto\eta}(X)$.

$\mathsf{rec}(\{S_{i_1}, \ldots, S_{i_t}\})$:

1. Let $X := \mathsf{rec}_{\mathsf{std}}(\{S_{i_1}, \ldots, S_{i_t}\})$.
2. Output $S := \mathsf{iter}^{1\mapsto\eta}(X)$.

(b)

Fig. 5. The IRSS protocol $(\mathsf{share}, \mathsf{rec})$ that satisfies Def. 5 (see Lemma 3 for the proof). The main procedure is depicted in point (b). It is based on an RSS scheme $(\mathsf{share}_{\mathsf{std}}, \mathsf{rec}_{\mathsf{std}})$ it uses a sub-routine iter, see point (a).

By induction, the same is true for each X^j (for $j = 1, \ldots, \eta-1$) and for $S_\eta = H((\eta-1) \,||\, X^{j-1})$. Therefore $(\mathsf{share}, \mathsf{rec})$ is a (t, a)-RSS.

To finish the proof we construct a knowledge extractor $\mathcal{E}$ that satisfies the security definition of IRSS. The extractor is presented in Fig. 6. Let $\mathcal{Z}$ denote the event that all the values $(0||X^0), (1||X^1), \ldots, ((\eta-1)||X^{\eta-1})$ in the computation of $\mathsf{iter}^{1\mapsto\eta}(X)$ (see Fig. 5 (a)) appear the global hash transcript $\mathcal{T}$.

Knowledge extractor $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$

1. Let $\ell_n = |\mathcal{T}_b^{\mathsf{fast}}|$.
2. For each $i = 1, \ldots, \ell_b$:
   if $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(j \,||\, \widehat{X}^{j-1})$ (for some $j \in \{1, \ldots, \ell\}$ and some $\widehat{X}^{j-1} \in \{0,1\}^\kappa$) then with probability κ/η:
   output $\mathsf{iter}^{j\mapsto\eta}(\widehat{X})$ (but do not terminate)
   and call such $(j \,||\, \widehat{X}^{j-1})$ *selected*.

Fig. 6. Our knowledge extractor for IRSS. For its analysis see proof of Lemma 3.

Claim 5. For $\mathcal{Z}$ defined above we have that

$$|\Pr[\mathsf{IRSS}(\sigma, 0, \mathcal{A}) = 1 | \neg\mathcal{Z}] - \Pr[\mathsf{IRSS}(\sigma, 1, \mathcal{A}) = 1 | \neg\mathcal{Z}]| \leq \mathsf{negl}(\kappa). \tag{12}$$

Proof. The argument is similar to the one from the proof of Lemma 3. Suppose $\mathcal{Z}$ did not happen. This means that there exists j such that the adversary never queried Ω_H on $(j \,||\, X^j)$. Since $X^{j+1} = H(j \,||\, X^j)$

is uniform over $\{0,1\}^\kappa$ from $\mathcal{A}$'s point of view, thus the probability that $\mathcal{A}$ queries Ω_H on $(j+1, X^{j+1})$ is negligible. By induction, the same is true for each $(j'+1, X^{j'+1})$ (for $j' \geq j$) and for $S_b = H((\eta-1)\,||\,X^{j-1})$. Therefore S_0 is indistinguishable from S_1 and hence Eq. (10) holds. □

Claim 6. For $\mathcal{Z}$ defined above we have that

$$\Pr[S_b \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a) | \mathcal{Z}] \geq 1 - \mathsf{negl}(\kappa), \tag{13}$$

Proof (sketch). First, observe that since S and X^j's are random strings of length κ, thus, by an argument similar to the one in the proof of Thm. 1, Ω^H is queried on any of them in the pre-processing phase with negligible probability. Thus, we can assume that no hash query was computed in Step. 1. Assume that $\mathcal{Z}$ occurred. We have to show that with overwhelming probability $S \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a)$. For $S \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a)$ to happen, it suffices that at least one (j, X^j) was selected by the extractor (since $\mathsf{iter}^{1\mapsto\eta}(X) = \mathsf{iter}^{(j+1)\mapsto\eta}(X_j)$). Since we assumed that $\mathcal{Z}$ occurred, thus there are at least $\eta - \sigma$ fast queries containing (j, X^j)'s (note that $\eta - \sigma > 0$, since we assumed that $\eta > 2\sigma$). Each of them is selected with probability κ/η by some $\mathcal{E}(\mathcal{T}_b)$ (note that $\kappa/\eta < 1$, since we assumed that $\eta > 2\kappa$). Therefore the probability that at least one of them is selected is at least $1 - (1-\kappa/\eta)^{\eta-\sigma} = 1 - (1-\kappa/\eta)^{(\eta/\kappa)\cdot(\eta-\sigma)\cdot\kappa/\eta}$, which, by Lemma 1 is at least $1 - e^{-(\eta-\sigma)\cdot\kappa/\eta} = 1 - e^{-\kappa\cdot(1-\sigma/\eta)}$, which is overwhelming, since we assumed that $\eta > 2 \cdot \sigma$. □

Now, denote $q := \Pr[\mathcal{Z}]$. We have

$$\begin{aligned}
p_{\mathcal{A}} &= |\Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1] - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1]| && (14)\\
&= |\Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1|\neg\mathcal{Z}]\cdot(1-q) + \Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1|\mathcal{Z}]\cdot q\\
&\quad - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1|\neg\mathcal{Z}]\cdot(1-q) - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1|\mathcal{Z}]\cdot q|\\
&\leq |(1-q)\cdot(\Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1|\neg\mathcal{Z}] - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1|\neg\mathcal{Z}])| && (15)\\
&\quad + |q\cdot(\Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1|\mathcal{Z}] - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1|\mathcal{Z}])|\\
&\leq |(\Pr[\mathsf{IRSS}(\sigma,0,\mathcal{A})=1|\neg\mathcal{Z}] - \Pr[\mathsf{IRSS}(\sigma,1,\mathcal{A})=1|\neg\mathcal{Z}])| + q\\
&\leq \mathsf{negl}(\kappa) + q, && (16)
\end{aligned}$$

where Eq. (14) comes from the definition of $p_{\mathcal{A}}$ (see Eq. (10)), Eq. (15) comes from rearranging the terms and from the triangle inequality, and Eq. (16) comes from Claim 5. Hence

$$q \geq p_{\mathcal{A}} - \mathsf{negl}(\kappa) \tag{17}$$

By Claim 6 we have

$$\begin{aligned}
1 - \mathsf{negl}(\kappa) &\leq \Pr[S_b \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a) | \mathcal{Z}]\\
&= \Pr[S_b \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a) \wedge \mathcal{Z}] / \Pr[\mathcal{Z}]\\
&\leq \Pr[S_b \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a)] / q,
\end{aligned}$$

which implies that

$$q \cdot (1 - \mathsf{negl}(\kappa)) \leq \Pr[S_b \in \mathcal{E}(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a)]. \tag{18}$$

The last inequality, together with Eq. (17) implies that Eq. (11) holds. Hence, the only thing that remains to finish the proof of the lemma is to analyze the efficiency of the extractor. Let ℓ denote the total number of hashes computed by the adversary. First observe that the expected number of values selected by $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$ is at most $\ell_b \cdot \kappa/\nu$, and therefore $\mathbb{E}[\alpha_{\mathsf{O}}] = \mathbb{E}\big[|\mathcal{E}(\mathcal{T}_1^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a^{\mathsf{fast}})|\big] \leq \kappa \cdot \ell/\eta$.

To analyze the time and space complexity of $\mathcal{E}$ observe that $\mathcal{E}$ is an online algorithm that reads $\mathcal{T}_b^{\mathsf{fast}}$ once. Each iter operation takes at most η hash computations, and therefore $\mathbb{E}[\alpha_{\mathsf{T}}] \leq \eta \cdot \ell/\eta = \ell$. Clearly, this computation can be done in constant space (in the number of hash output blocks), and therefore $\alpha_{\mathsf{S}} = O(\kappa)$. This concludes the proof. □

5.4 ISS Construction

Let $(\mathsf{Enc}, \mathsf{Dec})$ a CPA-secure symmetric encryption scheme (see Sect. 2). Let $(\mathsf{share}^{\mathsf{rnd}}, \mathsf{rec}^{\mathsf{rnd}})$ be a (t, a, σ)-IRSS. Our ISS scheme is constructed on top of $(\mathsf{share}^{\mathsf{rnd}}, \mathsf{rec}^{\mathsf{rnd}})$ by using the random shared secret as a key for $(\mathsf{Enc}, \mathsf{Dec})$ and letting *pub* be equal to the ciphertext C. Our $(t, a, \sigma, 1/2)$-ISS scheme is presented in Fig. 7 its security is proven below.

Theorem 2. *The protocol* $(\mathsf{share}, \mathsf{rec})$ *is a* $(t, a, \sigma, 1/2)$*-ISS scheme. Moreover, suppose the extraction efficiency of* $(\mathsf{share}_{\mathsf{IRSS}}, \mathsf{rec}_{\mathsf{IRSS}})$ *is* $\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}$*, and* α_{S}*. Then the extraction efficiency of* $(\mathsf{share}, \mathsf{rec})$ *is* $\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}$*, and* $\alpha_{\mathsf{S}} = O(\kappa) + \mathcal{S}(\mathsf{Dec})$*, where* $\mathcal{S}(\mathsf{Dec})$ *denotes the space needed for running the* Dec *algorithm.*

ISS scheme $(\mathsf{share}, \mathsf{rec})$

$\mathsf{share}(1^\kappa, S)$:

1. Let $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}_{\mathsf{IRSS}}(1^\kappa)$.
2. Let $pub := \mathsf{Enc}(K, S)$.
3. Output $(S_1, \ldots, S_a, pub)$.

$\mathsf{rec}(\{S_{i_1}, \ldots, S_{i_t}\}, pub)$:

1. Let $K := \mathsf{rec}_{\mathsf{IRSS}}(\{S_{i_1}, \ldots, S_{i_t}\})$.
2. Output $\mathsf{Dec}(K, pub)$

Fig. 7. Our ISS scheme that satisfies Def. 4 (see Thm. 2 for the analysis).

Proof. We start with the following fact.

Claim 7. The protocol $(\mathsf{share}, \mathsf{rec})$ is a (t, a)-SS scheme.

Proof. We first show that $(\mathsf{share}, \mathsf{rec})$ is a (t, a)-SS scheme. Let $(\mathcal{D}_1, \mathcal{D}_2)$ be a distinguisher and let 1^κ be a security parameter. The proof proceeds by a "game hopping" argument. Consider the following sequence of probabilities:

- $r_0 := \Pr\left[\mathcal{D}_2(Y, \{S_{i_1}, \ldots, S_{i_{t-1}}\}, pub) = 1\right]$ where we let $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}_{\mathsf{IRSS}}(1^\kappa)$ and $(S^0, S^1, Y) \leftarrow\!\$\ \mathcal{D}_1(1^\kappa)$ and $pub := \mathsf{Enc}(K, S^0)$,
- $r_1 := \Pr\left[\mathcal{D}_2(Y, \{S_{i_1}, \ldots, S_{i_{t-1}}\}, pub) = 1\right]$ where we let $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}_{\mathsf{IRSS}}(1^\kappa)$ and $(S^0, S^1, Y) \leftarrow\!\$\ \mathcal{D}_1(1^\kappa)$ and $pub := \mathsf{Enc}(\widehat{K}, S^0), \widehat{K} \leftarrow\!\$\ \{0,1\}^\kappa$,
- $r_2 := \Pr\left[\mathcal{D}_2(Y, \{S_{i_1}, \ldots, S_{i_{t-1}}\}, pub = 1\right]$ where we let $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}_{\mathsf{IRSS}}(1^\kappa)$ and $(S^0, S^1, Y) \leftarrow\!\$\ \mathcal{D}_1(1^\kappa)$ and $pub := \mathsf{Enc}(\widehat{K}, S^1), \widehat{K} \leftarrow\!\$\ \{0,1\}^\kappa$, and
- $r_3 := \Pr\left[\mathcal{D}_2(Y, \{S_{i_1}, \ldots, S_{i_{t-1}}\}, pub = 1\right]$ where we let $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}_{\mathsf{IRSS}}(1^\kappa)$ and $(S^0, S^1, Y) \leftarrow\!\$\ \mathcal{D}_1(1^\kappa)$ and $pub := \mathsf{Enc}(K, S^1)$.

It is easy to see that $|r_0 - r_1| \leq \mathsf{negl}(\kappa)$, as otherwise one could create the following distinguisher $\mathcal{D}^{\mathsf{IRSS}}$ for the IRSS scheme (cf. Eq. (7)). Recall that $\mathcal{D}^{\mathsf{IRSS}}$ receives as input $(K, (S_{i_1}, \ldots, S_{i_{t-1}}))$ and its goal is to distinguish between the scenario (0) where $(K, (S_1, \ldots, S_a)) \leftarrow\!\$\ \mathsf{share}(1^\kappa)$ and scenario (1) where $K \leftarrow\!\$\ \{0,1\}^\kappa$ and $(S_1, \ldots, S_a)$ are sampled independently. The distinguisher $\mathcal{D}^{\mathsf{IRSS}}(K, (S_{i_1}, \ldots, S_{i_{t-1}})$ simply runs $\mathcal{D}_1(1^\kappa)$ to obtain (S_0, S_1, Y), computes $pub := \mathsf{Enc}(K, S^0)$, and outputs $b = \mathcal{D}_2(Y, \{S_{i_1}, \ldots, S_{i_{t-1}}\}, pub$. It is easy to see that in scenario (0) we have that $\Pr[b = 1] = r_0$ and in scenario (1) we have $\Pr[b = 1] = r_1$. Thus from

the security of IRSS, $|r_0 - r_1|$ has to be negligible. Using a symmetric argument we get that also $|r_2 - r_3|$ is negligible.

Let us now argue that $|r_1 - r_2| \leq \mathsf{negl}(\kappa)$. The only difference in between the probability formulas for r_1 and r_2 is that in the former pub is computed as $pub := \mathsf{Enc}(\widehat{K}, S^0)$ and in the later as $pub := \mathsf{Enc}(\widehat{K}, S^1)$. Hence, it is easy to see that any extinguisher that causes $|r_1 - r_2|$ to be non-negligible can be converted to a distinguisher that distinguishes between $\mathsf{Enc}(\widehat{K}, S^0)$ and $\mathsf{Enc}(\widehat{K}, S^0)$ with non-negligible advantage, which would contradict the CPA-security of $(\mathsf{Enc}, \mathsf{Dec})$. Hence it needs to hold that $|r_1 - r_2| \leq \mathsf{negl}(\kappa)$.

Altogether, we get that $|r_0 - r_3| \leq |r_0 - r_1| + |r_1 - r_2| + |r_2 - r_3| \leq \mathsf{negl}(\kappa)$. This finishes, the proof of the claim, since obviously $|r_0 - r_3|$ is equal to Eq. (6). □

To finish the proof of the lemma, we need to prove the following.

Claim 8. Consider a distributed adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ and the $\mathsf{ISS}(\sigma, b, \mathcal{A})$ experiment from Fig. 4 (a). Then for every $\mathcal{A}$ there exists an extractor $\mathcal{E}$ such that for both $b \in \{0, 1\}$

$$\Pr[S_b \in \mathcal{E}(\mathcal{T}_1, pub) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a, pub)] \geq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa), \tag{19}$$

where $\mathcal{T}_1, \ldots, \mathcal{T}_a$ are the fast hash transcripts of $\mathcal{A}_1, \ldots, \mathcal{A}_a$ (respectively) in the experiment $\mathsf{ISS}(\sigma, b, \mathcal{A})$ and $p_{\mathcal{A}} := |\Pr[\mathsf{ISS}(\sigma, 0, \mathcal{A}) = 1] - \Pr[\mathsf{ISS}(\sigma, 1, \mathcal{A}) = 1]|$. Moreover $\mathcal{E}$ has the same extraction efficiency as the extractor that extracts the secrets in the IRSS attack on $(\mathsf{share}^{\mathsf{rnd}}, \mathsf{rec}^{\mathsf{rnd}})$.

Proof. We again use the game-hopping technique. Define the following experiments.

- ISS_0 is the same as $\mathsf{ISS}(\sigma, 0, \mathcal{A})$,
- ISS_1 is the same as ISS_0, except that the $pub := \mathsf{Enc}(\widehat{K}, S^0)$, where we have $\widehat{K} \leftarrow\!\!\$\, \{0,1\}^\kappa$,
- ISS_2 is the same as ISS_1, except that $pub := \mathsf{Enc}(\widehat{K}, S^1)$, and
- ISS_3 is the same as ISS_3, except that $pub := \mathsf{Enc}(K, S^1)$. Note that $\mathsf{ISS}_3 = \mathsf{ISS}(\sigma, 1, \mathcal{A})$.

Using the same argument as in the proof of Claim 7, from the CPA-security of $(\mathsf{Enc}, \mathsf{Dec})$ we get that $|\Pr[\mathsf{ISS}_1 = 1] - \Pr[\mathsf{ISS}_2 = 1]| \leq \mathsf{negl}(\kappa)$. Let $r := |\Pr[\mathsf{ISS}_0 = 1] - \Pr[\mathsf{ISS}_1 = 1]|$ and $r' := |\Pr[\mathsf{ISS}_2 = 1] - \Pr[\mathsf{ISS}_3 = 1]|$. We have

$$\begin{aligned} p_{\mathcal{A}} &= |\Pr[\mathsf{ISS}(\sigma, 0, \mathcal{A}) = 1] - \Pr[\mathsf{ISS}(\sigma, 1, \mathcal{A}) = 1]| \\ &\leq |\Pr[\mathsf{ISS}_0 = 1] - \Pr[\mathsf{ISS}_1 = 1]| + |\Pr[\mathsf{ISS}_1 = 1] - \Pr[\mathsf{ISS}_2 = 1]| \\ &\quad + |\Pr[\mathsf{ISS}_2 = 1] - \Pr[\mathsf{ISS}_3 = 1]| \\ &\leq r + \mathsf{negl}(\kappa) + r', \end{aligned} \tag{20}$$

where Eq. (20) follows from the triangle inqaulity. This implies that $r \geq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa)$ or $r' \leq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa)$. Without loss of generality, assume that $r \geq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa)$. Clearly, since $\widehat{K} \leftarrow\!\!\$\, \{0,1\}^\kappa$, by an argument similar to the one used in the proof of Claim 7, $\mathcal{A}$ can be converted to an adversary $\mathcal{A}'$ that wins the IRSS game against $(\mathsf{share}_{\mathsf{IRSS}}, \mathsf{rec}_{\mathsf{IRSS}})$ with advantage $p_{\mathcal{A}}/2$, i.e., such that Eq. (10) holds with $p_{\mathcal{A}}/2$. Using the assumption that $(\mathsf{share}_{\mathsf{IRSS}}, \mathsf{rec}_{\mathsf{IRSS}})$ is (t, a, σ)-secure, we get that there exists an extractor $\mathcal{E}'$ such that

$$\Pr[K \in \mathcal{E}'(\mathcal{T}_1) \cup \cdots \cup \mathcal{E}'(\mathcal{T}_a)] \geq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa).$$

where $\mathcal{T}_1, \ldots, \mathcal{T}_a$ are the fast hash transcripts of $\mathcal{A}_1, \ldots, \mathcal{A}_a$ (respectively). Extractor $\mathcal{E}(\mathcal{T}_m, pub)$ simply runs $\mathcal{E}'(\mathcal{T}_m)$, and for every $K \in \mathcal{E}'(\mathcal{T}_m)$ it outputs $\mathsf{Dec}(K, pub)$. Since $pub = \mathsf{Dec}(K, S^0)$, thus if $b = 0$ we get

$$\Pr[S_b \in \mathcal{E}(\mathcal{T}_1, pub) \cup \cdots \cup \mathcal{E}(\mathcal{T}_a, pub)] \geq p_{\mathcal{A}}/2 - \mathsf{negl}(\kappa). \tag{21}$$

It is also easy to see that this probability does not depend on b (since in the ISS experiment only pub depends on b). Hence Eq. (21) holds also for $b = 1$.

Clearly, $\mathcal{E}$ performs the same number of hashes as $\mathcal{E}'$ and has the same output size. Hence the α_{O} and α_{T} of $\mathcal{E}$ remains the same. For the space complexity, we need to add the space $\mathcal{S}_{\mathsf{Dec}}$ that is needed for the evaluation of Dec. □

These two claims complete the proof of the lemma. □

ISS scheme $(\mathsf{share}^{\mathsf{ISS}}_{\oplus}, \mathsf{rec}^{\mathsf{ISS}}_{\oplus})$

$\mathsf{share}(1^\kappa, S)$:

1. Let $(S_1, \ldots, S_n) \leftarrow\$\ \mathbb{F}^n$.
2. Let $K := \mathsf{iter}^{1\mapsto\eta}(S_1 + \cdots + S_n)$.
3. Let $pub := \mathsf{Enc}(K, S)$.
4. Output $(S_1, \ldots, S_n, pub)$.

$\mathsf{rec}(1^\kappa, (S_1, \ldots, S_n, pub))$:

1. Let $K := \mathsf{iter}^{1\mapsto\eta}(S_1 + \cdots + S_n)$.
2. Output $\mathsf{Dec}(K, pub)$.

(a)

ISS scheme $(\mathsf{share}^{\mathsf{ISS}}_{\mathsf{Shamir}}, \mathsf{rec}^{\mathsf{ISS}}_{\mathsf{Shamir}})$

$\mathsf{share}(1^\kappa, S)$:

1. Let $((1, P(1)), \ldots, (a, P(a)))$, where P is a random polynomial over $\mathbb{F}$ with degree at most $t-1$.
2. Let $K := \mathsf{iter}^{1\mapsto\eta}(P(0))$.
3. Let $pub := \mathsf{Enc}(K, S)$.
4. Output $(S_1, \ldots, S_n, pub)$.

$\mathsf{rec}(1^\kappa, (S_{1_t}, \ldots, S_{i_t}, pub))$:

1. Let P be the polynomial of degree at most t interpolated at points $(i_1, X_1), \ldots, (i_t, X_t)$.
2. Let $K := \mathsf{iter}^{1\mapsto\eta}(P(0))$.
3. Output $\mathsf{Dec}(K, pub)$.

(b)

Fig. 8. Two concrete examples of ISS schemes: (a) an (a, a) scheme based on $(\mathsf{share}_{\oplus}, \mathsf{rec}_{\oplus})$ and (b) a (t, a) scheme based on $(\mathsf{share}_{\mathsf{Shamir}}, \mathsf{rec}_{\mathsf{Shamir}})$ (see Sect. 5.1). Above $\mathbb{F}$ is a finite field and $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure symmetric encryption scheme. Procedure iter is defined on Fig. 5 (a).

5.5 Concrete examples

Concrete examples of ISS schemes are presented on Fig. 8.

Lemma 4. *Suppose* $(\mathsf{Enc}, \mathsf{Dec})$ *is a CPA-secure symmetric encryption scheme such that* Dec *works in space* $O(\kappa)$. *Then for any* a, t, σ, *and* $\eta \cdot \max(\kappa, \sigma)$ *we have that (a)* $(\mathsf{share}^{\mathsf{ISS}}_{\oplus}, \mathsf{rec}^{\mathsf{ISS}}_{\oplus})$ *is an* $(a, a, \sigma, 1/2)$*-ISS and (b)* $(\mathsf{share}^{\mathsf{ISS}}_{\mathsf{Shamir}}, \mathsf{rec}^{\mathsf{ISS}}_{\mathsf{Shamir}})$ *is an* $(t, a, \sigma, 1/2)$*-ISS and with extraction efficiency* $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, *where* $\mathbb{E}[\alpha_{\mathsf{O}}] \leq \kappa \cdot \ell/\eta$, $\mathbb{E}[\alpha_{\mathsf{T}}] \leq \ell$ *and* $\alpha_{\mathsf{S}} = O(\kappa) + \mathcal{S}(\mathsf{Dec})$ *with* $\mathcal{S}(\mathsf{Dec})$ *denoting space needed for running the* Dec *algorithm and* ℓ *denoting the number of fast hashes computed by* $\mathcal{A}$.

Proof. As explained in Sect. 5.1 $(\mathsf{share}_{\oplus}, \mathsf{rec}_{\oplus})$ and $(\mathsf{share}_{\mathsf{Shamir}}, \mathsf{rec}_{\mathsf{Shamir}})$ are (a, a)- and (t, a)-RSS Schemes (respectively). Hence, by Lemma 3, one can build an (a, a, σ)-IRSS scheme $(\mathsf{share}^{\mathsf{IRSS}}_{\oplus}, \mathsf{rec}^{\mathsf{IRSS}}_{\oplus})$ and a (t, a, σ)-IRSS scheme $(\mathsf{share}^{\mathsf{IRSS}}_{\mathsf{Shamir}}, \mathsf{rec}^{\mathsf{IRSS}}_{\mathsf{Shamir}})$ from them (respectively) using the transformation from Fig. 5. The resulting schemes have $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, where $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, where $\mathbb{E}[\alpha_{\mathsf{O}}] \leq \kappa \cdot \ell/\eta$, $\mathbb{E}[\alpha_{\mathsf{T}}] \leq \ell$ and $\alpha_{\mathsf{S}} = O(\kappa)$

The $(\mathsf{share}^{\mathsf{ISS}}_{\oplus}, \mathsf{rec}^{\mathsf{ISS}}_{\oplus})$ and $(\mathsf{share}^{\mathsf{ISS}}_{\mathsf{Shamir}}, \mathsf{rec}^{\mathsf{ISS}}_{\mathsf{Shamir}})$ schemes come from applying the transformation from Fig. 7 to $(\mathsf{share}^{\mathsf{IRSS}}_{\oplus}, \mathsf{rec}^{\mathsf{IRSS}}_{\oplus})$ and $(\mathsf{share}^{\mathsf{IRSS}}_{\mathsf{Shamir}}, \mathsf{rec}^{\mathsf{IRSS}}_{\mathsf{Shamir}})$ (respectively). By Lemma 3 they are $(a, a, \sigma, 1/2)$- and

$(t, a, \sigma, 1/2)$-ISS schemes (respectively) and their extraction efficiency is $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, where α_{O} and α_{T} are as before and $\alpha_{\mathsf{S}} = O(\kappa) + \mathcal{S}(\mathsf{Dec})$. □

Conclusion

In this paper, we initiated the formal study of individual cryptography. It would be exciting to search for particular computational problems that are MPC- or TEE-hard in practice (one good candidate would be functions that require many memory lookups), and to improve the practical parameters that we achieve. This would be, in a sense, a complementary effort to the search for "MPC-friendly primitives" (see, e.g., [23]).

References

1. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: Scrypt is maximally memory-hard. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 33–62 (2017). https://doi.org/10.1007/978-3-319-56617-7_2, https://doi.org/10.1007/978-3-319-56617-7_2
2. Bahmani, R., Barbosa, M., Brasser, F., Portela, B., Sadeghi, A., Scerri, G., Warinschi, B.: Secure multiparty computation from SGX. In: Kiayias, A. (ed.) Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10322, pp. 477–497. Springer (2017). https://doi.org/10.1007/978-3-319-70972-7_27, https://doi.org/10.1007/978-3-319-70972-7_27
3. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 390–420. Springer (1992). https://doi.org/10.1007/3-540-48071-4_28, https://doi.org/10.1007/3-540-48071-4_28
4. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. pp. 62–73. ACM (1993). https://doi.org/10.1145/168588.168596
5. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Santis, A.D. (ed.) Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings. Lecture Notes in Computer Science, vol. 950, pp. 92–111. Springer (1994). https://doi.org/10.1007/BFb0053428
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
7. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the SNARK. J. Cryptol. **30**(4), 989–1066 (2017). https://doi.org/10.1007/s00145-016-9241-9
8. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 103–112. ACM (1988). https://doi.org/10.1145/62212.62222
9. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. **37**(2), 156–189 (1988). https://doi.org/10.1016/0022-0000(88)90005-0
10. Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. Cryptology ePrint Archive, Paper 2021/1423 (2021), https://eprint.iacr.org/2021/1423, https://eprint.iacr.org/2021/1423
11. Chandran, N., Goyal, V., Moriarty, R., Ostrovsky, R.: Position-based cryptography. SIAM J. Comput. **43**(4), 1291–1341 (2014). https://doi.org/10.1137/100805005
12. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214
13. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing traitors. IEEE Trans. Inf. Theory **46**(3), 893–910 (2000). https://doi.org/10.1109/18.841169

14. Damgård, I.: Towards practical public key systems secure against chosen ciphertext attacks. In: Feigenbaum, J. (ed.) Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 445–456. Springer (1991). https://doi.org/10.1007/3-540-46766-1_36, https://doi.org/10.1007/3-540-46766-1_36
15. Dubhashi, D.P., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press (2009), http://www.cambridge.org/gb/knowledge/isbn/item2327542/
16. Dwork, C., Naor, M., Wee, H.: Pebbling and proofs of work. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3621, pp. 37–54. Springer (2005). https://doi.org/10.1007/11535218_3, https://doi.org/10.1007/11535218_3
17. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 585–605. Springer (2015). https://doi.org/10.1007/978-3-662-48000-7_29, https://doi.org/10.1007/978-3-662-48000-7_29
18. Dziembowski, S., Faust, S., Lizurej, T.: Secret sharing with snitching, manuscript
19. Dziembowski, S., Kazana, T., Wichs, D.: One-time computable self-erasing functions. In: Ishai, Y. (ed.) Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6597, pp. 125–143. Springer (2011). https://doi.org/10.1007/978-3-642-19571-6_9, https://doi.org/10.1007/978-3-642-19571-6_9
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 218–229. ACM (1987). https://doi.org/10.1145/28395.28420
21. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012
22. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 252–282. Springer (2022). https://doi.org/10.1007/978-3-030-97121-2_10, https://doi.org/10.1007/978-3-030-97121-2_10
23. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: Mpc-friendly symmetric key primitives. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 430–443. ACM (2016). https://doi.org/10.1145/2976749.2978332
24. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014), https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269
25. Kelkar, M., Babel, K., Daian, P., Austgen, J., Buterin, V., Juels, A.: Complete knowledge: Preventing encumbrance of cryptographic secrets. Cryptology ePrint Archive, Paper 2023/044 (2023), https://eprint.iacr.org/2023/044, https://eprint.iacr.org/2023/044
26. Kiayias, A., Tang, Q.: How to keep a secret: leakage deterring public-key cryptosystems. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 943–954. ACM (2013). https://doi.org/10.1145/2508859.2516691
27. Mangipudi, E.V., Lu, D., Kate, A.: Collusion-deterrent threshold information escrow. IACR Cryptol. ePrint Arch. p. 95 (2021), https://eprint.iacr.org/2021/095
28. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**(2), 122–144 (2004). https://doi.org/10.1016/j.jalgor.2003.12.002
29. Puddu, I., Lain, D., Schneider, M., Tretiakova, E., Matetic, S., Capkun, S.: Teevil: Identity lease via trusted execution environments. CoRR **abs/1903.00449** (2019), http://arxiv.org/abs/1903.00449
30. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176
31. Young, A.L., Yung, M.: Cryptovirology: Extortion-based security threats and countermeasures. In: 1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA. pp. 129–140. IEEE Computer Society (1996). https://doi.org/10.1109/SECPRI.1996.502676

A Appendix

A.1 Proof of Lemma 2

We will use the following versions of the Chernoff–Hoeffding bounds (see, e.g., Thm. 1.1 in [15]).

Lemma 5. *Let $U_1, \ldots, U_C$ be random variables independently distributed over $[0,1]$ and let $U = U_1 + \cdots + U_C$. Then for all $\varepsilon > 0$ we have*

(a) $\Pr[U < (1-\varepsilon)\,\mathbb{E}[U]] \leq \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/2)$ *and*
(b) $\Pr[U > (1+\varepsilon)\,\mathbb{E}[U]] \leq \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/3)$.

Clearly each $\mathbb{E}[U_i] = 2^{-\zeta}$ and hence $\mathbb{E}[U] = C \cdot 2^{-\zeta}$. Let us start with proving Point (a). Assume $C = \kappa \cdot 2^{\zeta+1}$. This implies that $\mathbb{E}[U] > \kappa \cdot 2^{\zeta+1} \cdot 2^{-\zeta} = 2\kappa$. Set $\varepsilon := 1/4$. We get

$$\begin{aligned}
\Pr[U \leq \kappa] &\leq \Pr[U \leq (1/2) \cdot \mathbb{E}[U]] && (22)\\
&= \Pr[U < (1-\varepsilon)\,\mathbb{E}[U]] && (23)\\
&\leq \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/2) && (24)\\
&= \exp(-\kappa/16) && (25)\\
&\leq \mathsf{negl}(\kappa). && (26)
\end{aligned}$$

Hence Point (a) of Lemma 2 is proven. Let us now proceed to proving Point (b). Assume $C \leq \kappa \cdot 2^{\zeta-1}$. This implies that

$$\kappa > C \cdot 2^{-\zeta+1} \tag{27}$$

Set $\varepsilon := \sqrt{9 \cdot \kappa \cdot 2^{\zeta-5}/C}$. Using the assumption that $C \leq \kappa \cdot 2^{\zeta-1}$ we get that

$$\begin{aligned}
\varepsilon &> \sqrt{9 \cdot \kappa \cdot 2^{\zeta-5}/(\kappa \cdot 2^{\zeta-1})} && (28)\\
&= 3/4 && (29)
\end{aligned}$$

We hence get

$$\begin{aligned}
\Pr[U \geq \kappa] &\leq \Pr\big[U \geq C \cdot 2^{-\zeta+1}\big] && (30)\\
&= \Pr[U \geq (1-1/2)\,\mathbb{E}[U]] && (31)\\
&\leq \Pr[U > (1-\varepsilon)\,\mathbb{E}[U]] && (32)\\
&\leq \exp(-(9 \cdot \kappa \cdot 2^{\zeta-5}/C) \cdot (C \cdot 2^{-\zeta})/2) && (33)\\
&= \exp(-9 \cdot \kappa \cdot 2^{-6}) && (34)\\
&\leq \mathsf{negl}(-\kappa), && (35)
\end{aligned}$$

where in Eq. (30) we used Eq. (27), in Eq. (32) — Eq. (29), and in Eq. (33) — Lemma 5 (Point (b)). Hence Point (b) of Lemma 2 is proven. □