Beholder Signatures

Stefan Dziembowski[1,2], Sebastian Faust[3], Paweł Kedzior[1], Marcin Mielniczuk[1], Susil Kumar Mohanty[1], and Krzysztof Pietrzak[4]

[1] University of Warsaw
[2] IDEAS Institute
[3] Technische Universität Darmstadt
[4] Institute of Science and Technology Austria

Abstract. We introduce a new primitive, called *beholder signatures*, which, in some sense, are the opposite of blind signatures. In a beholder signature, one signs a commitment to a (potentially very long) message, and the signature attests that the parties participating in the signing process who know the secret key, jointly also know the entire committed message. This guarantee holds even against distributed adversaries that use secure multi-party computation (MPC) to produce the signature. We work in the *distributed adversarial model* (Dziembowski, Faust, and Lizurej, Crypto'23), where one assumes that it is infeasible to evaluate a large number of hash queries without any of the participating parties learning the input. We propose a construction of beholder signatures in the random oracle model. The starting point of our construction is proofs of complete knowledge, recently proposed by (Kelkar et al. CCS'24), which again build on Fischlin's transformation of a sigma protocol to a noninteractive, straight-line extractable zero-knowledge proof of knowledge. Our scheme is concretely efficient and comes with a proof-of-concept implementation using Schnorr as the underlying sigma protocol.
The primary applications of beholder signatures can be found within the blockchain ecosystem. In particular, we describe how to use them to construct *proofs of custody* (Feist, 2021) that do not require ephemeral keys and are noninteractive. We also outline applications to *data dissemination*, *data availability*, and *proofs of replication*.

1 Introduction

The notion of *knowledge* in cryptography is a fundamental concept that has been studied for several decades, starting from the seminal work on *proofs of knowledge* [21,2], which are cryptographic protocols that allow one party (the *prover*) to convince another party (the *verifier*) that she knows some secret (typically a secret key related to some known public key) without revealing it. Digital signature schemes like the one of Schnorr [34] are derived by taking a special type of proof of knowledge and making it non-interactive by using hashing to derive a challenge (known as the Fiat-Shamir heuristic) and bind the message to be signed to the proof. A related notion are succinct non-interactive arguments of knowledge (SNARKs) [5], which allow to prove knowledge of a value with a short proof, even if the value itself is very long (often there's a succinct commitment, and the SNARK proves knowledge of the long committed message). Recently, it has been observed in [13,26] that standard definitions of knowledge do not prevent a particular type of attack that we will refer to as an *MPC attack*. Specifically, the authors of [13,26] note that these definitions only require that the value $\mathbf{v}$ (for which we want to prove knowledge) is known to some abstract prover without guaranteeing that it is known entirely to a *single* entity. In particular, nothing prevents the prover from being emulated by a group of parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, each knowing only a share of $\mathbf{v}$ (here, a share can be a share from a secret-sharing scheme, or literally just a part of $\mathbf{v}$). These parties can jointly emulate the prover's computation using a secure multi-party computation (MPC) protocol [38,3], with one party, say P_1, handling communication with the verifier. In this scenario, the verifier will accept even though no single party P_i knows $\mathbf{v}$. Given the efficiency of MPC for a wide range of functionalities, this observation raises the questions of whether (1) the above "attack" is of practical concern and (2) whether the standard knowledge definitions can be extended to prevent such an MPC attack.

Perhaps surprisingly, [13,26] answer these questions affirmatively. They show that the attack is practical and provide real-life scenarios where it can compromise the security of digital systems. These scenarios

include deniable messaging, electronic voting, and subscription-based online services. They also propose a new definition for proofs of knowledge that excludes such attacks, and they provide constructions that satisfy this definition. Their main idea is to construct proofs that require significant computation from the prover, and thus, it's infeasible to execute the proof in an MPC setting. The main building block in those constructions is a hash function H. Typical cryptographic hash functions like SHA-256 do not exhibit homomorphic properties, which makes their evaluation using MPC costly. Moreover, due to their popularity in Bitcoin mining, application-specific integrated circuits (ASICs) that can evaluate SHA-256 at an extremely high rate and low cost are pretty common. For the security proof, the authors of [13] model H as a random oracle. They impose some additional assumptions to make the model as realistic as possible. First, they require H to work only on short inputs so that we can think of it as a monolithic compression function. This rules out a large class of attacks, such as ones based on the length extension property of Merkle-Damgård hash functions. Second, they allow $\mathsf{P}_1, \ldots, \mathsf{P}_n$ to perform some bound, but still fairly large, number of computations of H using MPC (via so-called *slow oracle queries*). Since the prover's algorithm requires a large number of H computations, this means that $\mathsf{P}_1, \ldots, \mathsf{P}_n$ need to perform many H computations "in plain" (i.e., *not* using MPC). In the formal model, such computations are modeled as *fast oracle queries* to H (and the number of such queries is unbounded). It is assumed that for each fast query, at least one of the parties P_i must know the entire input sent to the oracle. The authors of [13] call this model *individual cryptography* and construct a proof of knowledge that is secure in this model. Their construction is based on forcing the prover to perform many fast oracle queries to H on different blocks of the value $\mathbf{v}$, so that at least one of the parties P_i knows the entire $\mathbf{v}$.

Going more into the detail, the authors [13] define and construct a protocol that they call *proof of individual knowledge (PIK)*. In this protocol, both the prover and the verifier know some message $\mathbf{m}$, and the goal is for the verifier to convince the prover that $\mathbf{m}$ is stored on a single machine. Despite being an important starting point, these works suffer from certain shortcomings. First, the PIK protocol of [13] assumes that the verifier knows $\mathbf{m}$, and uses this knowledge in the verification procedure. The authors outline how to convert their protocol into one where only the prover knows $\mathbf{m}$ while the verifier only has some short commitment to it. However, no formal definition or proof of correctness is given. Moreover, it requires the parties to run generic zero-knowledge protocols over large data (with respect to its hash), which is inefficient in practice (and hard to model in theory). Another drawback is that the scheme of [13] is secure assuming that the number of communication rounds between the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ is bounded. We outline the construction from [26], termed *proofs of complete knowledge*, below, when we sketch the construction of our new primitive. Both constructions [13,26] are only applicable when the value for which we want to prove knowledge is short, which limits their applicability as a standalone primitive, where we often want to attest knowledge of a long message.

1.1 Our new primitive: beholder signatures

In this paper, we address these issues by defining and constructing a new primitive, which we call *beholder signatures*.[5] A beholder signature is a signature that is verified with respect to a *commitment* to a message. In addition to the standard security notions for signatures (such as existential unforgeability under chosen message attacks), it also attests the knowledge of the entire committed message, even if it is very long. In the simplest setting, in which the presumed signer never shares their secret key—which in some settings can be incentivized—they must *individually* know the entire message. In general, the property of *individual extractability* ensures that the parties who participated in the signing process and knew the secret key (in the sense that they could extract the secret key from the queries they locally made to the random oracle) jointly know the entire committed message. In this sense, beholder signatures can be thought of as the opposite of *blind signatures*. For technical reasons, it is useful to consider a relaxed variant of this property, in which the parties are only guaranteed to know a *fraction* γ (typically, $\gamma = 1/2$) of the committed message. By encoding

[5] According to the Cambridge dictionary, a beholder is "a person who sees or looks at someone or something". This name reflects the fact that the signer must "see" the entire message they sign.

the message using an erasure code before it is committed, one can still conclude that the parties know the entire message.

Although our construction is proven in the individual cryptography model, i.e., taking into account that some bounded number of hash computations can be done using MPC (modeled by "slow queries"), we note that it makes sense to consider beholder signatures also in the settings where MPC attacks are not a realistic concern, i.e., when the number of allowed slow queries is zero (see Section 2 for an example of such an application). Our construction in this case is exactly the same, and one can relax the security parameters in this case, cf. Section 6. From this perspective, one can think of our beholder signatures as a relaxation of the notion of signatures of knowledge [7] for a particular relation, cf. Section 1.3.

An overview of our construction. Our starting point is a sigma protocol [8], which is a three-round, interactive zero-knowledge proof of knowledge. The first message a (called the *commitment*) is sent by the prover to the verifier. The verifier then replies with a random challenge c, and the prover responds with z. The verifier can efficiently compute a predicate on the transcript (a, c, z) to decide whether to accept.

A sigma protocol is assumed to be *specially sound*: from any two accepting transcripts (a, c, z) and (a, c', z'), sharing the same commitment but having different challenges, one can efficiently extract the secret witness. The protocol is also assumed to be *zero-knowledge*: for any c, there is an efficient way to sample accepting transcripts (a, c, z). A sigma protocol can be made non-interactive by applying the Fiat-Shamir transform, which removes interaction by computing the challenge as a hash of the first message and the statement x, i.e., $c = \mathsf{H}(x, a)$. In the random oracle model, this remains a proof of knowledge: one can run the prover to obtain a triple $(a, c = \mathsf{H}(x, a), z)$, then rewind the prover to the point where they query the random oracle on input (x, a) and give a different reply $c' \neq c$. From the two transcripts, we can extract the secret. This can be turned into a signature scheme by additionally hashing the message to be signed.

Fischlin [20] proposed an alternative transform which, unlike Fiat-Shamir is straight-line extractable. The key idea is to force the (potentially malicious) prover to hash at least two accepting tuples (a, c, z) and $(a, c' \neq c, z')$ while creating the proof. Due to special soundness, one can extract the secret from the prover's queries. In Fischlin's construction, the prover first samples a vector $\boldsymbol{a} = (a_1, \ldots, a_\rho)$ of first messages, and must then, for each a_i, find c_i, z_i so that (a_i, c_i, z_i) is accepting. Moreover, for each i, $\mathsf{H}(x, \boldsymbol{a}, i, c_i, z_i)$ must be small, e.g., start with d zeros; this is a straightforward instance of a proof of work (PoW). Unless the prover is extremely lucky and, for every i, the first random oracle query passes this test—which happens only with probability $2^{-\rho \cdot d}$—we can extract the secret from the inputs to the random oracle. Crucially, the vector $\boldsymbol{a}$ is hashed in each query, essentially tying the prover to its choice of $a_1, \ldots, a_\rho$ and ensures a malicious prover cannot try many different a's and keep only those where they succeed on their first try.

Straight-line extraction is useful because rewinding is problematic in the quantum setting or for composition, and also when proving "complete" or "individual" knowledge. The construction of non-interactive proofs of complete knowledge from [26] uses Fischlin's transform, but with much larger parameters. Thus, even if the prover is allowed to hide many of the queries from the extractor (as in MPC or a TEE), extraction is still possible. This proof of complete knowledge is the starting point for our construction of beholder signatures, as shown in Figure 2. To obtain a beholder signature, we must ensure that we can extract not only the secret key, but also a fraction of some data $D = D[1], \ldots, D[N]$ (think of each $D[i]$ as a block of, say, 256 bits). Let com be a commitment to D that allows for local opening, and op_i be the opening of the data block $D[i]$.

A first idea is to augment the hash $\mathsf{H}(x, \boldsymbol{a}, i, c_i, z_i)$ by always adding a random data block to the input, i.e., $\mathsf{H}(x, \boldsymbol{a}, i, c_i, z_i, D[j])$, where the index j is a hash of the challenge, $j = \mathsf{H}(i, c_i)$. If we set the parameters ρ and d large enough, this should force any prover to include a large fraction of D into the random oracle inputs so that they can be extracted. This initial idea has two main issues: First, a malicious prover could make many hash queries using invalid data blocks, which would make extracting the message impossible. To prevent these wrong inputs from confusing the extractor, we require that for each hashed data block, the signature must also include the opening information, i.e., we use a hash of the form $\mathsf{H}(x, \boldsymbol{a}, i, c_i, z_i, D[j], \mathsf{op}_j)$. Now, during extraction, we can recognize inputs with correct data blocks. Second, a prover can rejection-sample the index j and cherry-pick only the oracle queries that correspond to the data they know, thereby excluding a significant fraction of the data from being extracted. For example, the prover could decide to

make hash queries only where the index j is below $N/3$, thus preventing 2/3 of the data blocks from ever appearing as input and being extracted.

To address the second issue, we force the prover to hash not only one, but $m > 1$ random data blocks during each attempt to solve the PoW. This makes it much more difficult for a prover to exclude a particular part of the message, as now all m blocks must fall outside the region the adversary wants to omit. To further limit the prover's influence on what to hash, we design the protocol in such a way, that every prover has to cover a large fraction of the admissible m-tuples, ensuring that a prover who attempts to rejection-sample inevitably runs out of admissible challenges. For this, we limit the range of admissible challenges to 2^d, for each i. Now, even an honest prover will only have a $1-1/e \approx 63\%$ chance of finding a hash that starts with d zeros for every $i \in [\rho]$, and will succeed in at least $\rho/2$ of them with overwhelming probability. Therefore, we only require solutions for half of the $i \in [\rho]$ for the signature to be valid. We prove a technical result, Lemma 5, capturing this intuition: choosing a larger m and limiting the fraction of tuples to be rejected ensures a large fraction of the domain will be covered. Using this lemma, we show that $m = 6$ is sufficient. Having a small value of m is important for our construction, as the signature size depends linearly on this parameter. Even though m is constant, we still cannot hash m data blocks together with their opening information at once in a single hash input, as hash functions are typically designed using iterative constructions, such as Merkle-Damgård, and this could make the construction susceptible to attacks in which different subadversaries compute different segments of the iterative pipeline. For this reason, our final construction uses an XOR of m hashes, each using one of the data blocks, i.e., $\oplus_{k\in[m]}\mathsf{H}(G(x,\boldsymbol{a}), i, c_i, z_i, D[j_k], \mathsf{op}_{j_k})$ where $j_k = \mathsf{H}(i, c_i)$. We refer to Section 5 for the details of our construction.

1.2 Distributed adversaries to model coalitions with distinct interests

Before describing the applications of beholder signatures, let us mention another important recent paper that is particularly related to our work. In [14], the authors use the distributed adversarial model to prevent collusion in secret sharing schemes by constructing a scheme that they dub *secret sharing with snitching* (SSS). To explain their idea, recall that a secret sharing scheme [35] is a protocol that allows a dealer to split a secret $\mathbf{m}$ into n shares, such that any t shares can be used to reconstruct $\mathbf{m}$, but no $t-1$ shares reveal any information about $\mathbf{m}$. Of course, if more than $t-1$ parties are corrupt and collude, then $\mathbf{m}$ is no longer secret, as the malicious parties can trivially reconstruct $\mathbf{m}$. This is a substantial limitation of secret sharing schemes, since the "illegal" reconstruction of $\mathbf{m}$ can be done covertly, without anybody being aware. Hence, standard secret sharing schemes need to rely on strong honesty assumptions, much stronger than those made in many blockchain protocols (e.g., Nakamoto consensus [30] used in Bitcoin), that work even if all the parties are malicious, as long as no large coalition of them colludes.

The authors of [14] propose a new secret sharing scheme in the distributed adversarial model, providing security even if all parties are malicious, as long as no t of them collude. The main idea behind [14] is that malicious parties that do not belong to the same "colluding coalition" can "snitch" on each other to an external "judge", who can then punish the parties who participated in the illegal reconstruction. In blockchain applications, the judge can be a smart contract. In the formal definition, the authors of [14] define a *distributed adversary*, which is a tuple $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of *subadversaries*. Each subadversary can control a certain *coalition* of shareholders. The maximal size of such a coalition is a construction parameter, but every shareholder can belong to some coalition (i.e., all the shareholders can be corrupt). The scheme remains secure as long as the subadversaries have truly separate interests and they can benefit (for example, financially) from snitching on each other to the judge. The authors of [14] also consider the possibility that the distributed adversary might use MPC to reconstruct the secret and use the slow/fast query model of [13] to model the fact that some hashes may be computed using MPC. We follow this model in our work and refer to Section 3.1 for a more thorough introduction.

1.3 Related works

Another way to capture the notion of knowledge of the message being signed is to use signatures of knowledge [7]. In a nutshell, these can be viewed as "witness signatures"—the signature analog of witness encryp-

tion—where the public key is a statement x for a hard relation $\mathcal{R}$, and the secret key is the corresponding witness w. Signatures of knowledge are essentially equivalent to simulation-extractable NIZKs [22]. As an alternative to beholder signatures, one could consider a signature of knowledge for a relation $\mathcal{R}((\mathsf{pk}, \mathsf{com}), (\mathsf{sk}, m))$:

"sk corresponds to pk and com is a commitment to m"

where, for concreteness, the signing process involves producing a NIZK for this relation. If MPC attacks are not a concern, our beholder signatures provide an efficient and direct construction of a (relaxed) signature of knowledge for this relation. Unlike [7,22], which rely on generic NIZK proof systems and require, for example, instantiating circuits for cryptographic group operations, our approach is more concrete and direct.

The relaxation is that we do not guarantee zero-knowledge with respect to the message being signed, and assume this is handled at a higher layer. Moreover, the constructions of [7,22] are in the CRS setting and use trapdoor-based extraction, while our construction is in the ROM and is the first to provide straight-line extraction. Finally, unlike prior works, our construction provides security against distributed adversaries under MPC-hardness assumptions. Straight-line extraction is crucial for this kind of security, as the cryptographic extractor guaranteed by prior works is only a thought experiment and does not deter malicious behavior in a distributed adversarial setting. Formally, this is the case because it is not possible to rewind another subadversary.

2 Applications

In this section, we discuss several applications of beholder signatures. We do it in a relatively informal way, not aiming to provide a detailed analysis of these applications. We leave this as an open problem for future work. We focus on applications in the realm of blockchains and distributed systems, but we believe that beholder signatures can be used in other contexts as well.

Data availability. *Data availability* (DA) is a concept that has gained significant attention in the context of blockchain systems, particularly in relation to sharding and scalability. The main goal of DA is to ensure that some data M is available to all network participants. While lots of important work has been done in this area, most of it focuses on so-called *data-availability sampling* [1,31,24,23,28,15], which is a technique that allows nodes to verify the availability of data without downloading the entire dataset. In a nutshell, the idea here is that nodes can randomly sample small portions of the data to check for availability, and if the sampled data is available, it is likely that the entire dataset also is. Another key concept in this area is *data availability committees* [36], where the focus is on the economic incentives for nodes to provide data availability. Beholder signatures offer an attractive solution, as they can be used to ensure that a given node has actually downloaded the data.

To be more concrete, consider a data publisher D who wants to distribute some data M to a committee of parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. In case everybody is honest, the publisher D simply sends the data M to all the parties in the committee, who sign it and publish the signatures (which can be mediated by D). The data M is considered "publicly available" if at least some fixed threshold t of parties returned valid signatures on M. Therefore, a set of such t signatures forms a data availability certificate for M. Now, consider a scenario where the parties can be malicious and incentive-driven. If the parties use a standard signature scheme for signing D, then a malicious publisher D can bribe the parties in the committee to sign M without sending them the data. To discourage the committee members from signing data they have not seen, one option is to use a verification game, where external verifiers challenge the committee members to provide the data they signed (see [36] for more details). However, this approach is interactive and requires the committee members to put some collateral that can be confiscated if they fail to provide the data. Beholder signatures allow us to avoid these issues, as they ensure that a party can only sign data they have seen. Therefore, if the parties in the committee use beholder signatures for signing M, then a malicious publisher D cannot bribe them to sign M without sending them the data. Note that once a large number of committee members indeed saw M, it essentially becomes public. To see why, suppose some receiver R wants to obtain M. The receiver R can

simply contact the commifttee members and ask them to send M, possibly offering them a reward for their cooperation. Since sending data is cheap, the committee members will be put in a Bertrand competition scenario (see, e.g., [37]), driving the price down to essentially zero.

Data dissemination. A similar scenario to data availability is data dissemination. We have already seen that a data provider D might want to disseminate M across multiple clients $\mathsf{P}_1, \ldots, \mathsf{P}_n$. If the data has size N and the data provider wants to be sure that their data actually reached all the clients, the naive way to do so is to send the data directly to each client, which incurs a communication complexity of Nn for DP. In a sparse network, it might actually be impossible to communicate directly, and one option is to use a broadcast channel. However, in the asynchronous setting, it is known that broadcast is impossible for $t \geq n/3$ corruptions [11]. In the case of such failure, the data provider might not even be able to identify whether the data reached the recipients. Our beholder signatures also provide an alternative solution to the problem of data dissemination. Then, the provider might only send the data to a handful of nodes $\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_m}$ for $m \ll n$, and ask them to disseminate the data further in the network. Then, the beholder signatures created by the nodes can also serve the dealer as a proof that the data has actually been disseminated across all $n' \gg m$ nodes. Unlike broadcast, this does *not* guarantee that *all* nodes have received the data, but in many cases it is sufficient that n' is large enough. The above would be again impossible to achieve with usual signature schemes, which often only sign the hash $H(M)$ of the data, and not the data itself, and so the dishonest network could possibly only disseminate $H(M)$ and bribe the nodes to only sign the hash. The difference is that in data availability, beholder signatures served other users of the network and not the provider.

Proofs of custody. *Proofs of custody.* (PoC) [16] is a technique proposed as a solution to enhance the security of sharded blockchains like Ethereum 2.0. In such blockchains, a set of *validators* is responsible for storing data segments, and they are required to attest that they possess these segments. The aim of [16] is to deal with the *lazy validator problem*, in which the validator blindly signs the data without even querying it, for example, to save on bandwidth or computation costs. In a nutshell, the PoC of [16] works as follows: the validator samples an ephemeral key k and computes a Merkle hash h of the data D' where $D'[i] := D[i] \,||\, k$. The data D is considered a "bomb", if h starts with d zeros for, say, $d \approx 10$. At the end of each epoch, the validator is required to reveal their ephemeral secret, and is slashed if they signed a bomb.

The construction outlined above has several downsides: first, it requires interaction at the end of each epoch, followed by a massive computation, as each signature during the epoch needs to be checked, with computation being proportional to the size of the data. Moreover, the validator might be at risk of losing their funds if they are a victim of a denial-of-service attack during the decommitment phase. Beholder signatures allow one to avoid the issues mentioned above. Since the beholder signature is overwhelmingly invalid for a different validator, it cannot be just copied, and a validator needs to recompute the signature. Moreover, the properties of beholder signatures guarantee that the validator needs to essentially download the whole data, which is precisely what proofs of custody aim to achieve. Finally, the complexity of the verification is sublinear in the size of the data. Similarly, using beholder signatures, one can construct proofs of custody for computation, as outlined by [16]. In this case, the data D is the full execution trace of the computation, and the proof of custody is a beholder signature on the hash of the execution trace. In this case, the independence of the size of the data is even more important: for applications such as machine learning, the data D might be enormous, and while proving custody will indeed be expensive (but so is the training of the model), the verification will not.

Proofs of Replication. A proof of space [12] is a proof system by which a prover convinces a verifier that they have dedicated some amount of disk space. In a useful proof of space, the prover can use that space to store some useful information, but they shouldn't be able to use less space than specified, even if this file is compressible. A more general notion is a proof of replication [19,32], proposed initially in the Filecoin whitepaper. Here, the prover can convince a verifier that they stored some k copies of a file F for them.

This redundancy should give the verifier some guarantee that the file will be available even if some of the storage provided by the prover fails. One issue with this solution is the fact that the verifier has no guarantee that the redundant copies are actually stored one different disks, so a single disk failure will not erase all of them (see, e.g., the abstract of [18], where the author states that proof of replication *fundamentally cannot guarantee that the data is stored redundantly*). Beholder signatures give us an alternative approach to ensure redundant storage of a file. The verifier can provide access to F to k provers, each associated with a public key $pk_1, \ldots, pk_k$. To prove possession of the file, each prover will occasionally send a beholder signature to the verifier. Suppose there's some collateral associated with the public keys (say, each public key must have some stake on a blockchain that can be retrieved by whoever has the secret key). In that case, we have some guarantee that the provers will store the file locally, and thus at different locations, as each prover needs access to the entire file to compute the beholder signatures. As a special case, for $k = 1$, we obtain proofs of data possesion.

3 Preliminaries

For a deterministic algorithm A, we use $y := A(x)$ to denote its output. For a probabilistic algorithm A, we write $y := A(x; r)$ to denote its output with random coins r. We write $y \leftarrow A(x)$ when the random coins r are implicitly assumed to be uniformly random. We use $=$ to assign variables. We use log to denote the logarithm base 2.

3.1 Distributed adversarial model and multi-party random oracles

We consider a *distributed* adversarial model [13,14], in which protocols are attacked by a *distributed* adversary, which is a tuple $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of randomized polynomial-time machines called *subadversaries* together with the *attack strategy* Σ. During the execution of a protocol, the subadversaries can communicate freely, and so Σ is an arbitrary multi-round interactive protocol executed between $\mathcal{A}_1, \ldots, \mathcal{A}_a$. In particular, the adversary could bootstrap a secure computation protocol of its choice, for example by letting one subadversary send a garbled circuit to another one. Using MPC in a malicious way, the adversary could clearly circumvent any guarantees we hope to achieve. To restrict the malicious use of MPC by the adversary, we follow the line of research [13,14,6] which observes that hash functions tend to be *moderately hard* to compute in MPC, that is, given a limited timeframe, it is infeasible to compute a *massive* number of hashes using MPC. Formally, we adopt the multi-party random oracle model of [14], that is, we allow the honest parties and the adversary to evaluate a random hash function $\mathsf{H}\colon \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$ through the *multi-party random oracle* $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$, which accepts queries of two kinds, called *fast* and *slow*, which model *native* and *MPC* evaluation of the hash function, respectively. Formally, $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ internally simulates a standard random oracle Ω_{H}. Upon a query of the form (fast, x), $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ responds with $\mathsf{H}(x)$. Moreover, to handle slow queries, $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ waits for all subadversaries to send $(\mathsf{slow}, \mathsf{qid}, \varphi, \psi, x_i)$. Once it has collected the queries from all subadversaries for a given qid, if all φ and ψ match, the oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ performs deterministic preprocessing φ on these inputs and queries Ω_{H} on the result, obtaining $y = \mathsf{H}(\varphi(x_{\mathsf{D}}, x_1, \ldots, x_n))$. The postprocessed output $\psi(y)$ of this computation is returned to all subadversaries. The security we achieve in this model is a form of fine-grained security against malicious use of MPC. Since we assume that a massive computation of hashes using MPC is infeasible, we consider a *slow query bound* s, which is a fixed polynomial in κ and is known to all parties, including the adversary. The number of fast queries is unbounded, with the only restriction being that it remains polynomial; in particular, it may depend on the slow query bound s. However, such massive computation is only realistic given a limited timeframe, upon which the slow query quota implicitly depends. Therefore, the execution of a protocol is divided into two epochs: the *precomputation* epoch and the *online* epoch, which happen one after another. The total number of slow queries answered in the online epoch is at most s, and any queries that exceed this quota are answered with $\perp$. On the other hand, the number of slow queries during the precomputation epoch is bounded only by the computing time of the adversary (i.e., it is an arbitrary polynomial in κ that might, again, depend on s). This corresponds to the fact that adversaries can have an adaptively long "precomputation period" before the protocol starts, while

the MPC execution time is restricted only when the protocol execution takes place. Note that this form of fine-grained security is the best we can hope for. For example, if the adversary uses MPC to simulate the honest signing algorithm—by having one party input the secret key and another input the data—it can violate the guarantees we are interested in. In principle, the adversary can always do this if it is allowed to adapt its running time to the protocol parameters; there is always a larger polynomial. This is why, when working in the Distributed Adversarial Model, instead of bounding the adversary's running time, one takes the converse approach and adapts the protocol parameters to the MPC query bound.

3.2 Identification schemes

In this subsection we collect the full formal definitions, notation, and assumptions for the identification schemes used in this work. We recall the definitions of [27,20]:

Definition 1 (Canonical Identification Scheme). *A canonical identification scheme* ID *is a tuple of algorithms* $\mathsf{ID} := (\mathsf{KGen}, \mathsf{P}, \mathsf{V}, \mathsf{GetPk})$.

- *The key generation algorithm* KGen *takes system parameters* par *as input and returns the keypair* $(\mathsf{pk}, \mathsf{sk})$.
- *The prover algorithm* $\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2)$ *is composed of two phases.* P_1 *takes as input the secret key* sk *and returns the first message (usually a commitment)* a *and a state* st*;* P_2 *takes as input the secret key* sk*, the first message* a*, a challenge* c*, and the state* st *and returns a response* z.
- *The verifier algorithm* V *takes the public key* pk *and the protocol transcript as input and deterministically accepts or rejects.*
- *The algorithm* GetPk *takes as input the secret key* sk *and outputs the corresponding public key* pk.

We require correctness: that for all valid keypairs $(\mathsf{pk}, \mathsf{sk})$*, all* $(a, \mathsf{st}) \in \mathsf{P}_1(\mathsf{sk})$*, all challenges* c *and all* $z \in \mathsf{P}_2(\mathsf{sk}, a, c, \mathsf{st})$*, we have* $\mathsf{V}(\mathsf{pk}, a, c, z) = \mathsf{rej}$.

Our definitions differs from the one of [27] in that we assume the challenge set to be of the form $\{0, \ldots, M\}$ for some value M, and we include the algorithm GetPk. We will also require standard properties:

Definition 2 (Special soundness). *An identification scheme is said to be special sound if there exists a polynomial time extractor* E *that extracts the secret key* sk *from any pair of valid transcripts* (a, c, z) *and* (a, c', z') *such that* $c \neq c'$.

Definition 3 (Special Honest Verifier Zero Knowledge). *An identification scheme is said to be SHVZK (Special Honest Verifier Zero Knowledge) if there exists a simulator* Sim *that, given a public key* pk *and a challenge* c *outputs a transcript* $(a, c, z) \leftarrow \mathsf{Sim}(\mathsf{pk}, c)$ *which is indistinguishable from the real transcript.*

Fischlin transform [20] only applies to sigma-protocols with computationally unique responses, which our construction also requires two other properties.

Definition 4 (Computationally unique responses). *An identification scheme has computationally unique responses if for every polynomial time adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c} \mathsf{V}(\mathsf{pk}, a, c, z) = \mathsf{V}(\mathsf{pk}, a, c, z'), \\ z \neq z' \end{array} \;\middle|\; (\mathsf{pk}, a, c, z, z') \leftarrow \mathcal{A}(1^\kappa) \right] \leq \mathsf{negl}(\kappa).$$

Definition 5 (Commitment min-entropy). *For every secret key* sk*, the min-entropy of the first message is superlogarithmic in* κ*, i.e.,* $\mathrm{H}_\infty(a_i) = \omega(\log \kappa)$ *where* a_i *is the random variable defined as* $(a_i, \_) \leftarrow \mathsf{P}_1(\mathsf{sk}; \cdot)$.

3.3 Commitment Schemes

We employ a binding, deterministic vector commitment scheme with local openings—formalized as $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Com.Setup}(1^\kappa)$: takes as input a security parameter 1^κ, and outputs public parameters $\mathsf{pp} \in \{0,1\}^*$.
- $\mathsf{com} := \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{m})$: takes as input a public parameter pp and a vector $\mathbf{m} = (m[1], \ldots, m[|\mathbf{m}|])$. It outputs a commitment $\mathsf{com} \in \{0,1\}^*$.
- $(\mathbf{m}[i], \mathsf{op}) := \mathsf{Com.Open}(\mathsf{pp}, \mathbf{m}, i)$: outputs the i-th block of $\mathbf{m}$ and the *opening* op.
- $b := \mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, \tilde{m}, \mathsf{op}, i)$: outputs $b = \mathsf{rej}$ if it accepts ($b = \mathsf{acc}$ if it rejects) the opening claiming the i-th block of committed in com equals $\tilde{m}$.

We require perfect correctness, position binding, and constant-size openings. We also rely on a *non-standard* assumption: *computationally unique openings*—it is infeasible for any PPT adversary, given a commitment, to produce two distinct valid opening strings (e.g., decommitment randomness/proofs) for the same committed message (and position), except with negligible probability; see **??** for complete definitions.

3.4 Erasure codes

An (n, k) erasure coding scheme over an alphabet Σ is a pair of algorithms $\mathsf{EC} = (\mathsf{Enc}, \mathsf{Dec})$, where $k > n$, $\mathsf{Enc}\colon \Sigma^n \to \Sigma^k$ and $\mathsf{Dec}\colon \Sigma^n \to \Sigma^n$. The Enc algorithm takes as input a data block, consisting of n data chunks, and outputs $k > n$ coded chunks. The Dec algorithm takes as input any n-size subset of the coded chunks and outputs the original data m data fragments. In other words, if $[c_1, \ldots, c_k] \leftarrow \mathsf{Enc}(m)$, then $\mathsf{Dec}(c_{i_1}, \ldots, c_{i_n}) = m$ for any distinct indices $i_1, \ldots, i_n \in [k]$. The rate of the erasure code is $\gamma = n/k$.

4 Definitions

A *beholder signature scheme* is a quadruple of poly-time algorithms $\mathsf{BSig} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$. It uses a deterministic commitment scheme com as building block.

- $\mathsf{pp} \leftarrow \mathsf{BSig.Setup}(1^\kappa)$: takes as input a security parameter 1^κ, it outputs *public parameters* $\mathsf{pp} \in \{0,1\}^*$. We assume that those public parameters can also be used for com.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KGen}(\mathsf{pp})$: outputs a (secret key, public key) pair $(\mathsf{sk}, \mathsf{pk})$.
- $\sigma \leftarrow \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn})$: takes as input pp, a secret key sk, a message $\mathbf{m}$ and some beacon value $\mathsf{bcn} \in \{0,1\}^\kappa$. The output is a signature σ.
- $b := \mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn})$: outputs $b = \mathsf{rej}$ if it accepts ($b = \mathsf{acc}$ if it rejects) the signature σ for (the message committed by) com under public-key pk.

We require the following properties:

Overwhelming Correctness. The scheme is *correct* if for all messages $\mathbf{m} \in \{0,1\}^*$ and all $\mathsf{bcn} \in \{0,1\}^\kappa$ it holds that

$$\Pr\left[\mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn}) = \mathsf{rej} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{BSig.Setup}(1^\kappa) \\ \mathsf{com} := \\ \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{m}) \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KGen}(\mathsf{pp}) \\ \sigma \leftarrow \\ \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn}) \end{array}\right] \geq 1 - \mathsf{negl}(\kappa),$$

where we implicitly assume that $\mathsf{BSig.Verify}$ outputs acc if $\sigma = \perp$.

Existential Unforgeability under Chosen Message (and Beacon) Attack. Our unforgeability definition is similar to the standard definition of existential unforgeability under chosen message attacks for standard signature schemes. We need to adapt it, as a beholder signature is created with respect to the *message*, but it is verified with respect to a *commitment* to the message and not the message itself. Moreover, there's a beacon as extra input. In the definition below, the adversary can ask for signatures for any message and beacon of its choice, and they break security if they come up with any (commitment, beacon, signature) triple for which they did not already receive the signature from the signing oracle. Note that we do *not* require the adversary to know the message for the commitment in the forgery. We say that the scheme is *existentially unforgeable under chosen-message attacks (EUF-CMA)*, if for every (monolithic) PPT adversary $\mathcal{A}$ with access to a signing oracle (defined below) $\mathcal{O}_{\mathsf{Sign}}(\mathsf{pp}, \mathsf{sk}, \cdot)$ we have that:

$$\Pr\left[\begin{array}{c} \mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn}) = \mathsf{rej} \\ \text{and } (\mathsf{com}, \mathsf{bcn}) \text{ is fresh (cf. below)} \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{BSig.Setup}(1^\kappa) \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KGen}(\mathsf{pp}) \\ (\mathsf{com}, \sigma, \mathsf{bcn}) \leftarrow \\ \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(\mathsf{pp}, \mathsf{sk}, \cdot)}(1^\kappa, \mathsf{pp}, \mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\kappa)$$

where the oracle is defined as follows:

$$\begin{array}{l} \underline{\mathcal{O}_{\mathsf{Sign}}(\mathsf{pp}, \mathsf{sk}, (\mathbf{m}, \mathsf{bcn}))} \\ \sigma \leftarrow \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn}) \\ \textbf{return } \sigma \end{array}$$

In the definition above $(\mathsf{com}, \mathsf{bcn})$ is *fresh* if $\mathcal{A}$ did not query the oracle on input $(\mathbf{m}, \mathsf{bcn})$ where $\mathsf{com} = \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{m})$.

Fractional Individual Extractability. The key property of beholder signatures is that a valid signature σ for a recent beacon bcn, implies that some party that participated in signing knows the corresponding secret key sk and some fraction of the message $\mathbf{m} = (m_1, \ldots, m_N)$ committed in $c = \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{m})$. Consider a distributed adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ with access to the Multi-Party Random Oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$. For brevity, we will write $\mathcal{A}^{\mathsf{H}}$ instead of $\mathcal{A}^{\Omega_{\mathsf{H}}^{\mathsf{MPC}}}$. In the preprocessing phase, $\mathcal{A}$ obtains the public parameters pp and can perform arbitrary polynomial time computation, including an *unbounded* number of slow queries to $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$, and finally outputs a state st. In the signing phase, $\mathcal{A}$ gets the state st and a random beacon bcn, and must output any valid commitment/signature pair $(\mathsf{pk}, \mathsf{com}, \sigma)$, i.e., $\mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, c, \sigma, \mathsf{bcn}) = \mathsf{rej}$. Note that the public key pk for the signature may be adversarially chosen. During this phase, $\mathcal{A}$ can make any unbounded polynomial number of *fast* oracle queries, but is limited to some number s of *slow* queries. Let $\mathcal{T}_u$ denote the fast hash transcript of the subadversary $\mathcal{A}_u$ and N be the length (in blocks) of the message.

We say that BSig is (s, a, ℓ, γ)-extractable if there exists an extractor Ext such that for any $\mathcal{A}$ as above (consisting of a subadversaries making at most q_s slow queries and forging a message of length N), with overwhelming probability, 1. at least one of the subadversaries $\mathcal{A}_u$ extracts the secret key sk corresponding to pk; 2. the subadversaries who extract the secret key sk collectively know at least a γ fraction of the committed message D. As a consequence, if the subadversaries do not trust each other and are unwilling to share the secret keys, this implies that the unique party who knows the secret key sk also knows a γ fraction of the committed message D.

$$\Pr\left[\begin{array}{c} \exists P \subset [a] \colon \forall i \in P \colon \\ (\mathsf{sk}'_i, I_i, D_i, O_i) \leftarrow \mathsf{Ext}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathcal{T}_i) \\ (\mathsf{pk}, \mathsf{sk}'_i) \in \mathcal{R}, \\ \forall j \in I_i \colon \mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, D[j], O[j], j) = \mathsf{rej}, \\ \left|\bigcup_{i \in P} I_i\right| \geq \gamma \cdot N \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{BSig.Setup}(1^\kappa) \\ \mathsf{st} \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{pp}) \\ \mathsf{bcn} \leftarrow \{0,1\}^\kappa \\ (\mathsf{pk}, \mathsf{com}, \sigma) \leftarrow \mathcal{A}^{\mathsf{H}}(1^\kappa, \mathsf{pp}, \mathsf{bcn}, \mathsf{st}) \\ \mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn}) = \mathsf{rej} \end{array}\right] \geq 1 - \mathsf{negl}(\kappa).$$

If $\gamma = 1$, the scheme is called *fully individually extractable.*

4.1 Remarks on the definition

Fully individually extractable signatures. Given a γ-fractionally individually extractable scheme, we can obtain a fully individually extractable scheme by first using erasure coding to encode the underlying data. This is implicitly assumed for applications.

Beacon value. The beacon value bcn ensures the freshness of the signatures, as nothing prevents the adversary from deleting the data D *after* producing the signature and the beacon value ensures that the adversary has *recently* known the data, which is crucial for applications like proofs of data possession. Since in our model the adversary is allowed to make an unbounded number of slow queries in the preprocessing phase, the beacon must crucially be unpredictable prior to the signing phase; otherwise the adversary could simply precompute the signature entirely within MPC. Looking ahead, by using bcn for domain separation, i.e., including it to every oracle query, we can assume that, with overwhelming probability, all the queries made in the preprocessing phase are useless after the beacon is revealed.

Unforgeability. Standard notions of existential unforgeability for signature schemes (EUF-CMA) assume that the adversary outputs the *message* $\mathbf{m}$ together with the forgery. In our case, however, the signature is only verified with respect to the commitment. Therefore, we put a stronger requirement and demand that the adversary cannot even provide a valid commitment com, possibly without knowing the underlying message $\mathbf{m}$.

Message Length. In the definition above, we assumed that the message consists of N blocks. In practice, N will depend on the data being signed, but the parameters of the construction and the running times will depend on N. Moreover, our goal is to obtain signatures that are succinct, i.e., $|\sigma| = \mathsf{poly}(\kappa, \log N)$. We refer to Sections 6 and 7.3 for the discussion on how the length of the message influences the parameters of the scheme.

Straight-line Extraction. Our extractor is a straight-line extractor, as it takes a single transcript as input and does not rewind the adversary. In particular, any of the subadversary could actually eavesdrop on the queries sent to the random oracle and use the extractor to *actually* extract the secret key and the data; this would be impossible in case of a rewinding extractor. This means that, should the data or the secret key be unknown to all the subadversaries, prior to signature production, the signing process will essentially cause it to be leaked to at least one of them. Straight-line extractors are necessary while proving security against quantum adversaries or in a composable setting. Cryptography in the distributed adversarial model is a new setting where straight-line extraction is necessary, here to provide credible threats to distributed adversaries trying to attack the protocol using techniques such as MPCs.

5 Construction

Let H be a random oracle, and let H_d denote the truncation of H to its first d bits. Let G be a collision-resistant hash function. In this section, we present a construction that integrates two distinct proofs: one attesting to the knowledge of a secret key sk, and another confirming the possession of data D. Our objective is to ensure that the prover P *simultaneously* and *individually* knows both values. In other words, our construction is a specific instance of Proof of Individual Knowledge [13], in which the secret key of the prover remains hidden to the verifier, while the leakage of D is not a concern. This way, the generation of the proof cannot be outsourced.

One can trivially prove the knowledge of the data D by sending it in whole to the verifier, but we consider the setting in which the data is too large for this to be feasible. In other words, we are interested in a *succinct* proof, i.e., such that the communication complexity is $\mathsf{poly}(\kappa, \log|D|)$; looking ahead, our signature size will actually be independent of $|D|$. Moreover, since we do not want the verifier to have to fetch the entire data for verification, the signature verification should only use the commitment com to the data, and not the full data itself. Our methodology builds upon a proof-of-work paradigm analogous to that employed in the

Fischlin transform [20], which will enable the prover to send much shorter messages. In the standard Fischlin transform, the procedure involves finding values c_i such that: 1. the hash $\mathsf{H}(x, \boldsymbol{a}, c_i, z_i)$ begins with d zeros, 2. and, the tuples (a_i, c_i, z_i) form accepting transcripts.

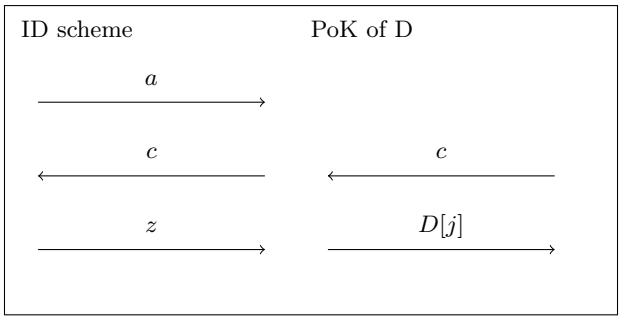

Fig. 1. Schematic starting point for our protocol, $j = \mathsf{H}(c)$.

As a starting point to our protocol, consider the protocol depicted in Figure 1, which can be thought of as a parallel composition of two sigma protocols: the underlying identification scheme and a 2-round protocol for proving knowledge of the data D by answering with a randomly chosen chunk $D[j]$ for $j = \mathsf{H}(c)$. Naively applying the Fischlin transform would lead to computing hashes of the form $\mathsf{H}(x, \boldsymbol{a}, c, z, D[j])$ for $j = H(c)$. However, while this gives us a signature scheme, it does not provide the extractability guarantees that we require from a *beholder* signature. For a concrete attack, consider a dishonest prover who only knows a fraction of the data, say, who knows $D[j]$ for $j \in J'$ where $|J'| = |D|/3$. Since now the proof is *non-interactive* and involves trying many possible values of c, such a prover could simply discard all the values of c such that $H(c) \notin J'$ and still succeed in computing the proof. In order to prevent such attacks, we could require the prover to *simultaneously* derive many indices $(j_1, \ldots, j_m)$ from $H(i, c)$, for an appropriate choice of m (cf. Section 6), and hash *all* of them, i.e., check whether: $\mathsf{H}_d(x, \boldsymbol{a}, c, z, i, k, D[j_1], \ldots, D[j_k]) = 0$.

Note that, at this point, nothing guarantees that the data $D[j]$ being hashed is indeed the one corresponding to the data being signed, making it non-trivial to even verify the signature without accessing the full data. Moreover, a malicious prover could try to confuse the extractor by hashing unrelated data and thereby prevent extraction. To address these two problems, we also include the opening of $D[j]$ in the hash, computed with respect to the commitment com. This leads to a proof-of-work-like check of the form: $\mathsf{H}_d(x, \boldsymbol{a}, c, z, i, k, D[j_1], \ldots, D[j_m], \mathsf{op}_{j_1}, \ldots, \mathsf{op}_{j_m}) = 0$.

Unfortunately, this construction would still not be secure against distributed adversaries. In the distributed adversarial model, one cannot assume the hash function to be a random oracle on *long* inputs; instead, all that one can assume is that the underlying compression function is a random oracle, and can be treated as an *individual* unit of computation. This is because hash functions are typically built using iterative constructions, such as the Merkle-Damgård or sponge. In our case, one possible attack would be to let one subadversary compute the prefix of the hash *not* containing the data chunks $D[j]$, and let the other subadversary complete the Merkle-Damgård. Therefore, to ensure that the input to the hash fits into a single compression function, we introduce two final changes: instead of including x and $\boldsymbol{a}$ verbatim in the hash, we only include their digest $\mathsf{prelude} = G(x, \boldsymbol{a})$ where G is a collision-resistant hash.[6] Furthermore, instead of concatenating the data chunks and their openings, we do it one-by-one, and xor the results, leading to the final proof-of-work-like check: $\bigoplus_{k \in [m]} \mathsf{H}_d(\mathsf{prelude}, c, z, i, k, D[j_k], \mathsf{op}_{j_k}) = 0^d$. This leads to a compression function of the form $\mathsf{H}\colon \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$, which is achievable in our model by a linear change of the security parameter. At a high level, since the responses z and the openings op_{j_k} are unique, all the prover can do is to grind through many values of c. Consequently, a large *constant* fraction of the data D must be hashed and, for some index i, use two different challenges $c \neq c'$, revealing the secret key sk through the special

[6] Note that this hash function does not need to be "MPC-hard".

soundness property of the underlying ID scheme. While the rejection sampling attack is unavoidable and the construction described above only guarantees the extraction of a *constant* fraction of data D, this is easy to fix by using an erasure code, as described in Section 4.1.

The final algorithm. The pseudocode for the final algorithm is given in Figure 2. We will explain the construction using Schnorr's protocol as an example. We start by generating the openings $\mathsf{op}_1, \ldots, \mathsf{op}_N$ for all the chunks of data $D[j]$.[7] Just as in the Fischlin transform, we compute *all* ρ prover's first messages: $(a_i, \mathsf{st}_i) := \mathsf{P}_1(\mathsf{sk}; r_i) = (g^{r_i}, r_i)$ and look for the values c_i satisfying the following:

1. z is the prover's final message for the challenge c_i, i.e., $z := \mathsf{P}_2(\mathsf{sk}, a_i, c_i, r_i) = r_i + c_i\mathsf{sk}$;
2. $(j_{i,1}, \ldots, j_{i,m}) = \mathsf{H}(i, c_i)$ are the indices of the chunks of data $D[j_{i,k}]$ that are going to be queried;
3. the "proof-of-work" check is satisfied, i.e., if $\oplus_{k\in[m]}\mathsf{H}_d(\mathsf{prelude}, c_i, z, i, k, D[j_{i,k}], \mathsf{op}_{j_{i,k}}) = 0$.

Above, $\mathsf{prelude}$ is the common prefix for all queries, i.e., $\mathsf{prelude} = G(x, \boldsymbol{a})$ and $x = (\mathsf{pk}, \mathsf{bcn}, \mathsf{com})$. Finally, for the iterations that succeed, we output the Schnorr transcripts (a_i, c_i, z_i) and the data chunks queried for the indices derived by $\mathsf{H}(i, c_i)$ together with their openings. For those iterations that fail, we only output a_i and set all the other values to $\perp$. The verification algorithm straightforwardly checks the correctness of the Schnorr triples, i.e., $\mathsf{V}(\mathsf{pk}, a_i, c_i, z_i)$ asserts that $g^{z_i} = a_i\mathsf{pk}^{c_i}$, it verifies the data together with the openings, and checks the "proof-of-work".

Remark. In the protocol above we have used Schnorr's protocol to simplify the exposition. However, one can replace it with any other specially-sound identification scheme, such as EdDSA. One can also instantiate beholder signatures using lattice primitives, such as ML-DSA (Crystals Dilithium) [9], or isogeny-based ones, such as CSI-FiSh [4].

6 Security analysis

Before stating our main security theorem, let us discuss the parameters that we use, and how they affect security and efficiency of our scheme . We also summarize the range of parameters that are necessary for the security proof to go through.

- κ: the computational security parameter; we assume that the random oracle has output length κ, with $\kappa = 256$ or $\kappa = 512$ being a typical choice.
- a: the number of (mutually distrustful) subadversaries. In practice this will be a small constant, say $a < 100$.
- N: the length of the encoded message to be signed, in blocks of κ bits. We assume $N \geq \kappa$, otherwise one can add redundancy using erasure coding. In practice, N will be much larger, say $N = 4096$.
- s: the number of slow queries the distributed adversary is allowed while signing. This parameter depends on the validity period during which the signature is considered fresh (see the discussion in Section 4.1). This parameter is a fixed polynomial in κ, as per Section 3.1.
- $\gamma < 1$: the extraction rate, which matches the rate of the erasure code that should be used to guarantee full extractability. A typical choice is $\gamma = 1/2$ which leads to $\log(1/\gamma) = 1$.
- ρ, d, m: the parameters that can be freely chosen: the number of Fischlin iterations, the difficulty for the proof of work, and the number of data samples per proof of work, respectively. These parameters directly affect the signature size (which is on the order of ρm) and the signing time (which is on the order of $\rho 2^d m$). We require that:
  - $2^{d-4}\rho \geq 2N$, to ensure that the (honest) signer always covers the whole message with high probability,

[7] Note that this step, which constitutes a significant part of the time required to compute a beholder signature (cf. Section 7.3), can be trustlessly outsourced and reused by various honest parties. Also, see Section 7.2 for a discussion how to achieve this efficiently.

$\mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, D, \mathsf{bcn})$

1 : set $c_i = \perp$ for $i \in [\rho]$
2 : $(D[i], op_i) \coloneqq \mathsf{Com.Open}(\mathsf{pp}, D, i)$ for $i \in [N]$
3 : **for** $i = 1, \ldots, \rho$ **do**
4 : $\quad (a_i, \mathsf{st}_i) \leftarrow \mathsf{P}_1(\mathsf{sk})$
5 : **endfor**
6 : $\boldsymbol{a} := (a_1, ..., a_\rho)$
7 : $\mathsf{pk} := PK(\mathsf{sk})$
8 : $\mathsf{com} := \mathsf{Com.Commit}(D)$
9 : $\mathsf{prelude} = \mathsf{H}((\mathsf{pk}, \mathsf{bcn}, \mathsf{com}), \boldsymbol{a})$
10 : **for** $i = 1, \ldots, \rho$ **do**
11 : $\quad$ **for** $c = 0, \ldots, 2^d - 1$ **do**
12 : $\quad\quad z := \mathsf{P}_2(\mathsf{sk}, a_i, c, \mathsf{st}_i)$
13 : $\quad\quad (j_{i,1}, ..., j_{i,m}) := \mathsf{H}(i, c)$
14 : $\quad\quad$ **if** $\oplus_{k\in[m]}\, \mathsf{H}_d(\mathsf{prelude}, c, z, i, k, D[j_{i,k}], \mathsf{op}_{j_{i,k}}) = 0^d$:
15 : $\quad\quad\quad c_i := c$
16 : $\quad\quad\quad z_i := z$
17 : $\quad\quad\quad$ **break**
18 : $\quad$ **endfor**
19 : **endfor**
20 : **if** $|\{i\colon c_i \neq \perp\}| < \rho/2$ **then return** $\perp$ $\quad$ // failed, restart with fresh randomness
21 : $\mathbf{op}' := \{\mathsf{op}_{j_{1,1}}, \ldots, \mathsf{op}_{j_{\rho,m}}\}$ $\quad$ // $\mathsf{op}_{j_{i,k}} = \perp$ if $c_i = \perp$
22 : $D' := \{D[j_{i,1}], \ldots, D[j_{i,m}]\}_{i\in \boldsymbol{f}}$ $\quad$ // $D[j_{i,k}] = \perp$ if $c_i = \perp$
23 : **return** $\sigma = (\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{z}, \mathbf{op}', D', \mathsf{bcn})$

$\mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn})$

1 : parse σ as $(\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{z}, \mathbf{op}', D', \mathsf{bcn})$
2 : corr $\coloneqq 0$
3 : **for** $i = 1, \ldots, \rho$ **do**
4 : $\quad\quad (j_{i,1}, ..., j_{i,m}) := \mathsf{H}(f, c_i)$
5 : $\quad\quad$ **if** $\mathsf{V}(\mathsf{pk}, a_i, c_i, z_i) = \mathsf{rej}$
6 : $\quad\quad\quad\quad \wedge\ \forall k \in [m]$: **assert** $\mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, D[j_{i,k}], \mathsf{op}_{j_{i,k}}, j_{i,k})$
7 : $\quad\quad\quad\quad \wedge \oplus_{k\in[m]} \mathsf{H}_d(\mathsf{prelude}, c_i, z_i, i, k, D[j_{i,k}], \mathsf{op}_{j_{i,k}}) = 0^d$
8 : $\quad\quad$ **then** corr $\coloneqq$ corr $+ 1$
9 : $\quad$ **endfor**
10 : **endfor**
11 : **return** corr $\geq \rho/2$

Fig. 2. Pseudocode for the Fischlin-transform-based beholder signature built from an arbitrary ID scheme. In the motivating case of Schnorr's protocol, $\mathsf{P}_1(\mathsf{sk}; r) = (g^r, r)$, $\mathsf{P}_2(\mathsf{sk}, a_i, c_i, \mathsf{st}_i) = \mathsf{st}_i + c_i\mathsf{sk}$; and $\mathsf{V}(\mathsf{pk}, a_i, c_i, z_i)$ checks that $g^{z_i} = a_i \mathsf{pk}^{c_i}$.

- $2^{d-4}\rho \geq s/m + \rho a$, to ensure that the signing process requires more hashes than the adversary's budget,
- $m \geq 6/\log(1/\gamma)$, to ensure that the adversary cannot rejection sample.

If these parameters are chosen as the minimal possible values, then the signing process requires hashing only a small constant times more data than the length of the signed message N. This is optimal, as we cannot hope to extract more than what we hash, and in particular the honest signer must compute $\Omega(N)$ hashes to sign a message of length N. However, one might want to choose larger parameters to increase security against malicious use of MPC. Finally, the soundness error is exponentially small in $\rho \cdot d$ (like in Fischlin's scheme), extraction can fail with probability exponential in only ρ. However, as our analysis shows, one can lower the signature size at the cost of increasing *both* d and m.

6.1 Main Theorem

In the remainder of this section we will prove our main theorem:

Theorem 1. *Suppose that the commitment scheme satisfies the properties of Section 3.3, the ID scheme satisfies the properties of ?? and the hash function* H *is modeled as a random oracle. Then, for every distributed adversary* $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$*, any slow query bound* s*, and an arbitrary message length* N*, one can choose the parameters* ρ, d, m *such that the construction* BSig *of Section 5 is a beholder signature satisfying correctness, fractional individual extractability and unforgeability.*

We prove correctness, individual extractability and soundness in the three subsections below, and the theorem is a direct consequence of Propositions 1 to 3.

6.2 Correctness

Proposition 1. *The signature scheme* BSig *is overwhelmingly correct.*

Proof. If the signing algorithm from Figure 2 outputs $\sigma = \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn}) \neq \perp$, then by construction, verification will accept, i.e., $\mathsf{BSig.Verify}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathsf{bcn}) = 1$. For this, the signer must succeed in line 14 for at least $\rho/2$ of the ρ possible i's. The probability of succeeding for any particular i is $1 - (1 - 2^{-d})^{2^d} \geq 1 - 1/e \approx 0.63$ (the probability of 2^d i.i.d. Boolean variables, each being 1 with prob. 2^{-d}, all turning out 0 converges to $1/e$ as d grows). Let X be the number of i's for which we succeed, in expectation it's $\mu = \rho \cdot 0.63$. Using a Chernoff bound, the probability of succeeding in less than $\rho/2$ cases is exponentially small in ρ. The exact probability is not so relevant here, as one can simply restart the signing process with fresh randomness in the rare cases where one fails. □

6.3 Fractional Individual Extractability

We now prove the following:

Proposition 2. *Under the assumptions of Theorem 1, the beholder signature scheme is* γ*-fractionally extractable against any distributed adversary consisting of at most* a *subadversaries, each making at most* s *slow queries.*

The extractor. As required by the definition, we define the extractor in Figure 3. The extractor takes as input a beholder signature σ that verifies under pk for (the committed) message com. The extractor tries to recover the secret key sk as well as some data D and its opening information ops based on the transcript $\mathcal{Q}$ of hash queries made during the signing process by a subadversary $\mathcal{A}_u$. In this section we will show that from the transcripts (containing the fast queries to H_d) of those subadversaries from which Ext outputs $\mathsf{sk} \neq \perp$, we can, w.h.p., also extract at least a γ fraction of the entire data.

$\mathsf{Ext}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathcal{Q})$:

1. Initialize $\mathsf{sk} := \bot, D := (\bot, \bot, \ldots, \bot), \mathsf{ops} := (\bot, \bot, \ldots, \bot)$ where $|D| = |\mathsf{ops}| = N$
2. Parse $\sigma = (\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{z}, \mathbf{op}, D', \mathsf{bcn})$.
3. Parse all H_d queries as $(p, c, z, i, k, w, \mathsf{op})$, cf. Figure 2.
4. For each $q = (p, c, z, i, k, w, \mathsf{op})$, let $j := H(i, c)[k]$. Check if:
   1. $1 \le k \le m$;
   2. $p = G(x, \boldsymbol{a})$ where $x = (\mathsf{pk}, \mathsf{com}, \mathsf{bcn})$;
   3. $\mathsf{V}(a_i, c, z) = \mathsf{rej}$ for some $i \in [\rho]$;
   4. the opening verifies: $\mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, d, \mathsf{op}, j) = \mathsf{rej}$.
5. Discard any elements that have failed the checks and let $\mathcal{Q}'$ be the set of the remaining elements. Let $\mathcal{Q}_i \subset \mathcal{Q}'$ be the set of queries for which $\tilde{a} = a_i$.
6. If $|\mathcal{Q}_i| > 1$ for some value of i, take any two $q_1, q_2 \in \mathcal{Q}_i$ such that $c_1 \neq c_2$ and set $\mathsf{sk} := \mathsf{Ext}_{\mathsf{ID}}(q_1, q_2)$.
7. For $q \in \mathcal{Q}'$, set $D[j] := d, \mathsf{ops}[j] = \mathsf{op}$ for $j = H(i, c)[k]$.
8. Output $(\mathsf{sk}, D, \mathsf{ops})$

Fig. 3. The extractor Ext for the beholder signature scheme as discussed in §6.3. $\mathsf{Ext}_{\mathsf{ID}}$ is the extractor for the identification scheme, which is guaranteed to exist by its special soundness.

Lemma 1. *The extraction is overwhelmingly correct, i.e., except with negligible probability, if* $(\mathsf{sk}^*, D^*, \mathsf{ops}) := \mathsf{Ext}(\mathsf{pk}, \mathsf{com}, \sigma, \mathcal{Q})$*, then* $\mathsf{sk} = \mathsf{sk}^*$ *and for all indices* j *it holds that either* $\mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, D^*[j], \mathsf{ops}[j], j) = \mathsf{rej}$ *or* $D^*[j] = \bot$.

Proof. The overwhelming correctness of the secret key is guaranteed by the special soundness of the ID scheme, step 4 trivially ensures the correctness of the data. □

Let $\mathcal{A}$ be an s-bounded distributed adversary who produces a signature σ. Let $\mathcal{T}$ be the full hash transcript, let $\mathcal{T}_{\mathsf{fast}}$ be the fast hash transcript and let $\mathcal{T}_u$ be the fast hash transcript of $\mathcal{A}_u$. We will call a query $q = (p, c, z, i, k, w, \mathsf{op})$ *useful* if it passes the checks in step 4 of the extractor. The intuition is that any query will not help the adversary to produce a valid signature. We will call a pair (i, c) an *attempt* if $\mathcal{T}$ contains a *useful* query of the form $q = (p, c, z, i, k, w, \mathsf{op})$. In the following lemma, we show that the adversary can produce at most m useful queries per attempt, which intuitively follows from the fact that the only value the adversary can mine over (for a fixed i) is the challenge c.

Claim 1. *Let* σ *be a valid beholder signature produced by* $\mathcal{A}$*. Useful queries* $q = (p, c, z, i, k, w, \mathsf{op})$ *in* $\mathcal{T}$ *made by the adversary* $\mathcal{A}$ *with overwhelming probability satisfy the following properties:*

1. *the value* z *is (computationally) uniquely given* p, i, c*;*
2. w, op *are (computationally) unique given* i, c, k*.*

In particular, the attempt (i, c) *admits at most* m *useful queries.*

Proof. If any of the properties above were violated, one could use the (a monolithic adversary induced by the) adversary $\mathcal{A}$ to build an adversary breaking either (1) the computational uniqueness of responses of the ID scheme (cf. **??**), or (2a) the position binding or (2b) the uniqueness of openings of the commitment scheme (cf. Section 3.3). Then the claim about m useful queries follows from pigeonholing. □

The following lemma demonstrates a concentration bound: that, up to an arbitrary constant factor, the adversary must make approximately the expected number of hash queries. The choice of this factor is somewhat arbitrary, but it directly affects the selection of m and d. Different approximation factors reflect tradeoffs between signature size and proving time. Our choice is motivated by the concrete results in Sections 7.3 and 7.4 and our goal of achieving compact signature sizes. In particular, keeping ρ low is essential for minimizing the concrete signature size.

Lemma 2. *With overwhelming probability,* $|\mathcal{T}| \ge 2^{d-4}\rho m$ *and* $|\mathcal{T}_{\mathsf{fast}}| \ge 2^{d-4}\rho m - s$.

Proof. Call an attempt (i, c) *complete* if $\mathcal{T}$ contains *exactly* m useful queries for that attempt. Let A be the set of all attempts, and let $|A| = T$. For an attempt $(i, c) =: \alpha \in A$ let $q_\alpha^1, \ldots, q_\alpha^m$ be the m associated elements of the hash transcript, and let Q_α be the random variable equal to 1 if the attempt passes the proof-of-work of line 14 (that is, if $\oplus_{k\in[m]} \mathsf{H}_d(q_\alpha^k) = 0^d$) and 0 otherwise. Intuitively, $Q_\alpha = 1$ if the attempt was successful. Clearly, $\mathbb{E}[Q_i] = 2^{-d}$. Let $\mu = T/2^d$. Suppose, to the contrary, that $T < 2^{d-4}\rho$ and so $8\mu < \rho/2$. Note that in order to produce a valid signature, it is necessary that $\sum_{\alpha\in A} Q_\alpha \geq \rho/2$. However, by the Chernoff Bound:[8]

$$\Pr\left[\sum_{\alpha\in A} Q_\alpha \geq \frac{\rho}{2}\right] \leq \Pr\left[\sum_{\alpha\in A} Q_\alpha \geq 8\mu\right] \overset{\text{C.B.}}{\leq} \exp\left(-\frac{49\mu}{3}\right) \leq \exp\left(-\frac{49\rho}{48}\right)$$

which is negligible as long as $\rho = \omega(\log\lambda)$. Therefore, $T \geq 2^{d-4}\rho$ and $|\mathcal{T}| \geq 2^{d-4}\rho m$ by Claim 1. □

We say that sk was *leaked* to $\mathcal{A}_u$ if $\mathcal{T}_u$ contains at least two Schnorr transcripts of the form $(a, c, z), (a, \tilde{c}, \tilde{z})$, and so $\mathcal{A}_u$ can recover the secret key sk thanks to the special soundness of the protocol. Finally, let $N_{\mathsf{leaked}} = |\{u \mid \mathsf{sk} \text{ was leaked to } \mathcal{A}_u\}|$.

Lemma 3. *Suppose that $a < 2^{d-4} - s/(\rho m)$. Then w.o.p. $\mathcal{T}_{\mathsf{fast}}$ contains $\geq a+1$ distinct ID scheme transcripts, and so $N_{\mathsf{leaked}} \geq 1$.*

Proof. As proved Lemma 2, $|\mathcal{T}_{\mathsf{fast}}| \geq 2^{d-4}\rho m - s$. Therefore, there exists an i for which $\mathcal{A}$ made at least $2^{d-4}m - s/\rho$ queries. By Claim 1, the adversary $\mathcal{A}$ can make at most m useful queries per attempt (i, c), and so must have queried at least $2^{d-4} - s/(\rho m) > a$ distinct values of c, and so $\mathcal{A}_u$ can recover the secret key sk using the special-soundness extractor. □

We will use the following lemma which states that one cannot compress a random variable below its entropy. We only state if for the special case of the uniform distribution over some set $\mathcal{X}$ (a proof of this lemma for the case where $\mathcal{X}$ is $\{0,1\}^n$ can be found in [10])

Lemma 4. *For a set $\mathcal{X}$ let $n = \log(|\mathcal{X}|)$. For any randomized encoding procedure $\mathsf{Enc} : \{0,1\}^r \times \mathcal{X} \to \{0,1\}^e$ and decoding procedure $\mathsf{Dec} : \{0,1\}^r \times \{0,1\}^e \to \{0,1\}^n$ where*

$$\Pr_{x\leftarrow\$\mathcal{X}, r\leftarrow\$\{0,1\}^r}[\mathsf{Dec}(r, \mathsf{Enc}(r, x)) = x] \geq \delta$$

we have $e \geq n - \log(1/\delta)$

We have already shown that the distributed adversary who produces a beholder signature σ must leak the secret key to at least one of the subadversaries. We now aim to show that any such distributed adversary must also make oracle queries spanning a large (at least γ) fraction of the data, of the same form as defined in line 14 of Figure 2. The difficulty lies in the fact that $\mathcal{A}$, after computing the indices as in line 13, can decide to not query on those indices. In particular, $\mathcal{A}$ can first compute m-tuples $X_1, \ldots, X_p$ of indices with $p = \mathsf{poly}(\kappa)$, and only query the hash oracle on a $\varepsilon = 1/\mathsf{poly}(\kappa)$ fraction, i.e., only query X_i for $i \in I$ where $|I| \geq p\varepsilon$. This fraction cannot be too small, as to find the required $\rho/2$ proofs of work one needs to query $2^{d-1}\rho$ of the m tuples in expectation, and with overwhelming probability at least $2^{d-4}\rho$ (as analyzed by Lemma 2), and so we can assume $\varepsilon p \geq 2^{d-4}\rho$. The adversary wins if the set Z of indices covered by the X_i for $i \in I$ is small, i.e., constitutes less than a γ fraction of the whole set, and the extractor cannot recover the required fraction of the data.

The following lemma shows that we can choose the parameters so that the adversary cannot avoid querying a large fraction of the data, and is explained visually in Figure 4. In the lemma, we will think of the X_i as the m-tuples of indices as computed in line 13.

[8] $\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\frac{\mu\delta^2}{3}\right)$ with $\delta = 7$.

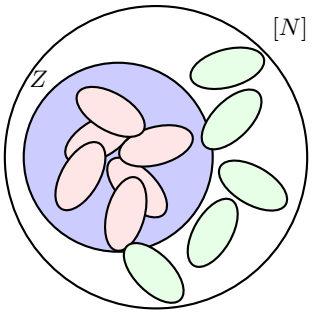

Fig. 4. A depiction of Lemma 5: the ellipses are the sets being sampled, and the red of them fall into the set Z of relative size γ (X_i for $i \in I$), while the green ones are discarded by the adversary.

Lemma 5. *Consider a sequence $X = (X_1, \ldots, X_p)$ of subsets $X_i \subset [N], |X_i| = m$ sampled uniformly at random.*

For $\varepsilon, \gamma \in (0,1)$, the probability (over the choice of X) that some ε subset of X is covered by a γ-dense subset of $[N]$ is at most

$$\begin{aligned} &\Pr[\exists\ Z \subset [N], |Z| \le \gamma N, I \subset [p], |I| \ge p \cdot \varepsilon \text{ s.t. } \forall i \in I \colon X_i \subset Z] \le 2^{-z} \\ &\text{where } z \ge \varepsilon p \cdot (m \log(1/\gamma) - \log(1/\varepsilon)) - N \underbrace{\gamma \log(1/\gamma)}_{\le 1} \end{aligned} \tag{1}$$

Proof. The proof is a compression argument. For a random X, assume Z, I as in the statement of the lemma exist. We will show that, in this case, X can be compressed by z bits (for z as in the statement of the lemma) by using the fact that for every $i \in I$ the elements X_i are contained in Z (and not all of $[N]$). The lemma then directly follows from Lemma 4.

The encoding $\mathsf{Enc}(X)$ outputs an (information theoretically optimal) encoding of the following sets:

1. The indices $I = \{i \ : \ X_i \in Z\}$ ($|I| = \varepsilon \cdot p$)
2. $Z \subset [N]$ ($|Z| = \gamma \cdot N$)
3. An encoding of the $\{X_i\}_{i \in I}$ (using the fact that $X_i \in Z$)
4. An encoding of the $\{X_i\}_{i \notin I}$

Since X is sampled from a uniform distribution over a set of size $p \cdot \binom{N}{m}$, it can be encoded as a uniform string of length $n = \log\left(p \cdot \binom{N}{m}\right)$.

In our encoding, instead of encoding each X_i directly, which requires $\log \binom{N}{m}$ bits per X_i, we first encode I, Z, and then, for the $X_i, i \in I$, use an encoding that only uses $\log \binom{\gamma N}{m}$ bits. This encoding requries

$$\underbrace{(1-\varepsilon)p \log \binom{N}{m}}_{\{X_i\}_{i \notin I}} + \underbrace{\varepsilon \cdot p \log \binom{\gamma \cdot N}{m}}_{\{X_i\}_{i \in I}} + \underbrace{\log \binom{p}{\varepsilon p}}_{I} + \underbrace{\log \binom{N}{\gamma \cdot N}}_{Z} \le$$

$$p \log \binom{N}{m} - \varepsilon \cdot p \cdot m \log(1/\gamma) + \varepsilon p \log(1/\varepsilon) + \gamma \cdot N \cdot \log(1/\gamma)$$

Note that this is $\varepsilon p m \log(1/\gamma) - \varepsilon p \log(1/\varepsilon) - \gamma N \log(1/\gamma)$ bits shorter than the encoding required to encode X, the lemma now follows from Lemma 4. □

We are now ready to prove individual extractability. For this, consider a distributed adversary $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ who computes a beholder signature σ making s slow queries. We will now argue that from the

fast queries made by the N_{leaked} subadversaries who know the secret key we can w.h.p. extract at least a γ fraction of the data D. We also assume $N \geq \kappa$, otherwise the message can be artificially inflated with erasure coding.

Lemma 6. *Let $P \subset [a]$ be the set of those subadversaries who extract the secret key, which is non-empty by Lemma 3 and let $J = \bigcup_{u \in P} I_u$ be the set of indices of D extracted by those adversaries who know the secret key. Suppose that $N \geq \kappa$, $m \geq 6/\log(1/\gamma)$,*

$$2^{d-4}\rho \geq s/m + \rho \cdot a \tag{2}$$

and

$$2^{d-4}\rho \geq 2N \tag{3}$$

Then, w.o.p., $|J| \geq \gamma N$.

Proof. In Lemma 2 we have shown that, with overwhelming probability, the adversary must make at least $p_{\min} = 2^{d-4}\rho$ attempts such proofs to find a signature. On the other hand, by construction (i.e., the bound t) it can make at most $p_{\max} = 2^d \cdot \rho$ attempts. Each attempt requires m hash queries, $\mathcal{A}$ can compute at most s/m attempts using slow queries, and the $(a - N_{\mathsf{leaked}})$ subadversaries who do not learn the secret key can make at most ρ attempts each, we get that the adversary can make at most $s/m + \rho a$ attempts "for free". If we choose d such that

$$s/m + \rho a \leq 2^{d-4}\rho \tag{4}$$

then the "free attempts" constitute at most half of the total queries, and at least $p_{\text{fast}} := p_{\min}/2 = 2^{d-5}\rho$ attempts must be completed using fast queries made by subadversaries who know the secret key. Suppose that the adversary made p attempts, and let $\varepsilon := p_{\text{fast}}/p \geq p_{\text{fast}}/p_{\max} =: \varepsilon_{\min}$ Observe that $\varepsilon_{\min} = (p_{\min}/2)/p_{\max} = 1/32$ and so $\log(1/\varepsilon_{\min}) = 5$. If we set

$$m \geq \frac{6}{\log(1/\gamma)} \tag{5}$$

we can ensure that $m\log(1/\gamma) - \log(1/\varepsilon_{\min}) \geq 1$. Finally, we use Lemma 5, which ensures that p_{fast} queries out of $p_{\max}$ contain a γ fraction of the data, except with probability 2^{-w}. Choosing d and ρ such that $2^{d-4}\rho \geq 2N$ ensures that:

$$\begin{aligned} w &\overset{\text{Lemma 5}}{\geq} \varepsilon p_{\max} \cdot (m\log(1/\gamma) - \log(1/\varepsilon)) - N \\ &= p_{\text{fast}} \cdot (m\log(1/\gamma) - \log(1/\varepsilon)) - N \\ &\overset{(3)}{\geq} 2N \cdot (m\log(1/\gamma) - \log(1/\varepsilon_{\min})) - N \\ &\overset{(5)}{\geq} 2N - N = N \geq \kappa \end{aligned}$$

and so $2^{-z} \leq 2^{-\kappa} = \mathsf{negl}(\kappa)$. Therefore, $|J| \geq \gamma N$, as desired. □

This concludes the proof of Proposition 2. Combining Lemmas 3 and 6 also implies the following corollary:

Corollary 1. *If only one of the subadversaries learns the secret key, then w.o.p. the extractor outputs both the secret key sk and a γ fraction of the data D for this subadversary. In particular, this holds if the adversary $\mathcal{A}$ is monolithic.*

6.4 Unforgeability

Proposition 3. *Assume that the beholder signature scheme* BSig *is built with respect to a specially sound identification scheme for a hard relation* $\mathcal{R}$*. Moreover, assume that* $\rho = \omega(\log \kappa)$*. Then the scheme is EUF-CMA secure (cf. Section 4).*

Proof. At a high level, we follow the proof strategy of [33] for Schnorr signatures, where the (knowledge) soundness of the underlying NIZK implies *no-message-attack* security, in which the adversary does not have access to the signing oracle. Then, the zero-knowledge property of the underlying proof system allows simulating the signing oracle, and lifting the security to the desired *chosen-message-attack* security. In our case, one can apply the (fractional) individual extractability to a monolithic adversary to essentially obtain knowledge soundness (cf. Corollary 1), and, the zero-knowledge property of the underlying ID scheme is enough to simulate the signing oracle, as the oracle is given the message as its input. We graphically depict the reduction in Section 6.4 and proceed with the formal proof. We now describe the reduction $\mathcal{A}_{\mathcal{R}}$ to the hardness of the relation $\mathcal{R}$. The reduction stores all the hash queries made by $\mathcal{B}_{\mathsf{BSig}}$ in the set $\mathcal{Q}$. It also simulates the signing oracle: upon a query $\mathcal{O}_{\mathsf{Sign}}(D, \mathsf{bcn})$, the reduction $\mathcal{A}_{\mathcal{R}}$ does the following for each $i \in [\rho]$:

1. simulates which mining attempts fail and which succeed by sampling independent random function $\tau_i : \{0,1\}^d \to \{0,1\}^d$; crucially, the domain and the codomain are both of polynomial size;
2. selects c_i as the smallest c with $\tau_i(c) = 0$;
3. if $c_i = \bot$, then $(a_i, c_i, z_i) \coloneqq (a_i, \bot, \bot)$, otherwise it simulates the transcript $(a_i, c_i, z_i) \leftarrow \mathsf{Sim}(\mathsf{pk}, c_i)$,
4. computes the openings $\mathbf{op}$

Afterward, it *programs* H_d on the inputs

$$q_{im} = (G(x, \boldsymbol{a}),\, c_i,\, z_i,\, i,\, k,\, D[j_{i,k}],\, \mathsf{op}_{j_{i,k}}) \qquad (k \in [m], i \in [\rho]), \tag{6}$$

choosing them randomly conditioned on $\bigoplus_{k\in[m]} \mathsf{H}_d(q_{im}) = 0$. Finally, it outputs the signature $(\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{z}, \mathbf{op}, D', \mathsf{bcn})$. All other hash queries are answered honestly. Finally, the reduction uses the extractor Ext to extract the secret key sk, which is where we need $\rho = \omega(\log \kappa)$ (cf. Corollary 1).

On the required min-entropy of the commitments. Note that for this programming procedure to work, we crucially we require the simulated commitments a_i to have superlogarithmic min-entropy (cf. Definition 5). To see why, notice that if the adversary manages, for some i, to query the random oracle H_d on the m inputs described by (6) *before* the reduction actually programs it, then the programming could be prevented. This is described by the following bad event:

$$\mathsf{Inconsistent} \;=\; \big\{\, \exists\, i \in [\rho] \colon \forall\, k \in [m] :\; \mathcal{B}_{\mathsf{BSig}} \text{ queried } q_{im} \text{ before step (v)} \,\big\}.$$

However, since the a_i are freshly sampled at each signing query and have high min-entropy, the probability that the adversary manages to produce such a query on its own is negligible. In fact, it easy to see that, for an adversary who makes at most Q random oracle queries, it holds that $\Pr[\mathsf{Inconsistent}] \le Q \cdot \rho \cdot 2^{-\mathrm{H}_\infty(a_i)} + \mathsf{negl}(\kappa)$. Consequently, the programming is undetectable and the proof of the reduction is complete. □

7 Efficiency analysis

Our construction of beholder signatures is efficient in terms of both the signature size and the concrete performance of signing. In fact, beholder signatures are feasible to be computed on a commodity laptop. In this section, we provide a detailed analysis of our proof-of-concept implementation.

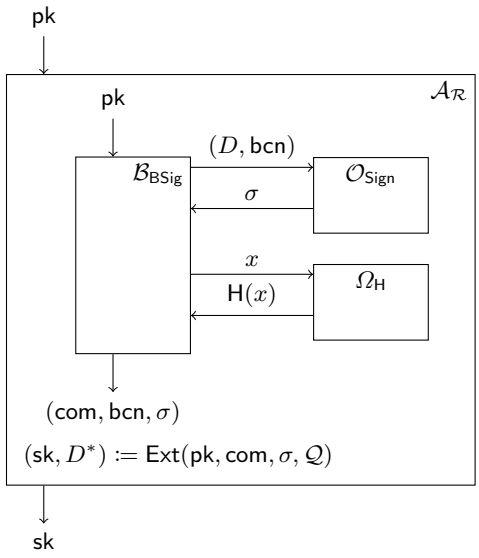

Fig. 5. A schematic depiction of the reduction $\mathcal{A}_{\mathcal{R}}$ to the hardness of the relation $\mathcal{R}$, proving the unforgeability of the beholder signature scheme BSig.

7.1 Implementation

We provide a proof-of-concept implementation of beholder signatures using Rust, which is available at https://github.com/marmistrz/beholders.

We use KZG commitment scheme [25] to instantiate the vector commitment scheme and, in particular, we use its multithreaded implementation, rust-kzg. The KZG commitments and the Schnorr protocol use the BLS12-381 curve, and we use SHA-512 compression function to instantiate the random oracle H. We furthermore used rayon to parallelize the computation of the "mining" phase of computing the beholder signature.

7.2 Computing the openings

Since the complexity of computing a KZG opening for a polynomial of degree d is $O(d)$, naively computing the openings in line 2 involves $O(N^2)$ operations, which would dominate the complexity of the prover. The issue can be avoided by using the technique of [17], which reduces the complexity to $O(N \log N)$. We use this approach in our proof-of-concept implementation.

Our implementation generates its own trusted setup rather than relying on one produced via a secure multi-party computation ceremony. Unfortunately, the trusted setup established by the Ethereum community in their ceremony only provides 4096 elements. We intentionally generate the trusted setup to enable experiments with very large datasets, which exceed the sizes that can be proved with the Ethereum trusted setup.

7.3 Experimental Results

Our experiments were conducted on a Dell Latitude 7440 equipped with an Intel i7-1365U CPU. For our evaluation, we follow the parameters mandated by Section 6. Specifically, we set $m = 6$ and $\rho = 10$, which guarantees extractability except with probability $\approx 3.6 \times 10^5$. The difficulty d is set as the minimal admissible value, i.e., $d = \lceil 5 + \log N - \log \rho \rceil$. Producing a beholder signature involves two main computationally intensive steps: (1) computing the openings using FK20 and (2) performing the mining process. If signatures

File Size [KiB]	FK20 Time [s]	Mining Time [s]	Freshness period
16	0.408757	0.289098	16h 3min
32	0.940463	0.649596	1d 12h
64	2.415401	1.490786	3d 10h
128	4.678211	4.523394	10d 11h
256	11.506188	6.847156	15d 20h
512	19.904092	18.735283	43d 8h
1024	48.823722	35.007809	81d 52min
2048	100.106238	67.015559	155d 3h

Table 1. Benchmarks of our implementation for different data sizes; averages over 50 samples. The parameters were: $m = 6$, $\rho = 10$, $d = \lceil 5 + \log N - \log \rho \rceil$, where N is size in bytes divided by 32. Freshness period is mining time multiplied by 2×10^5.

are required within a short timeframe, the former step can be performed as an *offline* phase that could be preprocessed and reused across multiple signatures or even (trustlessly!) between different signers. The latter step, however, is inherently *online* and must be performed anew for each data, signer, and beacon combination. For this reason, we report the timings for these two components separately in Table 1. Note that these are *minimal times*; one might wish to choose a higher difficulty d to increase security against malicious use of MPC.

MPC-resistance. Based on the current state-of-the-art in MPC implementations of SHA2 [29], we estimate that the adversary requires ≈ 100 ms to evaluate a single SHA-512 hash in a LAN setting, with ≈ 11 MB of communication. At these rates, the adversary's bottleneck would be communication, allowing for little speed-up due to parallelism. Comparing this to the realistic CPU hashing rate of ≈ 500 ns per hash on our machine, this corresponds to a slowdown factor on the order of $\approx 2 \times 10^5$. This highlights a trade-off between the time allocated to mining (honest protocol execution) and the *freshness* window (the post-beacon interval during which a signature remains valid). In our target applications, selecting a validity window between hours and months is practical.

7.4 Signature size

Our beholder signatures are also concretely small. In order to optimize for size, we assume that the signer provides proof for *exactly* half of the Fischlin iterations. In other words, for half of the Fischlin iterations, the signature only contais the commitment a_i, which is a group element (48B for BLS12-381). For the other half, we include (i) the commitment a_i (48B), (ii) the challenge c_i represented as a 32-bit integer (4B), (iii) the response z_i, which is represented as a field element (32B), (iv) the m data samples, each represented as a field element (32B), and (v) the m openings, each represented as a group element (48B). Table 2 shows their sizes, with $m = 6$ and different values of ρ.

ρ	**10**	**13**	**16**	**19**	**22**	**25**	**28**	**31**	**34**
Size [B]	3,060	4,284	4,896	6,120	6,732	7,956	8,568	9,792	10,404

Table 2. Size of the signature for different values ρ and $m = 4$, for a Schnorr protocol over the BLS12-381 curve.

References

1. Al-Bassam, M., Sonnino, A., Buterin, V., Khoffi, I.: Fraud and data availability proofs: Detecting invalid blocks in light clients. In: Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25. pp. 279–298. Springer (2021)
2. 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
3. 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
4. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. Cryptology ePrint Archive, Paper 2019/498 (2019), https://eprint.iacr.org/2019/498
5. 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
6. Bormet, J., Dziembowski, S., Faust, S., Lizurej, T., Mielniczuk, M.: Secret sharing with snitching against insured colluders. IACR Cryptol. ePrint Arch. (2024)
7. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Annual International Cryptology Conference. pp. 78–96. Springer (2006)
8. Damgård, I.: On σ-protocols. https://www.cs.au.dk/~ivan/Sigma.pdf (2010), lecture notes for CPT 2010, v.2
9. Dang, T., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., et al.: Module-lattice-based digital signature standard (2024)
10. De, A., Trevisan, L., Tulsiani, M.: Time space tradeoffs for attacks against one-way functions and prgs. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 649–665. Springer (2010). https://doi.org/10.1007/978-3-642-14623-7_35, https://doi.org/10.1007/978-3-642-14623-7_35
11. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
12. 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
13. Dziembowski, S., Faust, S., Lizurej, T.: Individual cryptography. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 547–579. Springer (2023). https://doi.org/10.1007/978-3-031-38545-2_18, https://doi.org/10.1007/978-3-031-38545-2_18
14. Dziembowski, S., Faust, S., Lizurej, T., Mielniczuk, M.: Secret sharing with snitching. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 840–853. ACM (2024). https://doi.org/10.1145/3658644.3690296
15. Evans, A., Mohnblatt, N., Angeris, G.: ZODA: Zero-overhead data availability. Cryptology ePrint Archive, Paper 2025/034 (2025), https://eprint.iacr.org/2025/034
16. Feist, D.: Proofs of custody. https://dankradfeist.de/ethereum/2021/09/30/proofs-of-custody.html (2021), (Accessed on 30/09/2021)
17. Feist, D., Khovratovich, D.: Fast amortized kzg proofs. Cryptology ePrint Archive (2023)
18. Fisch, B.: Poreps: Proofs of space on useful data. IACR Cryptol. ePrint Arch. p. 678 (2018), https://eprint.iacr.org/2018/678
19. Fisch, B.: Tight proofs of space and replication. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 324–348. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_12, https://doi.org/10.1007/978-3-030-17656-3_12
20. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Annual International Cryptology Conference. pp. 152–168. Springer (2005)

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. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Annual International Cryptology Conference. pp. 581–612. Springer (2017)
23. Hall-Andersen, M., Simkin, M., Wagner, B.: Foundations of data availability sampling. IACR Commun. Cryptol. **1**(4), 34 (2024). https://doi.org/10.62056/A09QUDHDJ
24. Hall-Andersen, M., Simkin, M., Wagner, B.: FRIDA: Data availability sampling from FRI. Cryptology ePrint Archive, Paper 2024/248 (2024), https://eprint.iacr.org/2024/248
25. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 177–194. Springer (2010)
26. Kelkar, M., Babel, K., Daian, P., Austgen, J., Buterin, V., Juels, A.: Complete knowledge: Preventing encumbrance of cryptographic secrets. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 2415–2429. ACM (2024). https://doi.org/10.1145/3658644.3690273
27. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. Cryptology ePrint Archive, Paper 2016/191 (2016), https://eprint.iacr.org/2016/191
28. Król, M., Ascigil, O., Rene, S., Rivière, E., Pigaglio, M., Peeroo, K., Stankovic, V., Sadre, R., Lange, F.: Data availability sampling in ethereum: Analysis of P2P networking requirements. CoRR **abs/2306.11456** (2023). https://doi.org/10.48550/ARXIV.2306.11456
29. Liu, Y., Lai, J., Yang, P., Wang, Q., Yang, A., Yiu, S., Weng, J.: Highly efficient actively secure two-party computation with one-bit advantage bound. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025. pp. 2846–2864. IEEE (2025). https://doi.org/10.1109/SP61157.2025.00183, https://doi.org/10.1109/SP61157.2025.00183
30. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
31. Nikolaenko, V., Boneh, D.: Data availability sampling and danksharding: An overview and a proposal for improvements (2023), https://a16zcrypto.com/posts/article/an-overview-of-danksharding-and-a-proposal-for-improvement-of-das/
32. Pietrzak, K.: Proofs of catalytic space. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA. LIPIcs, vol. 124, pp. 59:1–59:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPICS.ITCS.2019.59, https://doi.org/10.4230/LIPIcs.ITCS.2019.59
33. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**, 361–396 (2000)
34. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
35. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176
36. Tas, E.N., Boneh, D.: Cryptoeconomic security for data availability committees. In: International Conference on Financial Cryptography and Data Security. pp. 310–326. Springer (2023)
37. Wikipedia contributors: Bertrand competition — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bertrand_competition&oldid=1315494968 (2025), [Online; accessed 10-October-2025]
38. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164. IEEE Computer Society (1982). https://doi.org/10.1109/SFCS.1982.38