Beholder Signatures

Stefan Dziembowski[1,2], Sebastian Faust[3], Paweł Kedzior[1,2], Marcin Mielniczuk[1], Susil Kumar Mohanty[1,5], and Krzysztof Pietrzak[4]

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

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 build on Fischlin's transformation of a sigma protocol to a non-interactive, 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 space. In particular, we describe how to use them to construct *proofs of custody* (Feist, 2021) that do not require ephemeral keys and are non-interactive. We also outline applications to *data dissemination*, *data availability*, and *proofs of replication*.

1 Introduction

The notion of *knowledge* is a fundamental concept in cryptography that has been studied for several decades, starting from the seminal work on *proofs of knowledge* [23,2], 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 [36] 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 to bind the message to be signed to the proof.

A related notion is the one of succinct non-interactive arguments of knowledge (SNARKs) [5], which allows proving knowledge of a value with a short proof, even if the value itself is very long (often there is a succinct commitment, and the SNARK proves knowledge of the long committed message). Recently, it has been observed in [14,28] 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 [14,28] 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 [39,3], with one party, say P_1, handling communication with the verifier. In this scenario, the verifier will accept the proof, 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, [14,28] 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 is 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 [14] 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 a bounded, 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 [14] 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 [14] 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 [14] 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 [14] 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 [28], termed *proofs of complete knowledge*, below, when we sketch the construction of our new primitive. Importantly, the scheme of [28] works only 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 contributions

We introduce a new primitive, called *beholder signatures*[6], 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 means that the only way to produce a signature on a message $\mathbf{m}$ is to have the entire $\mathbf{m}$ stored on a single machine, or possibly distributed across several machines, but with the secret key stored on all of them. Since sharing the secret key is usually the last thing a user would want to do, one can think of the parties which are willing to share their secret keys with each other, as a single entity that is storing the message $\mathbf{m}$. Formally, this is captured by the distributed adversarial model (see Sect. 3.1), where the adversary is modelled as a set of mutually distrusting "subadversaries". More precisely, the main contributions of this paper are as follows:

[6] 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.

1. **Conceptualization of beholder signatures**: we introduce and formally define this new primitive.
2. **Construction of beholder signatures**: we provide a concrete construction of beholder signatures based on the Fischlin transform (see Sect. 1.2).
3. **Security analysis**: we prove that our construction is secure in the individual cryptography model.
4. **Practical applications**: we identify several key applications of beholder signatures within the blockchain ecosystem, such as proofs of custody, data availability, and proofs of replication.
5. **Implementation and Evaluation**: we provide a concrete implementation of beholder signatures in Rust. We also provide benchmarks that demonstrate that our scheme is concretely efficient and feasible for execution on commodity hardware.

In the next section, we give a more detailed overview of the notion of beholder signatures and our construction. We also discuss the applications of beholder signatures in more detail in Sect. 2.

1.2 Our new primitive: beholder signatures

Let us now give a more detailed overview of the notion of beholder signatures. 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 to 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 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 Sect. 2 for an example of such an application). Our construction in this case is the same, and one can relax the security parameters in this case, cf. Sect. 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. Sect. 1.3.

An overview of our construction. Our starting point is a sigma protocol [10], 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 *honest verifier 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 [21] 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, the value $\mathsf{H}(x, \boldsymbol{a}, i, c_i, z_i)$ 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. 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 Fig. 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, Lem. 4, 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 $m = 8$ is sufficient for extraction rate $\gamma = 1/2$ with the query budget in Sect. 6. 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, similar to the ones recently demonstrated by [9].

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 Sect. 5 for the details of our construction.

1.3 Other related work

Proofs of complete knowledge. Proofs of complete knowledge (PoCKs) from [28] provide a way to prove "individual" (called "complete" in their paper) knowledge of a witness to an NP relation. They ensure straight-line extractability using techniques similar to the ones employed in the Fischlin transform [21]. However, unlike our beholder signatures, their construction is crucially *interactive*. Furthermore, obtaining a

non-interactive variant of PoCKs is non-trivial, given that applying any standard transform such as the Fiat–Shamir or Fischlin transform does *not* preserve individual knowledge (called "forced revelation") in their paper and yields an insecure protocol. Finally, their construction can only be secure with short witnesses. Our construction overcomes these limitations and generalizes the notion of proofs of complete knowledge (PoCKs) from [28].

Cryptography in the distributed adversarial model. Several recent papers have been dedicated to the study of cryptography in the distributed adversarial model (DAM). *Secret sharing with snitching* [15] circumvents the impossibility of secret-sharing under full corruption by assuming that the adversary consists of several mutually distrustful entities and incentivizing honest behavior through fraud proofs; this result was later strengthened by [6] to provide resilience against distributed adversaries using *insurance contracts* as a countermechanism. Recent results by [29] indicate that these techniques face a challenge in the presence of mutually-assured destruction contracts.

Other work [9] has shown that one *cannot* assume that a hash function can be "MPC-hard" or "individual" when evaluated on long inputs, due to the fact that hash functions are typically designed using iterative constructions such as Merkle-Damgård or sponge constructions. In particular, they present attacks against [28] and provide mitigations. Individual cryptography has also been recently used in [8]. The main idea of this work is to make threshold decryptions auditable: each decryption produces a so-called self-incriminating proof that helps detect rogue decryptions.

Signatures of knowledge. 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 encryption—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 [24]. 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 [7,24], 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 [7,24] 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.

Non-Delegatable Commitments. Concurrent work by [22] introduces the notion of *non-delegatable commitments*, which is similar in spirit to beholder signatures. Informally, creating commitments and openings is tied to a secret key, and delegating this process necessarily compromises that key. They provide a construction, and discuss several applications of this new primitive. Their construction combines Merkle-tree commitments with verifiable secret sharing to achieve this property. This uses few public-key operations, and as a consequence their construction is more efficient in terms of compute time (as reported by the benchmarks in [22]). On the other hand, the resulting commitments and opening proofs are larger than the beholder signatures of this work. Moreover, the extraction guarantee achieved by non-delegatable commitments is formally weaker than that of beholder signatures.

2 Applications

In this section, we discuss several applications of beholder signatures. 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* [1,33,26,25,31,16], 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* [37], 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 [37] 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 committee 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., [38]), driving the price down to essentially zero.

Data dissemination. A similar scenario to data availability is data dissemination. While the former served the users of the network, the latter serves the data provider. In data dissemination, a data provider D wants to disseminate some data M across a network of parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. The goal is to ensure that M reaches all parties in the network, or at least a large fraction of them. This is particularly relevant in the context of blockchains, where a miner might want to disseminate a new block across the network, or a validator might want to disseminate some new state information. Beholder signatures can be used to ensure that the data has actually been disseminated across the network. As previously, consider a data provider D who wants 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 [12]. 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) [17] 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 [17] 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 [17] works as follows: the validator samples an ephemeral key k and computes a Merkle hash h of the data D' where $D'[i] := D[i] \mid\mid k$. The data D is considered a "bomb", if h starts with d zeros for, say, $d \approx 10$. During the epoch the validator attests to $D_1, \ldots, D_n$, and at the epoch, it is required to reveal their ephemeral secret, and is slashed if it signed a bomb.

The construction of [17] has several downsides: First, it requires interaction at the end of each epoch due to the disclosure of ephemeral keys, leading to on-chain transaction costs. Moreover, congestion poses a risk to the validator: if the disclosure transaction is submitted but not included in the blockchain within the required time frame, the validator faces financial losses. Furthermore, the protocol requires high verification complexity $O(n|D|)$; this computation needs to be done *en masse* at the end of each epoch and is difficult to amortize.

Beholder signatures avoid these issues. Instead of penalizing blind signing, beholder signatures prevent it in the first place. 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, or more precisely $O(n \log|D|)$. We believe that our construction of proofs of custody is practical enough to be included in real blockchain systems.

Similarly, using beholder signatures, one can construct proofs of custody for computation, by following the ideas of [17] and letting the data D be the full execution trace of the computation. Then, the proof of custody is a beholder signature on the execution trace. In this case, validation with respect to the commitment becomes 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 should not.

Proofs of replication. A proof of space [13] 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 [20,34], 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 [19], 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 possession.

3 Preliminaries

For a deterministic algorithm A, we write $y := A(x)$ to denote its output. For a probabilistic algorithm A, we write $y := A(x; r)$ when its random coins r are explicit, and $y \leftarrow A(x)$ when they are sampled uniformly at random. For a finite set S, the notation $y \leftarrow S$ means that y is sampled uniformly from S. We use $:=$ (or $:=$) for deterministic assignment and $=$ for equality. For a positive integer n, let $[n] := \{1, \ldots, n\}$. We use PPT to mean probabilistic polynomial time and $\mathsf{negl}(\kappa)$ for an unspecified negligible function of the security parameter κ. Expressions of the form $\Pr\left[E \;\middle|\; \mathsf{Exp}\right]$ denote the probability that event E occurs in the random experiment Exp specified to the right of the bar. We use log to denote the logarithm base 2.

3.1 Distributed adversarial model and multi-party random oracles

To model adversaries in our scheme, we use the *distributed adversarial model* of [14,15]. A *distributed adversary* is a tuple $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of PPT interactive Turing machines called *subadversaries*. One can think of the subadversaries as distinct entities that cooperate to attack the protocol while not necessarily trusting each other, and possibly interested in learning each other's secrets for their own benefit. During the execution of a protocol, the subadversaries can communicate freely. The adversary $\mathcal{A}$ communicates via one of the subadversaries, say $\mathcal{A}_1$, who can pass the received messages between the other subadversaries and the external entities. When we say that *a message m is sent to (resp. "sent by") $\mathcal{A}$*, we mean that it is sent to (resp. "sent by") $\mathcal{A}_1$. The distributed adversary $\mathcal{A}$ can employ an arbitrary strategy to attack the protocol. In particular, since the subadversaries are mutually distrusting, they can execute secure computation techniques in order to achieve a common goal (i.e., to break the security of the protocol they are attacking), without sharing their secrets. These techniques may include, for example, classical secure multi-party computation (MPC) protocols such as garbling or secret-sharing-based MPC protocols, but also advanced techniques such as fully homomorphic encryption (FHE), or indistinguishability obfuscation (iO); we use MPC as an umbrella term for all such techniques. Since—in theory—MPC protocols can be used to securely compute any function (under specific, but realistic assumptions), this means that in principle the subadversaries can emulate any (possibly randomized) polynomial-time strategy to attack the protocol. The key observation of [14,15] is that typically general-purpose MPC techniques are significantly more expensive than computations done in plain, and—for some functions—no tailored, non-general-purpose MPC protocols have been constructed. The nature of the overhead differs, being network overhead for some protocols, and computational overhead for others. Building on this insight, these works introduce a model that captures the difference between computations done using MPC and computations done in plain, assuming "MPC-hardness" of a specific function. Using this model, the authors of [14,15] design protocols that remain secure against distributed adversaries that are limited in the amount of MPCs they can perform.

More concretely, the authors of [14,15] observe that functions like SHA-256 have remained challenging to efficiently instantiate in MPC and each known technique incurs significant overhead. Therefore, they construct their protocols using a hash function H as a cryptographic building block. Since they want to avoid attacks in which the adversary exploits iterative structure of modern hash functions[7], they assume that H has a fixed-input length. In practice, this can be achieved by letting H be a hash compression function of a Merkle-Damgård hash function. For the purposes of this paper, we use a fixed-input-length function $\mathsf{H}\colon \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$, where κ is the security parameter; the input length matches the compression-function calls in our construction. This can be assumed up to a constant rescaling of the security parameter.

We model the fact that secure computation incurs significant overhead using the *multi-party random oracle model* (MPROM). In this model, the hash function H is modelled as a random oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ (containing a random function $\mathsf{H}\colon \{0,1\}^{5\kappa} \to \{0,1\}^{\kappa}$) that can be accessed in two ways: using *fast* queries and using

[7] For an example of such attacks, see [9].

slow queries. The number of fast queries is bounded only by the adversary's running time, while the number of slow queries in the online phase is limited by a parameter s (called the *slow query bound*). This models the fact that computing hashes using MPC is significantly more expensive than computing hashes in plain.

The fast queries are identical to the queries in the standard random oracle model: each subadversary can directly access a random oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ and query it on the input of its choice. In this case, the caller knows the input x and output $\mathsf{H}(x)$ of the query in its entirety. The slow queries, on the other hand, are used to model the fact that the subadversaries can compute hashes using an MPC protocol. In order to perform such evaluation, each subadversary $\mathcal{A}_i$ sends a query (qid, x_i) to the oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ (where $\mathsf{qid} \in \{0,1\}^*$ and $x_i \in \{0,1\}^{5\kappa}$). After the query for qid has been received from all the subadversaries, the oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ computes $y = \mathsf{H}(x_1 \oplus \cdots \oplus x_a)$ and returns a secret sharing of y to the subadversaries, i.e., it samples $(y_1, \ldots, y_a)$ such that $y = y_1 \oplus \cdots \oplus y_a$ and returns y_i to each $\mathcal{A}_i$.

While the above modeling may seem restrictive, it is in fact quite general. Firstly, assuming that every subadversary contributes to every MPC hash evaluation can be done without loss of generality, as any subadversary that does not want to contribute can simply send $x_i = 0^{5\kappa}$. Secondly, this model captures an arbitrary MPC protocol relative to a random oracle H: to compute $\psi(\mathsf{H}(\varphi(x)))$, the adversary can first use an MPC protocol to compute shares $x_1, \ldots, x_a$ of $\varphi(x)$, then use the oracle $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$ to obtain shares $y_1, \ldots, y_a$ of $\mathsf{H}(\varphi(x))$, and finally use another MPC protocol to compute shares of $\psi(\mathsf{H}(\varphi(x)))$. Importantly, in this model, this MPC computation does not count towards the slow query bound. The preceding argument treats the subadversaries as honest-but-curious participants in their internal MPC, but since this assumption concerns the *adversarial* behavior, it only makes the model stronger.

Similarly to the standard random oracle model, we record the oracle queries in transcripts. We define the *global hash transcript* $\mathcal{T}$ as the list of all queries sent to $\Omega_{\mathsf{H}}^{\mathsf{MPC}}$, and the *fast global hash transcript* $\mathcal{T}_{\mathsf{fast}}$ as its sublist of fast queries. For each subadversary $\mathcal{A}_i$, we define its *local fast hash transcript* $\mathcal{T}_i$ as the list of fast queries made directly by $\mathcal{A}_i$. Thus, every query in $\mathcal{T}_i$ has an input and output known in full to $\mathcal{A}_i$; slow-query shares are not included.

Of course, the bound on the number of slow queries is only realistic if the adversary has a limited time in which to evaluate hashes in MPC. In particular, if the adversary could run for an arbitrary polynomial time *before* the protocol execution starts, it could precompute an arbitrary number of slow queries. To reflect this, we divide the execution into a *preprocessing* (or precomputation) phase followed by an *online* phase. The total number of slow queries answered in the online phase is at most s, and any queries that exceed this quota are answered with $\perp$. The preprocessing phase is not subject to this bound, although the adversary remains polynomial time.

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 [30,21]:

Definition 1 (Canonical Identification Scheme). *A canonical identification scheme* ID *is a tuple of algorithms* $\mathsf{ID} := (\mathsf{KGen}, \mathsf{P}, \mathsf{V}, \mathsf{GetPk})$ *with challenge space* $\mathcal{C} = \{0, \ldots, M\}$.

- *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.

The associated key relation is $\mathcal{R} := \{(\mathsf{pk}, \mathsf{sk}) : \mathsf{GetPk}(\mathsf{sk}) = \mathsf{pk}\}$. *We require correctness: for all* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{R}$*, all* (a, st) *in the support of* $\mathsf{P}_1(\mathsf{sk})$*, all challenges* $c \in \mathcal{C}$ *and all* z *in the support of* $\mathsf{P}_2(\mathsf{sk}, a, c, \mathsf{st})$*, we have* $\mathsf{V}(\mathsf{pk}, a, c, z) = 1$.

Our definition differs from that of [30] in that the challenge space is written as $\{0,\ldots,M\}$ and the algorithm GetPk is explicit. We will also require the following standard properties:

Definition 2 (Special soundness). *An identification scheme is said to be special sound if there exists a polynomial-time extractor E that, given a public key pk and any pair of transcripts (a,c,z) and (a,c',z') accepting under pk, with the same first message and $c \neq c'$, outputs a secret key sk satisfying $(\mathsf{pk},\mathsf{sk}) \in \mathcal{R}$.*

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 [21] only applies to sigma-protocols with certain properties stated below, which our construction also requires.

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

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

Definition 5 (Commitment min-entropy). *For every secret key sk, if $(a,\mathsf{st})\leftarrow \mathsf{P}_1(\mathsf{sk})$, then the min-entropy of the first message is superlogarithmic in κ, i.e., $\mathrm{H}_\infty(a)=\omega(\log\kappa)$.*

3.3 Commitment Schemes

In this subsection, we recall the formal definitions and assumptions for the deterministic vector commitment scheme with local opening used by beholder signatures—including perfect correctness, position binding, computationally unique openings, and constant-size openings. We assume a binding and *deterministic* commitment scheme [8]. For our construction of beholder signatures we'll additionally require that the commitment scheme allows for local opening. We also require constant-size openings. The KZG commitment scheme [27] satisfies these requirements.

A message $\mathbf{m}\in\{0,1\}^*$ is a bitstring which we parse as a vector of N blocks (say of 256 bits each), $\mathbf{m}=(m[1],\ldots,m[N])$; we always assume that a message is padded to have a length that is a multiple of the block size. Thus, N denotes the number of blocks, not the bit length of $\mathbf{m}$.

A *deterministic vector commitment scheme with local opening* is given by a quadruple of polynomial-time algorithms $\mathsf{Com}=(\mathsf{Setup},\mathsf{Commit},\mathsf{Open},\mathsf{Verify})$ 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 $\mathbf{m}=(m[1],\ldots,m[N])$. 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 $b=1$ if it accepts ($b=0$ if it rejects) the claim that the value committed at position i equals $\tilde{m}$.

Perfect Correctness. For all messages $\mathbf{m}$ and indices $i\in[N]$:

$$\Pr\left[\mathsf{Com.Verify}(\mathsf{pp},\mathsf{com},\tilde{m},\mathsf{op},i)=1\;\middle|\;\begin{array}{c}\mathsf{pp}\leftarrow\mathsf{Com.Setup}(1^\kappa)\\ \mathsf{com}:=\mathsf{Com.Commit}(\mathsf{pp},\mathbf{m})\\ (\tilde{m},\mathsf{op}):=\mathsf{Com.Open}(\mathsf{pp},\mathbf{m},i)\end{array}\right]=1$$

[8] Deterministic means $\mathsf{Com.Commit}(\mathsf{pp},\mathbf{m})$ depends only on pp and $\mathbf{m}$. Note that such a commitment scheme cannot be hiding, as one can test a candidate message $\mathbf{m}$ by checking whether $\mathsf{Com.Commit}(\mathsf{pp},\mathbf{m})=\mathsf{com}$.

Position binding. The (computational) *position binding* property states that no PPT adversary $\mathcal{A}$ can open a commitment to two different values at the same index:

$$\Pr\left[\begin{array}{c}\mathsf{Com.Verify}(\mathsf{pp},\mathsf{com},m,\mathsf{op},i)=\\ \mathsf{Com.Verify}(\mathsf{pp},\mathsf{com},m',\mathsf{op}',i)=1,\\ m\neq m'\end{array}\;\middle|\;\begin{array}{c}\mathsf{pp}\leftarrow\mathsf{Setup}(1^\kappa)\\ (m,m',i,\mathsf{com},\mathsf{op},\mathsf{op}')\leftarrow\mathcal{A}(\mathsf{pp})\end{array}\right]\leq\mathsf{negl}(\kappa)$$

Computationally unique openings. We will need the property that the openings of the commitment scheme are unique. This property is satisfied by most natural schemes. We formalize it as a dual property to the position binding, that no PPT adversary $\mathcal{A}$ can produce two valid openings for a given index:

$$\Pr\left[\begin{array}{c}\mathsf{Com.Verify}(\mathsf{pp},\mathsf{com},m,\mathsf{op},i)=\\ \mathsf{Com.Verify}(\mathsf{pp},\mathsf{com},m',\mathsf{op}',i)=1,\\ \mathsf{op}\neq\mathsf{op}'\end{array}\;\middle|\;\begin{array}{c}\mathsf{pp}\leftarrow\mathsf{Setup}(1^\kappa)\\ (m,m',i,\mathsf{com},\mathsf{op},\mathsf{op}')\leftarrow\mathcal{A}(\mathsf{pp})\end{array}\right]\leq\mathsf{negl}(\kappa)$$

Constant opening size. We finally require that the opening size is independent of the number of committed blocks, i.e., $|\mathsf{op}|=O(\kappa)$.

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 $\mathsf{Dec}\colon([k]\times\Sigma)^n\to\Sigma^n$ on inputs with distinct indices. The encoder maps a message $m=(m_1,\ldots,m_n)$ to k coded chunks $(c_1,\ldots,c_k)$. The decoder takes any n indexed coded chunks and recovers the original message; formally, if $(c_1,\ldots,c_k)\coloneqq\mathsf{Enc}(m)$, then

$$\mathsf{Dec}\big((i_1,c_{i_1}),\ldots,(i_n,c_{i_n})\big)=m$$

for every choice of distinct indices $i_1,\ldots,i_n\in[k]$. The rate of the erasure code is $\gamma=n/k$.

4 Definitions

We are now ready to present the main technical definition of our work. A *beholder signature scheme* is a quadruple of polynomial-time algorithms $\mathsf{BSig}=(\mathsf{Setup},\mathsf{KGen},\mathsf{Sign},\mathsf{Verify})$. It uses a deterministic vector commitment scheme Com as a building block.

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

In the extractability experiment below, the beacon is sampled only after preprocessing and is therefore unpredictable during that phase. 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

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

where we implicitly assume that $\mathsf{BSig.Verify}$ outputs 0 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 adapt it because a beholder signature is created with respect to the *message* but verified with respect to a *commitment* to that message, and because the beacon is an additional signing input. The adversary can ask for signatures for any message and beacon of its choice and wins if it produces a valid, fresh commitment–beacon–signature triple. Note that we do *not* require the adversary to output, or even know, a message opening the forged commitment.

The signing oracle is

$$\frac{\mathcal{O}_{\mathsf{Sign}}(\mathsf{pp}, \mathsf{sk}, (\mathbf{m}, \mathsf{bcn}))}{\sigma \leftarrow \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn})}$$
$$\textbf{return } \sigma$$

A pair $(\mathsf{com}, \mathsf{bcn})$ is *fresh* if the adversary made no oracle query $(\mathbf{m}, \mathsf{bcn})$ satisfying $\mathsf{com} = \mathsf{Com.Commit}(\mathsf{pp}, \mathbf{m})$. We say that the scheme is *existentially unforgeable under chosen-message attacks (EUF-CMA)* if, for every monolithic PPT adversary $\mathcal{A}$,

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

Fractional Individual Extractability. The key property of beholder signatures is that, from the local fast-query transcripts generated while producing an accepting signature, one can extract a valid signing key and a sufficiently large fraction of the committed data. Fix an extraction rate $\gamma \in (0, 1]$, a public message length $N \geq 1$ measured in blocks, and an online slow-query bound s. 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 write $\mathcal{A}^{\mathsf{H}}$ instead of $\mathcal{A}^{\Omega_{\mathsf{H}}^{\mathsf{MPC}}}$. In the preprocessing phase, $\mathcal{A}$ receives pp, may perform arbitrary polynomial-time computation, and may make any polynomial number of slow queries not subject to the online bound s; it then outputs a state st. In the online signing phase, $\mathcal{A}$ receives the state st and a uniformly random beacon bcn, may make any polynomial number of fast queries and at most s slow queries, and outputs $(\mathsf{pk}, \mathsf{com}, \sigma)$. The public key pk may be chosen adversarially. For each $i \in [a]$, let $\mathcal{T}_i$ denote the local fast-query transcript of $\mathcal{A}_i$ from the online phase, and let $(\mathsf{sk}_i', I_i, D_i, O_i)$ denote the output of $\mathsf{Ext}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}, \sigma, \mathcal{T}_i)$, where $I_i \subseteq [N]$ and, for every $j \in I_i$, the values $D_i[j]$ and $O_i[j]$ are a purported block and its opening.

We say that BSig is γ-*fractionally individually extractable* if there exists a polynomial-time extractor Ext such that, for every distributed PPT adversary $\mathcal{A}$ as above,

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

In other words, except with negligible probability, an accepting output must reveal valid signing keys for some nonempty set of parties, and those parties must collectively reveal valid openings for at least a γ fraction of the committed blocks. 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 obtain a fully individually extractable scheme by first encoding the underlying data with an erasure code of rate γ. Any extracted set containing at least a γ fraction of the coded blocks then suffices to recover the original data. This encoding is implicitly assumed for applications.

Beacon value. The beacon makes the signature fresh: although a signer may delete the committed data after signing, an accepting signature for an unpredictable beacon certifies knowledge of the data during the online signing computation following the beacon's revelation. This is crucial for applications such as proofs of data possession. Since the preprocessing phase is not subject to the online slow-query bound, the beacon must be unpredictable before the online signing phase; otherwise the adversary could precompute the entire signature using MPC. Looking ahead, the construction includes bcn in every random-oracle query for domain separation, making preprocessing queries useless after a fresh beacon is revealed, except with negligible probability.

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 verified only with respect to the commitment. Therefore, we impose the stronger requirement 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 depends on the data being signed, and the parameters and running times of the construction depend on N. Moreover, our goal is to obtain succinct signatures, i.e., $|\sigma| = \mathsf{poly}(\kappa, \log N)$. We refer to Sects. 6 and 7.3 for a discussion of how the message length influences the parameters of the scheme.

Straight-line Extraction. Our extractor is straight-line: it takes a single local fast-query transcript as input and does not rewind the adversary. Consequently, each subadversary can record its own fast queries and run the extractor on its own transcript; this would not be possible with a rewinding extractor. Thus, producing an accepting signature forces a valid secret key to become extractable from at least one local transcript and a γ fraction of the data to become collectively extractable from the local transcripts of the key-extracting subadversaries. Straight-line extraction is also necessary when proving security against quantum adversaries or in a composable setting. In the distributed adversarial model, it makes the threat of extraction credible to parties attempting to outsource signing through MPC.

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 separate random oracle with κ-bit output. 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 [14], 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 [21], 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.

As a starting point to our protocol, consider the protocol depicted in Fig. 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 $j = H(c)$.

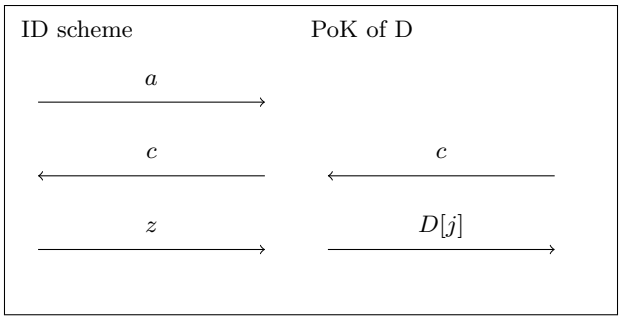


Fig. 1. Schematic starting point for our protocol, $j = \mathsf{H}(c)$.

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. Sect. 6), and hash *all* of them, i.e., check whether: $\mathsf{H}_d(x, \boldsymbol{a}, c, z, i, k, D[j_1], \ldots, D[j_m]) = 0^d$.

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: $\mathsf{H}_d(x, \boldsymbol{a}, c, z, i, k, D[j_1], \ldots, D[j_m], \mathsf{op}_{j_1}, \ldots, \mathsf{op}_{j_m}) = 0^d$.

Unfortunately, this construction would still not be secure against distributed adversaries. As recent works [9] have shown, 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 random oracle.[9] 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 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 Sect. 4.1.

The final algorithm. The pseudocode for the final algorithm is given in Fig. 2. The public parameters fix N, ρ, d, and m; in particular, these values are not chosen by the signature. The index sampler H uses a separate random-oracle domain: for every distinct input (i, c), it returns an independent uniform tuple in $[N]^m$, with independent coordinates sampled with replacement. We require $\{0, \ldots, 2^d - 1\}$ to be contained in the challenge space of the underlying identification scheme. The beholder verifier independently restricts each

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

non-$\perp$ challenge to this range. A well-formed signature contains exactly ρ entries in each of $\boldsymbol{a}$, $\boldsymbol{c}$, and $\boldsymbol{z}$, and ρ rows of exactly m entries in each of D' and $\mathbf{op}'$. If $c_i = \perp$, the corresponding response and data/opening entries are also $\perp$; otherwise, all these entries must have the types expected by the identification and commitment schemes. Malformed signatures are rejected. 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 $D[j]$.[10] 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 verifier first checks the signature format, the beacon against its external input, and the challenge range, and recomputes $\mathsf{prelude} = G((\mathsf{pk}, \mathsf{bcn}, \mathsf{com}), \boldsymbol{a})$. It then 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) [11], 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 total 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 Sect. 4.1). This parameter is a fixed polynomial in κ, as per Sect. 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:
  - $2^{d-4}\rho \geq 4N$, to leave at least $2N$ complete attempts after the exclusions in Lem. 5,

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

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

1 : set $c_i = z_i = \perp$ 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 : $\mathsf{pk} := \mathsf{GetPk}(\mathsf{sk})$
8 : $\mathsf{com} := \mathsf{Com.Commit}(\mathsf{pp}, D)$
9 : $\mathsf{prelude} := G((\mathsf{pk}, \mathsf{bcn}, \mathsf{com}), \boldsymbol{a})$
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 : $\quad\quad\quad c_i \coloneqq c$
16 : $\quad\quad\quad z_i \coloneqq z$
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 : $\mathsf{op}'_{i,k} := \mathsf{op}_{j_{i,k}}$ for $i \in [\rho], k \in [m]$ $\quad$ // $\perp$ if $c_i = \perp$
22 : $D'_{i,k} := D[j_{i,k}]$ for $i \in [\rho], k \in [m]$ $\quad$ // $\perp$ 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 : **if** σ is malformed or $\mathsf{bcn}' \neq \mathsf{bcn}$ **then return** 0
3 : $\mathsf{prelude} := G((\mathsf{pk}, \mathsf{bcn}, \mathsf{com}), \boldsymbol{a})$
4 : $\mathrm{corr} := 0$
5 : **for** $i = 1, \ldots, \rho$ **do**
6 : $\quad$ **if** $c_i = \perp$ **then continue**
7 : $\quad$ **if** $c_i \notin \{0, \ldots, 2^d - 1\}$ **then return** 0
8 : $\quad (j_{i,1}, ..., j_{i,m}) := \mathsf{H}(i, c_i)$
9 : $\quad$ **if** $\mathsf{V}(\mathsf{pk}, a_i, c_i, z_i) = 1$
10 : $\quad\quad \wedge\ \forall k \in [m] \colon \mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, D'_{i,k}, \mathsf{op}'_{i,k}, j_{i,k}) = 1$
11 : $\quad\quad \wedge \oplus_{k \in [m]} \mathsf{H}_d(\mathsf{prelude}, c_i, z_i, i, k, D'_{i,k}, \mathsf{op}'_{i,k}) = 0^d$
12 : $\quad$ **then** $\mathrm{corr} \coloneqq \mathrm{corr} + 1$
13 : **endfor**
14 : **return** $\mathrm{corr} \geq \rho/2$

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 $g^{z_i} = a_i \mathsf{pk}^{c_i}$.

- $2^{d-4}\rho \geq 2(s + \rho a)$, to ensure that at most half the complete attempts are touched by slow queries or subadversaries from whom the key is not extracted,
- $m \geq 8/\log(1/\gamma)$, so that the coverage bound in Lems. 4 and 5 rules out rejection sampling that hides more than a $1 - \gamma$ fraction of the data.

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 Sect. 3.3, the ID scheme satisfies the properties of Sect. 3.2 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 Sect. 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 Props. 1 to 3.

6.2 Correctness

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

Proof. If the signing algorithm from Fig. 2 outputs $\sigma = \mathsf{BSig.Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{m}, \mathsf{bcn}) \neq \bot$, 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 Thm. 1, the beholder signature scheme is* γ*-fractionally extractable against any distributed adversary consisting of at most* a *subadversaries, making at most* s *slow queries in total.*

The extractor. As required by the definition, we define the extractor in Fig. 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 $\mathsf{sk} \neq \bot$, 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 $(p, c, z, i, k, w, \mathsf{op})$, discarding malformed queries, cf. Fig. 2.
4. For each $q = (p, c, z, i, k, w, \mathsf{op})$, check:
   1. $i \in [\rho]$, $k \in [m]$, and $c \in \{0, \ldots, 2^d - 1\}$;
   2. $p = G(x, \boldsymbol{a})$ where $x = (\mathsf{pk}, \mathsf{bcn}, \mathsf{com})$;
   3. $\mathsf{V}(\mathsf{pk}, a_i, c, z) = 1$;
   4. the opening verifies: $\mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, w, \mathsf{op}, j) = 1$ where $j := H(i, c)[k]$.
5. Discard any elements that have failed the checks and let $\mathcal{Q}'$ be the set of the remaining elements. For each $i \in [\rho]$, let $\mathcal{Q}_i \subseteq \mathcal{Q}'$ contain the queries whose iteration index is i.
6. If there exist i and two queries $q_1, q_2 \in \mathcal{Q}_i$ with distinct challenges $c_1 \neq c_2$, set $\mathsf{sk} := \mathsf{Ext}_{\mathsf{ID}}(\mathsf{pk}, (a_i, c_1, z_1), (a_i, c_2, z_2))$. Otherwise, output $\bot$.
7. For each $q = (p, c, z, i, k, w, \mathsf{op}) \in \mathcal{Q}'$, set $D[j] := w, \mathsf{ops}[j] := \mathsf{op}$ 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* $\mathsf{Com.Verify}(\mathsf{pp}, \mathsf{com}, D^*[j], \mathsf{ops}[j], j) = 1$ *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 online hash transcript, let $\mathcal{T}_{\mathsf{fast}}$ be the fast online 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.

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 Sects. 7.3 and 7.5 and our goal of achieving compact signature sizes. In particular, keeping ρ low is essential for minimizing the concrete signature size.

Lemma 2. *Assume the parameter conditions of Sect. 6 and* $\rho = \omega(\log \kappa)$*. For every* s*-bounded PPT signer in the extractability experiment, except with negligible probability, an accepting output has at least* $2^{d-4}\rho$ *complete attempts, each containing* m *distinct useful online queries. For such an output,* $|\mathcal{T}| \geq 2^{d-4}\rho m$ *and* $|\mathcal{T}_{\mathsf{fast}}| \geq 2^{d-4}\rho m - s$*.*

Proof idea. A valid signature needs $\rho/2$ rows i for which the attempt (i, c_i) is successful, i.e., the xor of the d-bit answers to its m queries is 0^d. Once an attempt is complete, this xor is uniform, so the attempt is successful with probability exactly 2^{-d}, independently of everything before. Attempts are thus lottery tickets, and a valid signature needs $\rho/2$ winning ones. A signer that completes fewer than $L := 2^{d-4}\rho$ attempts expects only $\rho/16$ wins, so it must be much more lucky than expected. By a Chernoff-type bound this has probability $e^{-\Omega(\rho)}$, which is negligible since $\rho = \omega(\log \kappa)$. Hence, an accepting signer completes at least L attempts. By Clm. 1, distinct attempts consist of distinct queries, so there are at least Lm queries, of which at most s are slow.

The full proof is longer than this sketch because the signer is adaptive in two ways. First, the signer chooses which attempts to complete and when to stop, e.g., it may stop as soon as it has enough wins. Hence, we cannot apply a Chernoff bound to the attempts it happens to complete, as their outcomes are correlated with its choices. Instead, we look at the first completed attempts in the order in which they appear. Each one is a fresh coin flip whichever attempt it is, so their outcomes are independent and Chernoff applies (Step 1). Second, the lottery is fresh only for a fixed prelude input (the string hashed into the common first component p of all queries), whereas the signer chooses it. Since G is a random oracle, it cannot do so freely: except with negligible probability, the prelude input of the signature was already queried to G before its prelude was first used. There are only polynomially many such candidates, so we can run the fixed-prelude argument for each of them and take a union bound (Step 2). Finally, the signer may also include rows without querying them, hoping to get lucky, adding at most ρ further tickets.

Proof. Put $L' := L + \rho$. The prelude is $p = G(M)$ for the prelude input $M = (x, \boldsymbol{a})$, and we write M_σ for the prelude input of the output signature σ. Usefulness is relative to M_σ. For an arbitrary M we say *M-useful* when a query passes the checks of Fig. 3 with M in place of M_σ. This depends only on the query, on M and on the index sampler H, so it is known before the query is answered. The hash G is a random oracle with κ-bit output.

The padded transcript. Unfortunately, the transcript $\mathcal{T}$ alone does not capture everything the signer can do. A signer may include a row in σ without ever querying its m inputs, hoping to get lucky: the row then passes the proof-of-work check with probability 2^{-d}, but it is not an attempt in $\mathcal{T}$. To count such guesses, we extend the transcript after the signer has output σ. The *padded transcript* $\mathcal{T}'$ is $\mathcal{T}$ followed by the following *padding queries*: first $G(M_\sigma)$, then, for every row i of σ with $c_i \neq \bot$, the m inputs $(p, c_i, z_i, i, k, D'_{i,k}, op'_{i,k})$ of this row that have not been queried yet. Each padding query is answered by the oracle: with the value already defined, or with a fresh uniform one. Note that $\mathcal{T}'$ is only a proof device. Below we show that $\mathcal{T}'$ contains at least L' complete attempts, and then derive the statement about $\mathcal{T}$. We apply *attempt* and *complete* to $\mathcal{T}'$, and, for each $k \in [m]$, we take the earliest useful query with indices (i, c, k). An attempt is *successful* if the answers of H_d to these m queries xor to 0^d, as in line 14 of Fig. 2. Let Q_G and B be polynomial bounds on the number of distinct G-queries and of distinct preludes occurring in H_d-queries of $\mathcal{T}'$. We now have the following observation.

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. Sect. 3.2), or (2a) the position binding or (2b) the uniqueness of openings of the commitment scheme (cf. Sect. 3.3). Then the claim about m useful queries follows from pigeonholing. □

Let U be the event that the conclusion of Clm. 1 fails, i.e., $\mathcal{T}'$ contains two distinct useful queries with the same (i, c, k). By the claim, $\Pr[U] \leq \mathsf{negl}(\kappa)$. Let Bad be the event that σ is accepting, fewer than L' attempts are complete in $\mathcal{T}'$, and U does not occur. It suffices to show that $\Pr[\mathsf{Bad}]$ is negligible.

Step 1: fixed prelude input. By the *state* of the experiment at a given moment we mean the coins and internal states of all subadversaries together with the current tables of G, H and H_d, i.e., all oracle answers sampled so far. Consider a state S with the following properties:

- the beacon has already been sampled,
- a query with some prelude p is pending, meaning it is the first H_d-query with prelude p and has not been answered yet, and

– the table of G contains an entry $G(M) = p$.

We fix such an S. The only randomness left is the answers to oracle inputs not yet in the tables. We claim that, over this randomness,

$$\Pr\left[\mathsf{Bad} \wedge M_\sigma = M \,\middle|\, S\right] \le \mathsf{negl}(\kappa).$$

Since no H_d-query with prelude p has been answered before S, all M-useful queries occur at or after S.

We view the complete attempts of $\mathcal{T}'$ as lottery tickets, drawn one after another. Let Y_t be the success indicator of the t-th attempt to become complete, and let the *history* of trial t be everything sampled before the answer that completes this attempt: the coins of the signer and all oracle answers so far. We set Y_t to an independent Bernoulli(2^{-d}) dummy if fewer than t attempts become complete. Y_t equals 1 with probability exactly 2^{-d}, with all Y_t i.i.d. Now suppose Bad holds and $M_\sigma = M$. Then fewer than L' attempts are complete in $\mathcal{T}'$, so all of them are among the first $L' - 1$ trials, and $\sum_{t<L'} Y_t$ counts all successful complete attempts. On the other hand, σ has at least $\rho/2$ valid rows. The m queries of a valid row i are M-useful and lie in $\mathcal{T}'$, either made by the signer or added as padding. Since U does not occur, they are exactly the queries of the attempt (i, c_i). So this attempt is complete, and the proof-of-work check of the row says that it is successful. Distinct rows give distinct attempts, hence $\sum_{t<L'} Y_t \ge \rho/2$.

Tail bound. By the above, $Y_1, \ldots, Y_{L'-1}$ are i.i.d. Bernoulli(2^{-d}) variables. Their sum X has mean $\mu = (L'-1)2^{-d} \le (L+\rho)2^{-d} \le \rho/16 + \rho/32 = 3\rho/32$, because $L2^{-d} = \rho/16$ and $d \ge 5$. By the Chernoff bound $\Pr[X \ge t] \le (e\mu/t)^t$ for $t > \mu$, applied with $t = \rho/2$ (and monotone in μ, so we may use the upper bound),

$$\Pr\Big[\sum_{t<L'} Y_t \ge \rho/2\Big] \le \Big(\frac{3e}{16}\Big)^{\rho/2} \le e^{-0.3\rho}.$$

As $\rho = \omega(\log \kappa)$, this is negligible. It depends only on ρ, so the same negligible function bounds the probability for every state S and prelude input M, which proves the claim.

Step 2: adaptive prelude input. Suppose Bad holds, and let $p_\sigma := G(M_\sigma)$. Since σ has a valid row, $\mathcal{T}'$ contains an H_d-query with prelude p_σ and a G-query on M_σ (possibly a padding query). Consider the first G-query on M_σ. There are three cases.

1. *It is made before the beacon.* Then M_σ contains the beacon, which is uniform and independent of the preprocessing phase. For each of the at most Q_G queries this has probability $2^{-\kappa}$, so the case has probability at most $Q_G\, 2^{-\kappa}$.
2. *It is made after the beacon, but an H_d-query with prelude p_σ was already answered.* The answer $G(M_\sigma)$ is uniform and independent of the past, so it hits one of the at most B preludes used so far with probability at most $B\, 2^{-\kappa}$. Over the Q_G queries, this case has probability at most $Q_G B\, 2^{-\kappa}$.
3. *It is made after the beacon, and no H_d-query with prelude p_σ was answered yet.* Let M_t be the t-th distinct G-query, and let S_t be the state just before the first H_d-query with prelude $G(M_t)$ is answered (if M_t was queried after the beacon and such a query occurs). The state S_t satisfies the hypothesis of Step 1 with $M = M_t$. In this case, $M_\sigma = M_t$ for some $t \le Q_G$, so by Step 1 and a union bound over t, the probability is at most $Q_G \cdot \mathsf{negl}(\kappa)$.

Therefore $\Pr[\mathsf{Bad}] \le Q_G \cdot \mathsf{negl}(\kappa) + (B+1)\, Q_G\, 2^{-\kappa}$.

Conclusion. The probability that σ is accepting while fewer than L' attempts are complete in $\mathcal{T}'$ is at most $\Pr[U] + \Pr[\mathsf{Bad}]$. Both terms are negligible: the first by Clm. 1, and the second by Step 2, since Q_G and B are polynomials. Otherwise at least $L' = L + \rho$ attempts are complete in $\mathcal{T}'$. An attempt that is complete in $\mathcal{T}'$ but not in $\mathcal{T}$ contains a padding query, so it is (i, c_i) for one of the at most ρ rows of σ. Hence at least L attempts are complete in $\mathcal{T}$. Their sets of m queries are pairwise disjoint and contained in $\mathcal{T}$, so $|\mathcal{T}| \ge Lm$. At most s of these queries are slow, so $|\mathcal{T}_{\mathsf{fast}}| \ge Lm - s$. □

We say that sk was *leaked* to $\mathcal{A}_u$ if useful queries in $\mathcal{T}_u$ contain two accepting ID transcripts of the form $(a_i, c, z), (a_i, \tilde{c}, \tilde{z})$ with $c \ne \tilde{c}$ for some $i \in [\rho]$. Then the extractor recovers the secret key from $\mathcal{T}_u$ by special soundness. 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. By Lem. 2, $\mathcal{T}_{\mathsf{fast}}$ contains at least $2^{d-4}\rho m - s$ distinct useful queries. Therefore, there exists an i for which $\mathcal{A}$ made at least $2^{d-4}m - s/\rho$ queries. By Clm. 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 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 Fig. 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 Lem. 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 Fig. 4. In the lemma, we will think of the X_i as the m-tuples of indices as computed in line 13.

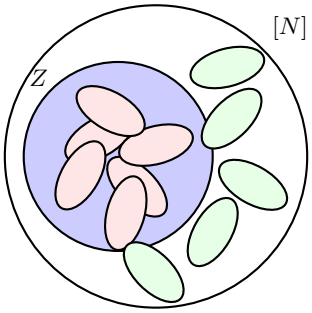


Fig. 4. A depiction of Lem. 4: the ellipses are the sets being sampled, and the red of them fall into the set Z of relative size γ (X_i for $i \in I$), while the green ones are discarded by the adversary.

Lemma 4. *Let N, p, m be positive integers, and let $X_1, \ldots, X_p$ be independent uniform samples from $[N]^m$. Write $X_i \subseteq Z$ when every coordinate of X_i belongs to Z. For $\varepsilon, \gamma \in (0,1)$, set $r = \lceil \varepsilon p \rceil$ and $k = \lfloor \gamma N \rfloor$. Then*

$$\begin{aligned} &\Pr\big[\exists Z \subseteq [N],\ |Z| \leq \gamma N,\ I \subseteq [p],\ |I| \geq \varepsilon p :\ \forall i \in I,\ X_i \subseteq Z\big] \\ &\qquad \leq \min\left\{1, \binom{N}{k}\binom{p}{r}\left(\frac{k}{N}\right)^{mr}\right\} \leq \min\{1, 2^{-z}\}, \qquad (1) \\ &z := rm\log(1/\gamma) - \log\binom{p}{r} - \log\binom{N}{k} \geq r\big(m\log(1/\gamma) - \log(ep/r)\big) - N. \end{aligned}$$

Here and below logarithms are base two, and e is Euler's number. The same bound holds when each X_i is an independent uniform m-element subset of $[N]$, provided $m \leq N$.

Proof idea. Call a pair (Z, I) *bad* if $|Z| \leq \gamma N$, $|I| \geq \varepsilon p$, and $X_i \subseteq Z$ for all $i \in I$. We bound the probability that a bad pair exists. The argument is a compression argument: if a bad pair exists, then $(X_1, \ldots, X_p)$ can

be described by naming the pair and then writing each X_i with $i \in I$ over the small set Z instead of over $[N]$, which saves about $m \log(1/\gamma)$ bits per sample. Instead of counting bits, we count outcomes directly. A fixed pair of the extremal sizes is bad with probability about γ^{mr}, and we take a union bound over all $\binom{N}{k}\binom{p}{r}$ such pairs. This number does not depend on m, whereas every extra coordinate of the samples multiplies the probability by γ. For large enough m, the factor γ^{mr} therefore beats the number of pairs.

Proof. Let B be the event on the left-hand side of (1), i.e., the event that a bad pair exists. We first note that $1 \le r \le p$ and $0 \le k \le N$, since $0 < \varepsilon p \le p$ and $\gamma < 1$. For $Z \subseteq [N]$ and $I \subseteq [p]$, let $E_{Z,I}$ be the event

$$E_{Z,I} = \{\forall i \in I \colon X_i \subseteq Z\},$$

and let $\mathcal{P}$ be the set of pairs (Z, I) with $|Z| = k$ and $|I| = r$.

Step 1: it suffices to consider pairs in $\mathcal{P}$. We show that $B \subseteq \bigcup_{(Z,I)\in\mathcal{P}} E_{Z,I}$. Suppose B holds, and let (Z, I) be a bad pair.

- Since $|Z|$ is an integer, $|Z| \le \lfloor \gamma N \rfloor = k \le N$. Hence, we can enlarge Z to a set $Z' \supseteq Z$ with $|Z'| = k$.
- Since $|I|$ is an integer, $|I| \ge \lceil \varepsilon p \rceil = r$. Hence, we can shrink I to a subset $I' \subseteq I$ with $|I'| = r$.

Both changes only make it easier for the pair to be bad: for every $i \in I'$ we have $X_i \subseteq Z \subseteq Z'$, so $E_{Z',I'}$ holds, and $(Z', I') \in \mathcal{P}$.

Step 2: a fixed pair. Fix $(Z, I) \in \mathcal{P}$. Each X_i consists of m independent uniform elements of $[N]$, so $\Pr[X_i \subseteq Z] = (k/N)^m$. This is 0 when $k = 0$, because $m \ge 1$. The samples $X_1, \ldots, X_p$ are mutually independent, so

$$\Pr[E_{Z,I}] = \prod_{i\in I} \Pr[X_i \subseteq Z] = \Big(\frac{k}{N}\Big)^{mr}.$$

Step 3: union bound. We have $|\mathcal{P}| = \binom{N}{k}\binom{p}{r}$. By Steps 1 and 2,

$$\Pr[B] \;\le\; \sum_{(Z,I)\in\mathcal{P}} \Pr[E_{Z,I}] \;=\; \binom{N}{k}\binom{p}{r}\Big(\frac{k}{N}\Big)^{mr}.$$

Since also $\Pr[B] \le 1$, this proves the first inequality in (1).

Step 4: the exponent. Since $k/N \le \gamma$, we have $(k/N)^{mr} \le 2^{-rm\log(1/\gamma)}$, and therefore

$$\binom{N}{k}\binom{p}{r}\Big(\frac{k}{N}\Big)^{mr} \le 2^{-z}, \qquad \text{where} \quad z = rm\log(1/\gamma) - \log\binom{p}{r} - \log\binom{N}{k}.$$

Moreover, $\binom{N}{k} \le 2^N$, and $\binom{p}{r} \le p^r/r! \le (ep/r)^r$, where the last step uses $e^r \ge r^r/r!$ (one term of the series for e^r). Hence

$$z \;\ge\; r\big(m\log(1/\gamma) - \log(ep/r)\big) - N.$$

as desired. □

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 $N \ge \kappa$, otherwise the message can be artificially inflated with erasure coding.

Lemma 5. *Let $P \subseteq [a]$ be the set of subadversaries from whose fast transcripts the extractor recovers the secret key, which is non-empty by Lem. 3. For each $u \in P$, let I_u be the set of indices of D recovered from that transcript, and let $J = \bigcup_{u\in P} I_u$. Suppose that $N \ge \kappa$, $m \ge 8/\log(1/\gamma)$,*

$$2^{d-4}\rho \ge 2(s + \rho a) \tag{2}$$

and

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

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

Proof. Call an attempt (i, c) complete if its m distinct useful queries all occur in $\mathcal{T}$. On the completed-attempt lower-bound event of Lem. 2, there are at least $p_{\min} = 2^{d-4}\rho$ complete attempts. The verifier and extractor restrict $i \in [\rho]$ and $0 \leq c < 2^d$, so the fixed universe $\mathcal{U} = [\rho] \times \{0, \ldots, 2^d - 1\}$ contains exactly $p_{\max} = 2^d\rho$ possible attempts.

Exclude every complete attempt touched by a useful slow query. There are at most s such attempts, since each of the at most s slow queries belongs to a single attempt. This exclusion includes attempts that combine fast and slow queries; a single slow query suffices to exclude an attempt. Next, exclude every complete attempt touched by a useful fast query of a subadversary outside P. For each fixed i, such a subadversary can query at most one challenge c; otherwise the extractor recovers the secret key by special soundness. Thus each of the $a - N_{\mathsf{leaked}}$ subadversaries outside P touches at most ρ attempts, and at most $\rho(a - N_{\mathsf{leaked}}) \leq \rho a$ additional attempts are excluded. By Eq. (2), at least

$$p_{\min} - s - \rho a \geq p_{\min}/2 =: p_{\text{fast}} = 2^{d-5}\rho$$

complete attempts remain. All m useful queries of every remaining attempt occur in the fast transcripts of subadversaries in P. Their queried data indices therefore belong to J, even when different queries of an attempt are made by different subadversaries in P.

For each $(i, c) \in \mathcal{U}$, let $X_{i,c}$ be the sample returned by $H(i, c)$. Apply Lem. 4 to this entire fixed family of $p_{\max}$ samples, including samples that the adversary never requests, with

$$\varepsilon = \varepsilon_{\min} := p_{\text{fast}}/p_{\max} = 1/32.$$

The event in that lemma quantifies over every subset of at least p_{fast} samples, so it includes the adaptively selected remaining attempts. In particular, the family size used in the bound does not depend on when the adversary stops. As $\log(1/\varepsilon_{\min}) = 5$, choose

$$m \geq \frac{8}{\log(1/\gamma)}. \tag{4}$$

Using Eq. (3), we have $p_{\text{fast}} = p_{\min}/2 \geq 2N$. With $r = \lceil p_{\text{fast}} \rceil$, we have $p_{\max}/r \leq 32$, so the exponent in Lem. 4 satisfies

$$\begin{aligned} z &\geq r\big(m \log(1/\gamma) - \log(e p_{\max}/r)\big) - N \\ &\geq r(3 - \log e) - N \\ &\geq r - N \geq 2N - N = N \geq \kappa, \end{aligned}$$

where $\log e < 2$. The coverage bound applies to the unconditional family of samples; we do not condition its distribution on the completed-attempt lower-bound event. Consequently, outside the failure event of Lem. 2 and an event of probability at most $2^{-\kappa}$, the samples of the remaining attempts cover at least γN indices, all contained in J. Therefore, $|J| \geq \gamma N$, as desired. □

This concludes the proof of Prop. 2. Combining Lems. 3 and 5 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. Sect. 4).*

Proof idea. At a high level, we follow the proof strategy of [35] 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. Cor. 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.

The one new difficulty is the proof-of-work condition. In each row, a real signer tries challenges $c = 0, 1, \ldots$ until the attempt (i, c) succeeds, say at $\hat{c}_i$, so every attempt with $c < \hat{c}_i$ *fails*. The reduction cannot run this loop without sk. Instead, it fixes $\hat{c}_i$ in advance, simulates a transcript for $\hat{c}_i$ alone, and programs H_d so that this attempt has the desired outcome. The attempts with $c < \hat{c}_i$ stay unprogrammed, so in the simulation each of them succeeds with probability 2^{-d}, which is non-negligible because d is logarithmic in κ. A forger could tell the two worlds apart by evaluating such an attempt, but this requires an accepting response for a challenge $c \neq \hat{c}_i$. Together with the simulated transcript for $\hat{c}_i$, this gives the reduction two accepting transcripts with the same first message, and hence sk by special soundness. So if the forger tries to distinguish, it is too late: the reduction already has what it wants. Otherwise, the forger's view is indistinguishable from the real one, and we extract sk from its forgery. The remaining details (freshness of the programmed inputs, SHVZK, and uniqueness of responses) are technical but unsurprising.

Proof. Let $\mathcal{B}_{\mathsf{BSig}}$ be a PPT forger. Since it is monolithic, it makes no slow queries, and the choice of the beacon is irrelevant. Let Q be a polynomial bound on the number of oracle queries, counting both the queries of $\mathcal{B}_{\mathsf{BSig}}$ and the inputs programmed by the reduction; it also bounds the number of signing queries. The reduction $\mathcal{A}_{\mathcal{R}}$ receives pk, runs $\mathcal{B}_{\mathsf{BSig}}$, simulates the random oracles G, H and H_d by lazy sampling, and stores the queries of $\mathcal{B}_{\mathsf{BSig}}$ in $\mathcal{Q}$. The reduction is depicted in Fig. 5.On a query (D, bcn) to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ (a *signing query*), the reduction computes $\mathsf{com} := \mathsf{Com.Commit}(\mathsf{pp}, D)$ and $x := (\mathsf{pk}, \mathsf{bcn}, \mathsf{com})$, and then the signature for this query as follows. For every row $i \in [\rho]$ it proceeds as follows.

1. It samples a random function $\tau_i : \{0,1\}^d \to \{0,1\}^d$, which is feasible since both sets have polynomial size. The value $\tau_i(c)$ plays the role of the xor of the hashes in the real signer's attempt (i, c). Row i *succeeds* if $\tau_i(c) = 0$ for some c, and then we let $\hat{c}_i$ be the smallest such c. In a failed row, $\hat{c}_i := 0$ is a placeholder.
2. It samples $(a_i, \hat{c}_i, \hat{z}_i) \leftarrow \mathsf{Sim}(\mathsf{pk}, \hat{c}_i)$. A successful row is published with $c_i := \hat{c}_i$ and $z_i := \hat{z}_i$, and a failed row with $c_i = z_i = \bot$.

Let $\boldsymbol{a} := (a_1, \ldots, a_\rho)$ and $(\hat{j}_{i,1}, \ldots, \hat{j}_{i,m}) := H(i, \hat{c}_i)$. The m hash inputs of the attempt $(i, \hat{c}_i)$ are

$$q_{ik} = \big(G(x, \boldsymbol{a}),\, \hat{c}_i,\, \hat{z}_i,\, i,\, k,\, D[\hat{j}_{i,k}],\, \mathsf{op}_{\hat{j}_{i,k}}\big), \qquad k \in [m]. \tag{5}$$

The reduction *programs the attempt* $(i, \hat{c}_i)$ by choosing the answers of H_d on $q_{i1}, \ldots, q_{im}$ uniformly subject to $\bigoplus_{k \in [m]} \mathsf{H}_d(q_{ik}) = \tau_i(\hat{c}_i)$. It does so for every row. The target is 0 for a successful row and nonzero for a failed one. This is possible because one simulated transcript suffices to write down all m inputs of the attempt: they depend on the transcript only through $(\hat{c}_i, \hat{z}_i)$, and the data blocks and openings are known to the reduction, which holds D. Failed rows are programmed as well. The reduction must still publish a_i, and it can sample a_i only via Sim, which fixes the challenge $\hat{c}_i$ together with an accepting response $\hat{z}_i$. This is exactly what is needed to program the attempt $(i, \hat{c}_i)$. If it were left unprogrammed, its xor would be uniform instead of nonzero, so the oracles would differ with probability $2^{-d} = 1/\mathrm{poly}$. A forger could exploit this by querying $q_{i1}, \ldots, q_{im}$, but such queries reveal only one accepting transcript, which does not let the reduction extract sk. After programming, the oracles agree at $\hat{c}_i$, so they differ only at challenges $c \neq \hat{c}_i$,

where a query does yield a second transcript. If fewer than $\rho/2$ rows succeeded, the reduction returns $\perp$. Otherwise it returns $\sigma = (\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{z}, \mathbf{op}, D', \mathsf{bcn})$, where the data and opening entries of failed rows are $\perp$. The responses $\hat{z}_i$ of failed rows stay private.

The reduction stores, for every signing query, the prelude of the signature it returns in response, together with the transcripts $(a_i, \hat{c}_i, \hat{z}_i)$ of its rows. If $\mathcal{B}_{\mathsf{BSig}}$ makes a H_d-query under such a prelude that contains an accepting transcript (a_i, c, z) for a challenge $c \neq \hat{c}_i$ of one of its rows, the reduction holds two accepting transcripts (a_i, c, z) and $(a_i, \hat{c}_i, \hat{z}_i)$ with the same first message and different challenges. By special soundness, it computes a valid secret key, outputs it and halts. All other queries are answered from the table, or uniformly if fresh. When $\mathcal{B}_{\mathsf{BSig}}$ outputs a forgery $(\mathsf{com}^*, \sigma^*, \mathsf{bcn}^*)$, the reduction runs $\mathsf{Ext}(\mathsf{pp}, \mathsf{pk}, \mathsf{com}^*, \sigma^*, \mathcal{Q})$ and outputs the secret key it returns, if any.

Programming is consistent. Programming an attempt requires that its inputs q_{ik} are still undefined. All inputs of a signing query share the prelude $p := G(x, \boldsymbol{a})$, so it suffices that p is not the prelude of an earlier H_d-input, i.e., one queried by $\mathcal{B}_{\mathsf{BSig}}$ or programmed for an earlier signing query. The bad event is

$$\mathsf{Inconsistent} \;:=\; \big\{ \text{the prelude of some signing query is the prelude of an earlier } \mathsf{H}_d\text{-input} \big\}.$$

Outside $\mathsf{Inconsistent}$, programming is possible and, in the real experiment, the answers on the inputs of every signing query are fresh, hence uniform and independent. To bound $\Pr[\mathsf{Inconsistent}]$, fix a signing query and let S be the set of preludes of earlier H_d-inputs, so $|S| \leq Q$. If $(x, \boldsymbol{a})$ was not queried to G before, then p is uniform and independent of S, so $p \in S$ with probability at most $Q\,2^{-\kappa}$. Otherwise, $\boldsymbol{a}$ is the second component of one of at most Q earlier G-queries. For honest commitments this has probability at most $Q\,2^{-h}$, where $h = \omega(\log \kappa)$ lower-bounds their min-entropy (Def. 5). For simulated commitments the probability differs only negligibly by SHVZK, since the event can be checked efficiently from the forger's view. A union bound over the signing queries gives $\Pr[\mathsf{Inconsistent}] \leq \mathsf{negl}(\kappa)$ in both experiments. For the real signer, the same argument applies without the SHVZK step.

Real vs. simulated signing. Let ε be the probability that $\mathcal{B}_{\mathsf{BSig}}$ outputs a valid fresh forgery in the real experiment. In either experiment, row i of a signature returned by the signer comes with a transcript $(a_i, \hat{c}_i, \hat{z}_i)$, where $\hat{c}_i$ is the first successful challenge of the row (or 0 in a failed row) and $\hat{z}_i$ is the corresponding response. In the simulation, the transcript is sampled by Sim. In real signing, $\hat{z}_i$ is computed by the signer using sk, and in a failed row it is not published. A H_d-query $(p, c, z, i, k, w, \mathsf{op})$ of $\mathcal{B}_{\mathsf{BSig}}$ is *revealing* if p is the prelude of some returned signature and (a_i, c, z) is an accepting transcript with $c \neq \hat{c}_i$, where a_i and $\hat{c}_i$ belong to row i of that signature. These are the queries on which the reduction halts with sk. Let Detect be the event that $\mathcal{B}_{\mathsf{BSig}}$ makes a revealing query. We show that the forger's view is the same in the real and in the simulated experiment until Detect occurs, up to a negligible term.

First, we locate the difference between the real and the simulated signing oracle. Outside $\mathsf{Inconsistent}$, the inputs of the attempts of a returned signature have fresh preludes, so the answers on them are independent of all other answers. In real signing, the xors of the attempts (i, c) are uniform and independent, conditioned on the signer's stopping rule, i.e., nonzero before $\hat{c}_i$ and zero at $\hat{c}_i$ (in a failed row: nonzero everywhere). This is exactly the distribution of τ_i in the simulation, which enforces $\tau_i(\hat{c}_i)$ by programming but nothing else. So the two oracles differ only on the inputs of the attempts (i, c) with $c \neq \hat{c}_i$ that the real signer evaluates and fails, i.e., those with $c < \hat{c}_i$ (in a failed row, all $c \neq \hat{c}_i$). There, the real xor is nonzero, while the simulated answers are uniform. On all other inputs, the answers are identically distributed in both experiments. This includes the programmed inputs at $\hat{c}_i$, inputs with a response that is not accepting, and the attempts that the real signer never evaluates.

Second, every input on which the two oracles differ is of the form $(p, c, z, i, k, w, \mathsf{op})$, where p is the prelude of a signature returned by the signing oracle, $c \neq \hat{c}_i$, and z is a response such that (a_i, c, z) is accepting. Querying such an input is a revealing query. Hence, until Detect, the forger queries only inputs on which the oracles are identically distributed, so, given the transcripts $(a_i, \hat{c}_i, \hat{z}_i)$ of the returned signatures, its view has the same distribution in both experiments. Once Detect occurs, the reduction already holds sk, so the rest of the view is irrelevant.

Third, we replace the real transcripts by simulated ones. By SHVZK, $\mathsf{Sim}(\mathsf{pk}, c)$ outputs a transcript that is indistinguishable from a real transcript with challenge c, and in both experiments $\hat{c}_i$ is determined by τ_i, independently of a_i. We hence replace the at most $Q\rho$ real transcripts one at a time. A distinguisher embeds its SHVZK challenge as row i, programs this row's attempt as the reduction does, and runs the forger. Until Detect, the real oracle behaves like the programmed one, so the distinguisher can answer oracle queries and decide Detect and any event A determined by the forger's view. Each step thus changes the probability of such an event by at most a negligible term. Together with $\Pr[\mathsf{Inconsistent}] \leq \mathsf{negl}(\kappa)$ in both experiments, this gives, for every event A determined by the forger's view,

$$\Pr_{\text{sim}}[A \wedge \neg\mathsf{Detect}] = \Pr_{\text{real}}[A \wedge \neg\mathsf{Detect}] \pm \mathsf{negl}(\kappa) \quad \text{and} \quad \Pr_{\text{sim}}[\mathsf{Detect}] = \Pr_{\text{real}}[\mathsf{Detect}] \pm \mathsf{negl}(\kappa).$$

Taking A to be the event that the forger outputs a valid fresh forgery yields

$$\Pr_{\text{sim}}\left[\mathsf{Detect} \ \vee \ \text{valid fresh forgery}\right] \ \geq \ \varepsilon - \mathsf{negl}(\kappa).$$

The forgery. Suppose $\mathcal{B}_{\mathsf{BSig}}$ outputs a valid forgery and Detect has not occurred. Since $(\mathsf{com}^*, \mathsf{bcn}^*)$ is fresh, no signing query (D, bcn) has $\mathsf{bcn} = \mathsf{bcn}^*$ and $\mathsf{com}^* = \mathsf{Com.Commit}(\mathsf{pp}, D)$. So the prelude input $(x^*, \boldsymbol{a}^*)$ of the forgery differs from that of every signing query, and by collision resistance of the random oracle G, its prelude p^* differs from all programmed preludes, except with probability $Q^2\, 2^{-\kappa}$. Hence all queries under p^* receive plain random-oracle answers. The simulated signing oracle uses only pk and Sim, and the extractability argument looks only at queries under p^*, so it applies to $\mathcal{B}_{\mathsf{BSig}}$ together with the simulated oracle (Cor. 1). As $\rho = \omega(\log \kappa)$, Ext outputs a valid sk except with negligible probability.

This is where uniqueness of responses is used (Clm. 1). The forger may reuse the commitments $\boldsymbol{a}$ of a signing query with a new x^*, and then it already knows accepting transcripts $(a_i, \hat{c}_i, \hat{z}_i)$ for the successful rows. Without uniqueness, it could grind over many responses accepting for the same $(a_i, \hat{c}_i)$. Each would be a new attempt under p^* with an independent chance of passing the proof-of-work check, and no second challenge, hence no special soundness, would ever be needed. With unique responses, an attempt is determined by (i, c), so a forger that completes enough attempts uses two challenges in some row, and Ext recovers sk.

By the paragraphs, $\mathcal{A}_{\mathcal{R}}$ outputs a valid secret key whenever Detect occurs (by special soundness), or the forger outputs a valid fresh forgery without Detect (by extraction, except with negligible probability). Hence, it succeeds with probability at least $\varepsilon - \mathsf{negl}(\kappa)$. Since $\mathcal{R}$ is hard, ε is negligible. □

7 Efficiency analysis

In this section, we report the concrete evaluation of our beholder signatures.

7.1 Implementation

Our Rust implementation is available at `https://github.com/marmistrz/beholders`. It uses the multi-threaded rust-kzg library for KZG commitments [27], the BLS12-381 curve for KZG and Schnorr, and the SHA-512 compression function for the random oracle H. Rayon parallelizes the mining phase. The implementation omits the beacon from the prelude, so our timings measure its core operations. The freshness estimates below apply to the construction with a beacon in Fig. 2.

7.2 Computing the openings

Computing one KZG opening for a vector of N elements takes $O(N)$ operations. Computing all openings separately in line 2 would therefore take $O(N^2)$ operations. We use the FK20 technique [18] to compute them together in $O(N \log N)$ operations. For the experiments, we generate a trusted setup large enough for each data size.

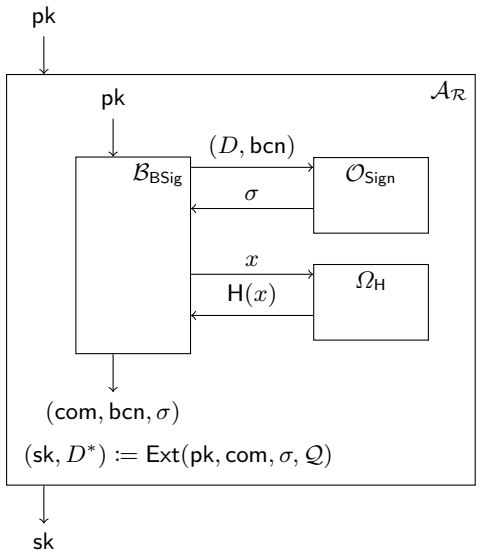


Fig. 5. A schematic depiction of the reduction $\mathcal{A}_{\mathcal{R}}$ to the hardness of the relation $\mathcal{R}$, proving the unforgeability of the beholder signature scheme BSig.

File size [KiB]	d	FK20 [s]	Mining [s]	Freshness period
16	12	0.274	0.442	1 min 3 s
32	13	0.571	0.853	2 min 7 s
64	14	1.233	1.799	4 min 15 s
128	15	2.619	3.432	8 min 31 s
256	16	5.537	6.008	17 min 3 s
512	17	11.933	15.276	34 min 7 s
1024	18	26.375	28.297	1 h 8 min 15 s
2048	19	54.201	62.999	2 h 16 min 31 s

Table 1. Mean prover times for 10 verified signatures per vector size, with $m = 8$ and $\rho = 10$. Freshness uses the displayed d, $a = 2$, and an aggregate rate of 20 slow queries per second.

7.3 Experimental results

We ran the experiments on an Apple M4 with macOS 27.0.1 and ten Rayon worker threads. Sizes refer to committed vectors of N field elements of 32 bytes each; erasure-code expansion and coding time are excluded. We use $m = 8$, $\rho = 10$, and $d = \lceil \log(64N/\rho) \rceil$ for both the mining measurements and freshness estimates.

Tab. 1 reports FK20 and mining times separately. The openings depend only on the data and can be computed offline and reused across signatures. In the construction with a beacon, mining starts after the beacon is revealed. Each sample computes the openings once, then mines until it obtains a signature that verifies. Mining time includes any restarts with fresh commitments; verification is timed separately in Tab. 2. All times are averages over ten verified signatures per size. Setup generation, KZG initialization, and data generation are outside the timers.

7.4 Freshness and MPC resistance

The individual knowledge properties of our beholder signature inherently rely on the assumed bound on the number of slow queries that an adversary can perform during the online phase, i.e., after the beacon is

File size [KiB]	d	Verification time [ms]
16	12	8.079
32	13	7.818
64	14	7.676
128	15	8.184
256	16	8.339
512	17	8.763
1024	18	9.222
2048	19	9.489

Table 2. Mean verification times for the same signatures as in Tab. 1.

revealed (see Sect. 4.1). Conversely, one might want to choose the scheme's parameters based on the desired *freshness window*, i.e., the time interval during which it is considered infeasible to compute the beholder signature within MPC. Since we only assume MPC-hardness of the hash function H, we take a conservative approach and define the freshness window as the time required to perform the slow queries using MPC, while neglecting all other components of signing. In other words, the freshness-window estimates constitute lower bounds on the MPC signing time, conditional on the state-of-the-art MPC techniques for computing H.

More concretely, let a be the number of subadversaries and s their total number of online slow queries. Rearranging the accounting condition (2) gives the largest budget against which the scheme remains secure:

$$s_{\max} = \left\lfloor 2^{d-5}\rho - \rho a \right\rfloor .$$

We use $N \geq \kappa$, $m = 8$, and $\gamma = 1/2$, and require both $s_{\max} \geq 0$ and the data-coverage condition $2^{d-4}\rho \geq 4N$. Let R_{slow} be the assumed per-second rate at which the distributed adversary can perform slow queries, including any gains from preprocessing, batching, or parallelism. A window of length Δ seconds stays within the budget when

$$\Delta \leq \Delta_{\max} := \frac{s_{\max}}{R_{\text{slow}}}.$$

We present the resulting values of $\Delta_{\max}$ in Tab. 1. There, we assume $a = 2$ and an aggregate rate of $R_{\text{slow}} = 20$ queries per second. Furthermore, we consider the *minimal* value of d satisfying the data-coverage condition; longer freshness windows can be obtained by increasing d at the expense of proportionally increasing the signing time. Given the state-of-the-art overhead of evaluating SHA-2 in MPC [32], we assume an MPC slowdown factor $h = 10^5$ compared to native CPU execution; with the reference native evaluation time of $\tau_{\text{fast}} = 500\,\text{ns}$, this gives $\tau_{\text{slow}} = h\tau_{\text{fast}} = 50\,\text{ms}$ per slow hash. For example, a 16 KiB data vector gives $d = 12$ and $s_{\max} = 1260$, hence an estimated window of 63 seconds.

7.5 Signature size

For the size calculation, we retain exactly $\lceil \rho/2 \rceil$ proved iterations and only the commitments for the remaining iterations. Each commitment occupies 48 bytes. A proved iteration adds a 4-byte challenge, a 32-byte response, and m data-opening pairs of 80 bytes each. The compact payload therefore occupies

$$48\rho + \lceil \rho/2 \rceil (36 + 80m) \quad \text{bytes.}$$

Tab. 3 gives these sizes for $m = 8$, excluding the beacon and serialization metadata. The benchmarked implementation retains every successful iteration, so its signatures can be larger.

Acknowledgments. We used generative AI for grammar, spell checking, and basic editing. This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85. This work has been partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant CRYPTOLAYER-101044770), and by the European Research Council under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

ρ	**10**	**13**	**16**	**19**	**22**	**25**	**28**	**31**	**34**
Size [B]	3,860	5,356	6,176	7,672	8,492	9,988	10,808	12,304	13,124

Table 3. Compact signature payload for Schnorr over BLS12-381, with $m = 8$ and exactly $\lceil \rho/2 \rceil$ proved iterations; excludes the beacon and serialization metadata.

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.: Strong secret sharing with snitching. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO (4). Lecture Notes in Computer Science, vol. 16003, pp. 295–327. Springer (2025), http://dblp.uni-trier.de/db/conf/crypto/crypto2025-4.html#BormetDFLM25
7. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Annual International Cryptology Conference. pp. 78–96. Springer (2006)
8. Chiang, J.H., David, B., Frederiksen, T.K., Mondal, A., Yeniaras, E.: Detecting rogue decryption in (threshold) encryption via self-incriminating proofs. IACR Cryptol. ePrint Arch. p. 794 (2024), https://eprint.iacr.org/2024/794
9. Ciszewski, W., Dziembowski, S., Faust, S., Lizurej, T., Mielniczuk, M.: On the pitfalls of modeling individual knowledge. IACR Cryptol. ePrint Arch. (2024)
10. Damgård, I.: On σ-protocols. https://www.cs.au.dk/~ivan/Sigma.pdf (2010), lecture notes for CPT 2010, v.2
11. 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)
12. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
13. 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
14. 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
15. 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
16. Evans, A., Mohnblatt, N., Angeris, G.: ZODA: Zero-overhead data availability. Cryptology ePrint Archive, Paper 2025/034 (2025), https://eprint.iacr.org/2025/034
17. Feist, D.: Proofs of custody. https://dankradfeist.de/ethereum/2021/09/30/proofs-of-custody.html (2021), (Accessed on 30/09/2021)

18. Feist, D., Khovratovich, D.: Fast amortized kzg proofs. Cryptology ePrint Archive (2023)
19. Fisch, B.: Poreps: Proofs of space on useful data. IACR Cryptol. ePrint Arch. p. 678 (2018), https://eprint.iacr.org/2018/678
20. 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
21. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Annual International Cryptology Conference. pp. 152–168. Springer (2005)
22. Fuchsbauer, G., Garimidi, P., Policharla, G.V., Resnick, M., Tas, E.N.: Non-delegatable commitments. Cryptology ePrint Archive, Paper 2025/2039 (2025), https://eprint.iacr.org/2025/2039
23. 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
24. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Annual International Cryptology Conference. pp. 581–612. Springer (2017)
25. 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
26. 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
27. 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)
28. 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
29. Kelkar, M., Ganesh, A., Partap, A., Bonneau, J., Weinberg, S.M.: Breaking omertà: On threshold cryptography, smart collusion, and whistleblowing. ACM CCS **2025**, 1582 (2025)
30. 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
31. 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
32. 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
33. 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/
34. 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
35. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**, 361–396 (2000)
36. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
37. 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)
38. 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]
39. 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