2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179470

Threshold BBS+ Signatures for Distributed Anonymous Credential Issuance

Jack Doerner
Technion
j@ckdoerner.net

Yashvanth Kondi
Aarhus University
ykondi@cs.au.dk

Eysa Lee
Northeastern University
lee.ey@northeastern.edu

abhi shelat
Northeastern University
abhi@neu.edu

LaKyah Tyner
Northeastern University
tyner.l@northeastern.edu

***Abstract*—We propose a secure multiparty signing protocol for the BBS+ signature scheme; in other words, an anonymous credential scheme with threshold issuance. We prove that due to the structure of the BBS+ signature, simply verifying the signature produced by an otherwise semi-honest protocol is sufficient to achieve composable security against a malicious adversary. Consequently, our protocol is extremely simple and efficient: it involves a single request from the client (who requires a signature) to the signing parties, two exchanges of messages among the signing parties, and finally a response to the client; in some deployment scenarios the concrete cost bottleneck may be the client's local verification of the signature that it receives. Furthermore, our protocol can be extended to support the strongest form of blind signing and to serve as a distributed evaluation protocol for the Dodis-Yampolskiy Oblivious VRF. We validate our efficiency claims by implementing and benchmarking our protocol.**

1. Introduction

An *anonymous credential* allows an issuer to delegate authority to some particular individual, such that the individual can use the issuer's delegated authority without revealing their own identity. The notion was originally introduced by Chaum [1], and has been refined by a long line of follow-up works [2], [3], [4], [5], [6], [7], [8]. Anonymous credentials satisfy two basic security properties: the first is *unlinkability*, which guarantees that no verifier can correlate multiple uses of the same credential (even under arbitrary collusion), and the second is *unforgeability*, which guarantees that no valid credential can be generated without the consent of the issuer. These properties are essential, but a number of additional properties have been defined and realized, such as keyed-verifiability [9], [10] and delegatability [11], [12], [13].

A common and conceptually simple way to construct an anonymous credential scheme is to combine a signature scheme with a zero-knowledge proof of knowledge of a signature satisfying some predicate [5], [14]. The credential itself is a signature under the issuer's public key on a message indicating what is authorized, and the individual user, who receives the credential, uses the zero-knowledge proof to authenticate to others without revealing any information about the credential other than that it satisfies some predicate. This basic configuration allows the credential-holder to be anonymous with respect to the credential validator, but gives no anonymity property with respect to credential *issuance*. Anonymity during issuance can be achieved by using *blind* signing protocols. Much effort has been put into developing efficient signature schemes that accommodate efficient zero-knowledge proofs of knowledge, and efficient blind signing protocols.

Credential issuers as a single point of failure. The weak point in a traditional anonymous credential system is the issuer, who must hold a secret signing key for the underlying signature scheme. If the issuer is corrupted and the secret is leaked to an adversary, then that adversary can produce valid credentials with any properties it desires. Due to the anonymous nature of their use, such credentials are inherently difficult to revoke, and due to the primary use-case of anonymous credentials in governing access and granting authority, the consequences of such a leak are often extremely high. This risk can be mitigated by securely distributing the issuance authority across multiple servers (controlled by one entity, or many) in such a way that many or all of the issuing servers must be corrupted in order for the adversary to gain the power of forgery.

When an anonymous credential comprises a signature scheme plus a zero-knowledge proof of knowledge of a signature, distributing the issuing authority is as simple as replacing the issuer and its signing function with an ideal functionality that computes the *same* signing function when queried by the servers among which issuing authority is to be delegated. If this ideal functionality is then realized by a threshold signing protocol (with a threshold t) that has security against malicious adversaries *under composition*, then we can be certain that the resulting scheme has exactly the same security properties when up to $t-1$ issuers are corrupt as the original one did when the single issuer was honest. Due to the composable nature of the signing protocol, no properties of the credential need to be re-proven; the signing protocol can simply be dropped into any existing anonymous credential scheme that uses the same kind of signature. This, then, is the focus of the present paper: to composably thresholdize the signature scheme underlying an anonymous credential. Specifically, we choose the well-known BBS+ signature scheme.

BBS+ Signatures. The BBS+ signature scheme was introduced by Au, Susilo, and Mu [15] and derives its name from the group signature scheme of Boneh, Boyen, and Shacham [16], which served as an inspiration. BBS+ allows vectors of messages to be signed at once, and the size of the resulting signature depends upon the security parameter, but not the number of messages signed. The scheme also supports efficient zero knowledge proofs of knowledge of a signature that reveal elements of the message vector selectively; this feature allows it to serve as a flexible anonymous credential. Beyond this, BBS+ signatures have served as the basis for many other privacy-preserving protocols, such as Direct Anonymous Attestation (DAA) [17], [18], k-Times Anonymous Authentication [15], and blacklistable anonymous credentials [19]. Of particular note is the Enhanced Privacy ID (EPID) of Brickell and Li [17], which is deployed in Intel's SGX framework[1]. There is also an ongoing effort by the Internet Research Task Force (IRTF) to standardize BBS+ [20], which has broad industry support via a consortium known as the Decentralized Identity Foundation.[2]

The main difficulty. The BBS+ scheme uses a bilinear pairing to verify a simple relation in the exponent. The signing operation requires computing the following group element (stated using additive elliptic curve notation):

$$A := \frac{G_1 + s \cdot H_1 + \sum_{i \in [\ell]} m_i \cdot H_{i+1}}{x + e}$$

where x is the secret signing key, and m_i is part if the input message, and e and s are signing nonces. In order to thresholdize BBS+, this equation must be computed given a secret sharing of x. The main difficulty is that the signing operation involves computing $1/(x+e)$: the inverse of a secret value, modulo the order of the bilinear group. This must be done, and then the final signature computed, with security against a malicious adversary.

1.1. Securely Distributing Anonymous Credentials

We can frame the task of distributed key management for anonymous credentials as an instance of secure multiparty computation under carefully chosen constraints of interaction and statefulness. Our system will involve a fixed number of signing servers (i.e. the issuers), who secret-share the key among themselves, and many clients (who might alias the servers), to whom credentials must be issued. It is important to note that the clients are transient. That is, they are not fixed members of the protocol, and are expected to interact only minimally (and never with one another), expend few computational or network resources, and keep no state between signing sessions (or, if possible, even within signing sessions). In addition, unlike personal-scale decentralization (as relevant for cryptocurrency custody) where one might want to hide the fact that signing is distributed from outside observers, full transparency is desirable in the setting of credential-issuance, and so we assume that clients are able to connect to the issuing servers individually. A client initiates a signing request by sending a vector of messages to the servers as its input, and the servers run a multiparty computation among themselves and return an output to the client. We wish to maximize the throughput of the servers, since they may be issuing credentials to many clients simultaneously. This goal tends to coincide with minimizing the latency of the servers' responses, from the client's point of view. In order to achieve it, we are willing to incur client-side computational costs, so long as the total issuance latency observed by the client remains reasonable concretely.

Ideally, we would like to avoid an elaborate stateful protocol, and instead limit the interaction of the servers to two exchanges of messages. That is, each server sends a message to every other server, and then each server replies to the messages it receives. We refer to this pattern as a *round-trip* of communication,[3] and consider a single round-trip per issued credential to be reasonable, as it is also the amount of communication required in order to keep track of logistical information that supports the secure computation; for example it takes one round-trip interaction to coordinate session IDs, which are important in the Universal Composability framework [21], random oracle prefixes, logs of issued credentials, and to establish consensus on whether the client should be issued a credential at all. Our communication model is illustrated in Figure 1.

We would also like to avoid the so-called *preprocessing* model [22] in which a finite amount of correlated state is produced *offline* and consumed *online*. The preprocessing model creates a risk of state reuse, and requires a periodic replenishment of the correlated state.

Which secure computation paradigm? The most common techniques for secure computation fall into a few broad paradigms: constant-round protocols based on Garbled Circuits [23], [24], [25], arithmetic MPC systems where round complexity depends on the multiplicative depth of the circuit [26], [27], [28], and, recently, Pseudorandom-Correlation-based schemes [29], [30]. The BBS+ scheme is defined over a large finite field. The garbled circuit approach incurs substantial overhead for arithmetic circuits over large fields and so we do not pursue it further. While the Pseudorandom Correlation paradigm is promising, known constructions are either in the preprocessing model [29] or require non-standard assumptions or heavy machinery [30]. The arithmetic MPC approach is known to be efficient in terms of computation and bandwidth for large fields [28], and in our setting it does not induce too many rounds of interaction, because the multiplicative depth of the BBS+ signature algorithm is 1. As with most protocols in the arithmetic paradigm, ours is based on linear secret sharing.

When using a linear secret sharing scheme, non-linear operations on secret data are typically expensive to securely compute. In our case, we must invert a function of the secret key for each signature. Bar-Ilan and Beaver [31] provided an elegant template for solving this problem, which uses only

1. See https://api.portal.trustedservices.intel.com/EPID-attestation.
2. See https://identity.foundation/

3. Note that a *round-trip* as we use it here is distinct from the notion of a *round*, which typically refers to a single exchange of messages.

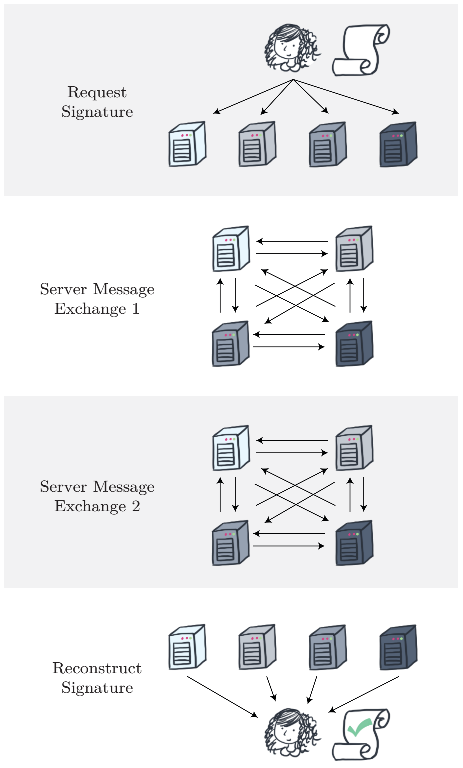

Figure 1: We consider a client (i.e. user) requesting a signature from a group of signing servers (i.e. credential issuers). The servers execute a protocol requiring two exchanges of messages before sending responses to the client. From these responses the client is able to reconstruct a signature.

a single secure multiplication on secret inputs. However, achieving security against a malicious adversary corrupting the majority of parties is nontrivial.

Lessons From Threshold ECDSA. The ECDSA signing algorithm has a non-linear structure that is similar in spirit to the BBS+ signing algorithm. In particular, both algorithms work over similarly-sized elliptic curve groups, and can trace their non-linearity to an inversion of a secret value. Multiparty ECDSA signing protocols have recently seen a surge in interest, motivated by key management concerns similar to those previously outlined. We refer the reader to Aumasson et al. [32] for a full survey of threshold ECDSA schemes, and we highlight below the key lessons that can be applied to our problem:

1. **Computational resources are the bottleneck.** The most sophisticated machinery required by threshold ECDSA is the secure multiplication for the inversion protocol [31], which is typically instantiated via either Additively Homomorphic Encryption (AHE) [33], [34], [35], or Oblivious Transfer (OT) [36], [37], [38]. The trade-off in concrete costs between these two approaches is roughly that AHE requires less bandwidth, whereas OT is computationally lightweight. In our setting, the secure multiplication protocol is assumed to be run by relatively well-connected servers (conservatively, with gigabit connections), and so we opt for the OT-based approach. This decision is supported by the work of Dalskov et al. [38] who investigated maximizing throughput in the context of DNSSEC.
2. **Leverage the structure of the problem.** Achieving malicious security for threshold ECDSA requires a multiplication protocol with malicious security, and also a consistency checking mechanism to ensure the various inputs and outputs from the multiplier are not altered as the signature is assembled. This mechanism typically comprises some combination of zero-knowledge proofs, SPDZ-style MACs [39], and equality checks in the ECDSA curve group. The works of Dalskov et al. [38] and Smart and Alaoui [40] show how to implement consistency checking for any secure computation within the ‘arithmetic black box’ framework [28] over an elliptic curve. However, their approach involves considerable computation and bandwidth overhead, and several rounds of interaction in order to generate and validate SPDZ-style MACs. On the other hand, Doerner et al. [36], [37] were able to avoid the costs of the generic approach by checking a few relations in the ECDSA curve group, and showed that subverting the checks implied forging signatures or breaking standard assumptions in the same group. Like Doerner et al., we avoid the generic approach in this work, and study how to exploit the structure of the signature itself in order to verify consistency.

With the constraints of the problem and an understanding of potential solutions in place, we are ready to describe our approach.

1.2. Our Techniques

We make use of the single-round-trip OT-based multiplier developed by Doerner et al. [36], as it achieves our desired interaction complexity, induces low computational burden, and is instantiable from the same qSDH assumption that BBS+ relies upon. We leave as an open question how one could use the more recent OT multiplier of Haitner et al. [41] in this context; their construction realizes a ‘weak’ multiplication functionality that could be sufficient, although as written their protocol requires three rounds. Their realization of the fully secure multiplication functionality induces dozens of scalar operations in an elliptic curve per invocation, which could be a computation bottleneck, especially when pairing-friendly curves (such as those required for BBS+) are used.

The signing protocol that we construct from this multiplier requires no additional rounds and only minimal additional interaction among the servers. We prove our protocol secure in the Universal Composability framework of Canetti [21], with respect to a straightforward ideal functionality that simply executes the BBS+ credential issuer’s algorithm internally when the issuing servers agree that a signature should be generated. Because the functionality simply runs the signing scheme, it serves as an intuitive drop-in replacement for centralized credential issuers. Moreover, the composable security guarantee enables credential requests to come in any order and spawn independent concurrent instances. We formalize our security notion as Functionality 3.1.

Protocol Template. A BBS+ signature consists of a triple (A, e, s), such that $A \in \mathbb{G}_1$ is a point on an elliptic curve that supports pairings and e and s are values from the finite field defined by the order of that curve. When a client sends a signing request to a set of servers, they engage in a protocol to generate e and s, and *shares* A_i of A. These values are then communicated by the servers individually to the client, who assembles A from the shares and verifies that (A, e, s) is indeed a valid signature. Though A is a point on an elliptic curve, each share A_i comprises both a curve point R_i and an element from the curve-order field u_i, and the reconstruction operation for A is defined to be $A \coloneqq \sum_i R_i / (\sum_i u_i)$. In order to sample such a sharing of A, the servers sample a uniform r in the curve-order group, in the form of secret shares r_i. From this they compute secret shares u_i of $u = r \cdot (x + e)$. If B is a public value that both the servers and clients can derive from the messages and s, then setting $R_i = r_i \cdot B$ produces the share $A_i = (R_i, u_i)$ of the value A defined as above. This is essentially a version of the Bar-Ilan and Beaver secure inversion technique [31]. The novelty and difficulty lie in ensuring that no malicious adversary cheats in this framework.

Verifying Consistency. Assuming that the multiplier is ideally secure (formally, in the $\mathcal{F}_{\mathsf{Mul2P}}$ hybrid model), we show that to achieve security against a malicious adversary, it suffices for the client to check if the (A, e, s) value is indeed a valid credential. While it is folklore that the consistency of a multiparty computation protocol that computes a "self-verifying" object like a digital signature could be validated simply by checking the signature, proving that no information is leaked in the event of a malformed signature is subtle. In particular, these types of folklore arguments often miss the potential for *selective failure* attacks in which a cheating adversary can slowly learn about the other parties' keys by inducing failures that are correlated with the secrets of the other parties. Defending against such attacks often requires elaborate zero-knowledge proof techniques.

In contrast, at a high level, we observe that the shares r_i serve as linear MAC keys, and reconstruction involves implicitly checking the MAC against A, which is fixed by e, s, and the public key. We show that if a server cheats and passes this correctness check, then it has effectively forged a signature (but in this case, the output signature is correct, the protocol does not abort, and the adversary learns nothing forbidden). We contrast this implicit MAC with the *explicit* MAC used by the generic MPC approach [40], [38]—computing an explicit MAC induces a computation and communication overhead factor of roughly two, and validating and using it to check correctness requires several extra rounds of interaction.

Conceptually, this allows us to place BBS+ signatures in between Schnorr and ECDSA in terms of complexity of decentralization. In particular, Schnorr signatures are known to be straightforward to decentralize even with UC security [42], requiring only commitments and proofs of knowledge. ECDSA requires multiple invocations of secure multiplication, along with as many accompanying consistency checks via implicit MACs, some of which are computational [37]. In particular, all of the modern threshold protocols for ECDSA require extra work to ensure that the inputs to the different multipliers are consistent. Our protocol to decentralize BBS+ interpolates an intermediate decentralization complexity between ECDSA and Schnorr, as it requires only a single invocation of secure multiplication, and thus a corresponding information-theoretic implicit MAC check with no need for additional consistency checks.

Extensions. Okamoto's signature scheme [43] can be viewed as a variant of BBS+, and is therefore thresholdizable via our scheme. As we discuss in Section 5, our techniques can also be used to distribute the computation of the Dodis and Yampolskiy Verifiable Random Function [44]. In addition, since the state that our servers are required to maintain comprises additive secret key shares and base OTs for the OT extension [45] used by the multiplier, we can use the proactivization scheme of Kondi et al. [46] to refresh the state of the system and defend it against mobile attackers [47].

1.3. Prior Works

Dodis and Yampolskiy [44] proposed a verifiable random function (VRF) of the form $F_x(e) \mapsto \mathsf{e}(G_1, G_2)/(x+e)$ where the proof of correct evaluation is of the form $\pi = G_1/(x+e)$. This structure is very similar to the BBS+ signature scheme. Dodis and Yampolskiy themselves proposed that their VRF could be evaluated via the inversion trick of Bar-Ilan and Beaver [31]; our protocol can be viewed as the minimal way to add malicious security to their distributed VRF construction. Moreover, our scheme can be extended to make such a threshold VRF *oblivious*. We discuss this further in Section 5.

The problem of thresholdizing the BBS+ signature scheme was previously taken up by Goldfeder, Gennaro, and Ithurburn [48]. That solution, like ours (and the Dodis-Yampolskiy scheme) begins with the inversion protocol of Bar-Ilan and Beaver. Our scheme, however, is distinct in several important regards. That work provides a monolithic proof of standalone security, whereas we provide a full modular proof of composable security in the UC paradigm. In particular, our scheme is based upon an ideal multiplication functionality, which is realizable in two rounds from the same assumption as the BBS+ signature scheme. Theirs, on the other hand, hardcodes a multiplication strategy based upon Paillier encryption. This potentially degrades efficiency, because it is unclear whether their scheme can be adapted to require only a single round-trip of communication as our does, and because securing Paillier-based multipliers requires zero-knowledge range proofs that are far more costly than any other component of the protocol. It also degrades security, because it means that their protocol relies upon the Strong RSA assumption (which is entirely unrelated to the underlying signature scheme). Even more troublingly, the multiplication techniques used in [48] have been shown explicitly to be insecure in the context of threshold ECDSA signing [49], [50]; the impact of this on the BBS+ protocol is at present unclear. Finally, unlike the [48] scheme, our scheme does not require A to be revealed to the simulator in order to simulate the protocol when the client is honest. Achieving this requires a somewhat subtle analysis, but it opens the door to a fully-blind signing extension, and to applying our protocol to the threshold oblivious VRF problem as previously mentioned. Due to all of these reasons and the simplicity of our protocol, we are also able to provide an implementation with concrete benchmarks in Section 7.

An anonymous credential scheme supporting threshold issuance was also given by Sonnino, Al-Bassam, Bano, Meiklejohn, and Danezis [51]. This scheme, Coconut, is primarily based upon the signature scheme of Pointcheval and Sanders [52] (PS signatures), which base their security on an interactive assumption similar but not equivalent to LRSW [4]. In terms of credential-showing efficiency, Coconut and our work are in similar in that they both require proving and verifying a 2-clause non-interactive zero-knowledge proof of knowledge. A follow-up paper by Rial and Piotrowska [53] (RP-Coconut), however, identifies security problems with the proof sketch of Sonnino *et al.* [51] and provides a patch. In order to prove unforgeability, RP-Coconut requires increasing the size of the public key, bringing up the size to ℓ elements of $\mathbb{G}_1$ plus $\ell+1$ elements of $\mathbb{G}_2$. This is nearly double the public key size of Coconut, which required $\ell+1$ elements of $\mathbb{G}_2$. In comparison, the public key for our scheme only requires 1 element of $\mathbb{G}_1$ and 1 element of $\mathbb{G}_2$. Additionally, BBS+ signatures are compatible with all group types, while PS signatures specifically require type-3 pairings. As Pointcheval and Sanders [52] write, the existence of an efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ would make their signature scheme "totally insecure". Finally, it is worth noting that Rial and Piotrowska provide a *sequentially*-secure simulation-based proof for RP-Coconut. Unlike proofs in the UC-model, sequential security makes no guarantees for concurrent or parallel executions of the protocol.

2. Preliminaries

Notation. We use λ to denote the (computational) security parameter and n to denote the number of parties. The symbols $\approx_{\mathsf{c}}$ and $\approx_{\mathsf{s}}$ denote computational and statistical indistinguishability, respectively, with respect to λ. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ denote three groups of prime order p, such that $|p| = \kappa$, and we represent operations over these groups additively. By convention, variables representing group elements are capitalized, and the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ are G_1 and G_2 respectively. Single-letter variables are set in *italic* font, multi-letter variables and function names are set in sans-serif, and string literals are set in `slab-serif`. Bold variables represent vectors of subscripted elements, so that $\mathbf{x} = \{x_1, x_2, x_3\}$ in a context where the latter three variables are defined, and we use $[n]$ to denote the vector of integers $\{1, \ldots, n\}$ and $\|$ to denote concatenation.

Bilinear Groups. A bilinear group (or *pairing* group) is a trio of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T})$ with an efficient map (or pairing) operation $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$, such that for any $x, \in \mathbb{Z}_p$ and $y \in \mathbb{Z}_p$, $\mathsf{e}(x \cdot G_1, y \cdot G_2) = x \cdot y \cdot \mathsf{e}(G_1, G_2)$. We define $\mathsf{BilinGen}$ to be an efficient algorithm $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, p) = \mathcal{G} \leftarrow \mathsf{BilinGen}(\lambda)$, which samples a description $\mathcal{G}$ of the group (with λ bits of security). There are three types of pairings [54]: type-1, in which $\mathbb{G}_1 = \mathbb{G}_2$; type-2, in which $\mathbb{G}_1 \neq \mathbb{G}_2$ and there exists an efficient isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$; and type-3, in which $\mathbb{G}_1 \neq \mathbb{G}_2$ and there does not exist an efficient isomorphism ψ.

2.1. The BBS+ Signature Scheme

The BBS+ signature scheme uses bilinear groups to produce a signature for a vector of ℓ messages. Its algorithms are as follows.

Algorithm 2.1. $\mathsf{BBS{+}Gen}(\mathcal{G}, \ell)$

```
1. Let $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, p) := \mathcal{G}$.
2. Sample a vector of $\ell+1$ random group elements $\mathbf{H} \leftarrow \mathbb{G}_1^{\ell+1}$.
3. Uniformly choose secret key $x \leftarrow \mathbb{Z}_p^*$.
4. Calculate $X := x \cdot G_2$.
5. Set $\mathsf{sk} := (\mathbf{H}, x)$ and $\mathsf{pk} := (\mathbf{H}, X)$.
6. Output $(\mathsf{sk}, \mathsf{pk})$.
```

Algorithm 2.2. $\mathsf{BBS{+}Sign}(\mathsf{sk}, \mathbf{m} \in \mathbb{Z}_p^\ell)$

```
1. Parse $\mathsf{sk}$ as $(\mathbf{H}, x)$.
2. Uniformly sample nonces $e \leftarrow \mathbb{Z}_p$ and $s \leftarrow \mathbb{Z}_p$.
3. Compute
       $$A := \frac{G_1 + s \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1}}{x + e}$$
4. Output signature $\sigma := (A, e, s)$.
```

Algorithm 2.3. $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, \sigma)$

```
1. Parse $\mathsf{pk}$ as $(\mathbf{H}, X)$ and $\sigma$ as $(A, e, s)$.
2. Check the following:
       $$\mathsf{e}(A,\ X + e \cdot G_2) = \mathsf{e}(G_1 + s \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1},\ G_2)$$
   Output 1 if and only if the equality holds.
```

Au *et al.* [15] introduced BBS+ and proved it secure for type-1 and type-2 pairings, using the original "conference version" of the qSDH assumption [55].

Lemma 2.4 ([15]). *The BBS+ signature scheme is existentially unforgeable against adaptive chosen messages under* *the conference version of the qSDH assumption for type-1 and type-2 pairings.*

BBS+ was later proved secure without any assumptions about the existence (or non-existence) of an isomorphism for type-3 pairings by Camenisch *et al.* [18], under the updated "journal version" of the qSDH assumption [56].

Lemma 2.5 ([18])**.** *The BBS+ signature scheme is existentially unforgeable against adaptive chosen messages under the journal version of the qSDH assumption for type-3 pairings.*

Note that while the BBS+ signature scheme requires the qSDH assumption to achieve unforgeability, and while oblivious transfer can also be securely instantiated under the same assumption, our protocols are secure in the "OT-hybrid model", and thus do *not* specifically require qSDH or any computational assumption.

Comparison to Okamoto Signatures Okamoto's signature scheme [43] was originally introduced in the context of constructing blind signatures. As Au *et al.* [15] observed previously, BBS+ can be viewed as an extension of Okamoto signatures for signing blocks of messages. Apart from the number of messages signed, the schemes mainly differ in their proofs of security. The original conference version of Okamoto's paper introduced the 2-variable strong Diffie-Hellman (2SDH) assumption, and proved security under a variant of this assumption. A later version of the paper [57] revised the proof to achieve security under the conference version of qSDH. Both versions of this proof rely on an isomorphism between the groups. Okamoto signatures are known to be strongly existentially unforgeable, whereas Au *et al.* [15] and Camenisch *et al.* [18] claim only standard unforgeability for BBS+. Our techniques can easily be adapted to thresholdize the Okamoto scheme.

2.2. Blind Signatures

A blind signature protocol allows a signer (who holds the secret key) to sign a message belonging to another party, without learning the contents of the message. In *weakly-blind* signing schemes, only the message is hidden, whereas in *strongly-blind* schemes, the resulting signature is also hidden from the signer, so that it cannot be used to identify the client later. In this work, we achieve weak *partially*-blind signing. This is a variant of weakly-blind signing in which the message is hidden from the signer, but the signer receives a proof that the message satisfies some predicate. In this way, an arbitrary signing policy can be enforced, even though the signer signs blindly. In the threshold context, we will allow the client (who requests and receives the signature) to prove a *different* predicate to each signer. Note that weak partial-blindness implies weak blindness: the client need only omit the predicate.

2.3. Universal Composability

We formalize our protocols and prove them secure in the Universal Composability (UC) framework, using standard UC notation. We refer the reader to Canetti [21] for a full description of the model, and give a brief overview here.

In the UC framework, the *real-world* experiment involves n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ that execute a protocol π, an adversary $\mathcal{A}$ that can corrupt a subset of the parties, and an environment $\mathcal{Z}$ that is initialized with an advice string z. All entities are initialized with the security parameter λ and with a random tape. The environment activates the parties involved in π, chooses their inputs and receives their outputs, and communicates with the adversary $\mathcal{A}$, who may may instruct the corrupted parties to deviate from π arbitrarily. In this work, we consider only *static* adversaries, who announce their corruptions at the beginning of the experiment. The real-world experiment completes when $\mathcal{Z}$ stops activating parties and outputs a decision bit. Let $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$ denote the random variable representing the output of the experiment.

The *ideal-world* experiment involves n dummy parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, an ideal functionality $\mathcal{F}$, an ideal-world adversary $\mathcal{S}$ (the simulator), and an environment $\mathcal{Z}$. The dummy parties forward any message received from $\mathcal{Z}$ to $\mathcal{F}$ and vice versa. The simulator can corrupt a subset of the dummy parties and interact with $\mathcal{F}$ on their behalf; in addition, $\mathcal{S}$ can communicate directly with $\mathcal{F}$ according to its specification. The environment and the simulator can interact throughout the experiment, and the goal of the simulator is to trick the environment into believing it is running in the real experiment. The ideal-world experiment completes when $\mathcal{Z}$ stops activating parties and outputs a decision bit. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$ denote the random variable representing the output of the experiment.

A protocol π *UC-realizes* a functionality $\mathcal{F}$ if for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$,

$$\{\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}^+, z\in\{0,1\}^{\mathsf{poly}(\lambda)}} \approx_{\mathsf{c}} \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}^+, z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$$

3. Functionalities

We give the ideal functionality that we intend our protocol to realize.

Functionality 3.1. $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G})$**.**

This functionality interacts with n fixed parties denoted by $\mathcal{P}_1, \ldots, \mathcal{P}_n$, an a-priori unspecified number of transient clients, all of them denoted by $\mathcal{C}$, and with an ideal adversary $\mathcal{S}$. Clients may be aliases of any of the parties. The set of corrupt parties is indexed by $\mathbf{P}^*$. The functionality is also parameterized by a threshold $t \leq n$ and the description of a bilinear group, $\mathcal{G}$.

Key Generation: On receiving $(\texttt{key-gen}, \mathsf{sid}, \ell)$ from every party $\mathcal{P}_i$ for $i \in [n]$, where $\ell \in \mathbb{N}^+$ is agreed-upon and sid is agreed-upon and fresh, if there exists no record of the form $(\texttt{key}, \mathsf{sid}, *, *)$ in memory, then sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BBS+Gen}(\mathcal{G}, \ell)$, store $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk}, \ell)$ in memory, and send $(\texttt{pub-key}, \mathsf{sid}, \mathsf{pk})$ to every party $\mathcal{P}_i$ for $i \in [n]$ as adversarially-delayed private output.

Signing: Upon receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from $\mathcal{C}$, where $\mathbf{m} \in \mathbb{Z}_p^\ell$, $\mathbf{J} \subseteq [n]$, $|\mathbf{J}| = t$, and sigid is fresh, if a record of the form $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk}, \ell)$ exists in memory, then send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to every party $\mathcal{P}_j$ for $j \in \mathbf{J}$. On receiving $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ from every $\mathcal{P}_j$ for $j \in \mathbf{J}$, compute $(A, e, s) \leftarrow \mathsf{BBS+Sign}(\mathsf{sk}, \mathbf{m})$, send $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, e, s)$ to $\mathcal{S}$ and send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk}, \mathbf{J})$ to $\mathcal{C}$ as adversarially-delayed private output. If any $\mathcal{P}_j$ instead sends $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$, then send $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{C}$ as adversarially-delayed private output.

Weak Partially-Blind Signing: Upon receiving $(\texttt{wb-sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J}, \boldsymbol{\phi})$ from $\mathcal{C}$, where $\mathbf{m} \in \mathbb{Z}_p^\ell$, $\mathbf{J} \subseteq [n]$, $\boldsymbol{\phi}$ is a vector of descriptions of predicates on $\mathbf{m}$, $|\mathbf{J}| = |\boldsymbol{\phi}| = t$, and sigid is fresh, if a record of the form $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk}, \ell)$ exists in memory, and if $\phi_j(\mathbf{m}) = 1$ for every $j \in \mathbf{J} \setminus \mathbf{P}^*$, then send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \phi_j, \mathbf{J})$ to every party $\mathcal{P}_j$ for $j \in \mathbf{J}$. On receiving $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ from every $\mathcal{P}_j$ for $j \in \mathbf{J}$, compute $(A, e, s) \leftarrow \mathsf{BBS+Sign}(\mathsf{sk}, \mathbf{m})$, send $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, e)$ to $\mathcal{S}$, and send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk}, \mathbf{J})$ to $\mathcal{C}$ as adversarially-delayed private output. If any $\mathcal{P}_j$ instead sends $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$, then send $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{C}$ as adversarially-delayed private output.

Note that the weak partially-blind signing interface of the foregoing interface can be converted into a *strong* partially-blind signing interface simply by removing the leakage to the adversary. We discuss how to realize such a modified functionality in Section 5.

3.1. Building Blocks

Now we will define the building blocks of our protocol. We begin with our communication model: every pair of parties can communicate via an authenticated channel, and we also assume the existence of a broadcast channel. Formally, the protocols are defined in the $(\mathcal{F}_{\mathsf{Auth}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model (see [21], [58]). We leave this implicit in their descriptions. Since we desire only to achieve security with abort, our broadcast channel can be realized via the echo-broadcast technique [59]. Specifically, the parties send broadcast messages optimistically over point-to-point channels, and at the end, every party hashes the entire transcript of broadcast messages and sends the digest to all other parties. If the digests do not agree, the parties abort.

Standard functionalities. The parties make use of standard commitment, zero-knowledge, and committed-zero-knowledge functionalities; $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{ZK}}$, $\mathcal{F}_{\mathsf{Com\text{-}ZK}}$ respectively. We specify the commitment and zero knowledge functionalities to work in a broadcast fashion, but they are otherwise similar to the standard versions [58]. The parties also use a functionality $\mathcal{F}_{\mathsf{Zero}}$ that produces secret sharings of zero in a particular group. These functionalities are given in Appendix A.

$\mathcal{F}_{\mathsf{Com}}$ and $\mathcal{F}_{\mathsf{Zero}}$ can both be realized via simple folkloric methods. $\mathcal{F}_{\mathsf{Com}}$ is realized in the Random Oracle model simply by feeding the value to be committed into the random oracle, along with a random salt of security parameter length, and then transmitting the oracle's output as the commitment, and the salt along with the original value as the opening. $\mathcal{F}_{\mathsf{Zero}}$ can be realized in the following way: each pair of parties commits and decommits a pair of λ-bit seeds to one another, then sums the pair to form a single shared seed. When $\mathcal{F}_{\mathsf{Zero}}$ is invoked, each pair of parties evaluates the random oracle on their shared seed concatenated with the next index in sequence. Each party then computes a single output share for itself by accumulating the random oracle outputs: it subtracts oracle outputs for pairings in which it is lower-indexed, and adds oracle outputs for pairings in which it is higher-indexed.

The zero-knowledge and committed-zero-knowledge functionalities will be used with the standard discrete logarithm relation

$$R_{\mathsf{DL}} = \{((X, B), x) \ : \ X = x \cdot B\}$$

and in addition, in the context of partially-blind signing, they will be used with the conjunction of an arbitrary predicate and accumulated-discrete-logarithm

$$R_{\mathsf{DL}\wedge\phi} = \Big\{((X, B_1, \ldots, B_\ell), (x_1, \ldots, x_\ell)) : \\ X = \sum_{i \in \ell} x_i \cdot B_i \ \wedge \ \phi(x_1, \ldots, x_\ell) = 1\Big\}$$

$\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ can be realized via the Fischlin transform [60] applied to the Schnorr protocol [61]. The realization of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}\wedge\phi}}$ depends upon the predicate ϕ, but in simple cases, such as selectively checking equality with known values, the cost is no more than that of proving knowledge of ℓ discrete logarithms. $\mathcal{F}_{\mathsf{Com\text{-}ZK}}$ can be realized similarly to $\mathcal{F}_{\mathsf{ZK}}$, but with the addition of a commitment and decommitment via $\mathcal{F}_{\mathsf{Com}}$.

Multiplication functionality. The main building block of our protocol is two-party multiplication. Specifically, we require a functionality that enables Alice and Bob, who have a and b respectively, to learn c and d respectively, such that $c + d = a \cdot b$. This functionality is given in Appendix A.

There are many techniques in the literature of multiparty computation for realizing multiplication functionalities, but for the sake of achieving our desired efficiency targets, we choose to realize $\mathcal{F}_{\mathsf{Mul2P}}(p)$ via the two-round protocol of Doerner et al. [36]. Their protocol is based upon Oblivious

Transfer (OT) [62], and can be seen as a malicious extension of the classic semi-honest distributed multiplication technique of Gilboa [63]. Because their protocol is OT based, it can be based upon many assumptions, including the assumption that the computational Diffie-Hellman problem is hard in $\mathbb{G}_1$, which is implied by the qSDH assumption under which BBS+ itself is proven secure.

4. Threshold BBS+ Protocol

Before we give the formal description of our signing protocol, we give an overview of the subprotocols. An informal description of techniques is also given in Section 1.2.

Key Generation. Before servers can sign messages, they must first run a one-time setup phase to jointly generate keys. These parties begin by using standard techniques for sampling a Shamir secret sharing of a random value: each party samples a random polynomial and sends to each other a point on their polynomial. Each party $\mathcal{P}_i$ adds a point from their own polynomial with the sum of the points received and takes this as their share of the secret key x_i. To generate the public key corresponding to this secret key, parties perform a commit-and-release of $X_i := x_i \cdot G_2$ along with a proof of knowledge of discrete logarithm. Parties can compute $X := x \cdot G_2$ by interpolating a size t subset of X_i in the exponent. Malicious parties may however send malformed X_i, so parties check that all received X_i lie on the same degree-$(t-1)$ polynomial.[4]

Parties must also generate $\mathbf{H}$, which can be done using a standard commit-and-release of random $\mathbb{G}_1$ elements, which are then summed. This commit-and-release can be done in parallel with that of X_i so as to not increase round-count. Communication efficiency efficiency can be improved (and the public key can be compressed) by using a programmable random oracle to generate $H_2, \ldots, H_{\ell+1}$ from H_1. Oracle programming can be avoided if $H_2, \ldots, H_{\ell+1}$ are generated in the same way as H_1.

Signing. The client initiates the signing protocol with t servers by sending the messages $\mathbf{m}$ to be signed, and $\mathbf{J} \subseteq [n]$, the identities of the signing parties.

Suppose the set of t signing parties know a secret sharing $\mathbf{r}$ of a random value r, and that they know (uniformly sampled) values s and e. Suppose furthermore that the signing parties know a secret sharing $\mathbf{u}$ of the product $u = r \cdot (x+e)$. It is easy to see that signing party $\mathcal{P}_i$ can compute

$$R_i := r_i \cdot \left(G_1 + s \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1} \right)$$

and that if each party $\mathcal{P}_i$ sends s, e, R_i, and u_i to the client, then the client can compute

$$A := \frac{\sum_{i \in \mathbf{J}} R_i}{\sum_{i \in \mathbf{J}} u_i} = \frac{r \cdot \left(G_1 + s \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1} \right)}{r \cdot (x + e)}$$

4. There are many ways of implementing this check which do not require interpolating with *all* $\binom{n}{t}$ possible subsets of size t. For instance, it is sufficient check that subsets that cover $[n]$ interpolate to the same value.

which is a BBS+ signature on $\mathbf{m}$. Notice that the sum $\sum_{i \in \mathbf{J}} u_i$ information-theoretically hides x, and that the sum $\sum_{i \in \mathbf{J}} R_i$ reveals exactly what A reveals, given knowledge of $\mathbf{u}$. The task of the signing parties is thus to generate $\mathbf{r}$, $\mathbf{u}$, s, and e in only two message-exchanges.

s and e can both be sampled quite simply via commit-and-release coin tossing. The shares $\mathbf{r}$ are sampled locally, and then a two round multiplier is used to compute shares of the pairwise products $r_i \cdot \lambda_j^{\mathbf{J}}(0) \cdot x_j$ for all $(i,j) \in \mathbf{J} \times \mathbf{J}$.[5] Each party $\mathcal{P}_i$ then computes u_i as the sum of its shares of these pairwise products and $r_i \cdot e + \alpha_i$, where $\boldsymbol{\alpha}$ is a secret sharing of 0.[6] It is possible for the parties to sample $\boldsymbol{\alpha}$ noninteractively using well-known techniques.

Weak Partially-Blind Signing. To achieve weak partial-blindness, the client does not send $\mathbf{m}$ to the signing parties, but instead samples a masking nonce $s_0 \leftarrow \mathbb{Z}_p$ and computes $B' := s_0 \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1}$. The client sends B' to the signing parties instead of $\mathbf{m}$, along with a zero-knowledge proof of knowledge of s_0 with respect to B'. This proof is necessary in order to permit the simulator to extract the client's mask share s_0, but it can also be extended to allow the client to prove properties of its messages to the signing parties. The signing parties determine s as usual and construct $B := G_1 + s \cdot H_1 + B'$, and when the client receives the signature, it computes $s' := s_0 + s$, and takes the signature to be (A, e, s') instead of (A, e, s). This modification information-theoretically hides the messages from the signing parties (in the $\mathcal{F}_{\mathsf{ZK}}$-hybrid model), even if all of the signing parties are corrupt.

4.1. t-of-n Threshold Signing

Our protocol contains three subprotocols, corresponding to the three phases of $\mathcal{F}_{\mathsf{BBS+}}$. The first generates a public key and Shamir shares of the corresponding secret key. This subprotocol is a derivative of the protocol Doerner et al. [36], [37]. Thereafter we give protocols for plain signing and for weak partially-blind signing.

Protocol 4.1. $\pi_{\mathsf{BBS+}}(n, t, \mathcal{G})$.

This protocol runs among n fixed parties denoted by $\mathcal{P}_1, \ldots, \mathcal{P}_n$, an a-priori unspecified number of transient clients, all of them denoted by $\mathcal{C}$. Clients may be aliases of any of the parties. The parties (and clients) also have access to the ideal functionalities $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}}}$, $\mathcal{F}_{\mathsf{Zero}}(p)$ and $\mathcal{F}_{\mathsf{Mul2P}}(p)$. The protocol is parameterized by the threshold t and the description of a bilinear group, $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, p) = \mathcal{G}$. If at any point during this protocol, a functionality aborts, any party that observes

5. Recall that $\mathbf{x}$ is a *Shamir* secret sharing of x. The set of publicly-calculable Lagrange coefficients $\lambda_j^{\mathbf{J}}(0)$ for $j \in \mathbf{J}$ converts it into a t-party *additive* sharing. Also note that when $i = j$, local multiplication suffices.

6. The secret sharing of 0 rerandomizes the summed shares. This ensures that if the corrupt parties use incorrect inputs for some or all of the multiplication protocol instances, then the offset induced into u_h for some honest party $\mathcal{P}_h$ are independent of that that party's secret x_h.

the abort also aborts to the environment.

Key Generation: On receiving $(\texttt{key-gen}, \mathsf{sid})$ from $\mathcal{Z}$, where sid is fresh, each party $\mathcal{P}_i$ for $i \in [n]$ performs the following sequence of steps:

1. $\mathcal{P}_i$ samples a random degree $t-1$ polynomial $\hat{x}_i(\cdot)$ and sends $\hat{x}_i(j)$ to every other party $\mathcal{P}_j$ for $j \in [n] \setminus \{i\}$.
2. Upon receiving $\hat{x}_j(i)$ from every other $\mathcal{P}_j$, party $\mathcal{P}_i$ computes its share of the secret key as point $x_i := \sum\limits_{j\in[n]} \hat{x}_j(i) \bmod p$.
3. $\mathcal{P}_i$ computes $X_i := x_i \cdot G_2$ and sends $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, X_i, x_i)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.
4. $\mathcal{P}_i$ samples $\mathbf{D}_i \leftarrow \mathbb{G}_1^{\ell+1}$ and sends $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \mathbf{D}_i)$ to $\mathcal{F}_{\mathsf{Com}}$.
5. Upon being notified of all other parties' commitments to both $\mathcal{F}_{\mathsf{Com}}$ and $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, $\mathcal{P}_i$ sends $(\texttt{prove}, \mathsf{sid}\|\mathcal{P}_i)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ and $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{Com}}$.
6. On receiving $(\texttt{reject}, \mathsf{sid}\|\mathcal{P}_j)$ from $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ for any j, $\mathcal{P}_i$ aborts.
7. On receiving $(\texttt{accept}, \mathsf{sid}\|\mathcal{P}_j, X_j)$ from $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ for every $j \in [n] \setminus \{i\}$, $\mathcal{P}_i$ aborts if the exponents of each X_j do not lie on the same degree $t-1$ polynomial. That is, $\mathcal{P}_i$ aborts if there exists any sets $\mathbf{J}, \mathbf{J}' \subset [n]$ such that
$$\sum_{j\in\mathbf{J}} \lambda^{\mathbf{J}}_j(0) \cdot X_j \neq \sum_{j\in\mathbf{J}'} \lambda^{\mathbf{J}'}_j(0) \cdot X_j$$
where $\mathbf{J} \neq \mathbf{J}'$, $|\mathbf{J}| = |\mathbf{J}'| = t$, and $\lambda^{\mathbf{J}}_j(0)$ and $\lambda^{\mathbf{J}'}_j(0)$ are the Lagrange coefficients for interpolating with the set of parties indexed by $\mathbf{J}$ and $\mathbf{J}'$, respectively.
8. On receiving $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_j, \mathbf{D}_j)$ from $\mathcal{F}_{\mathsf{Com}}$ for every $j \in [n] \setminus \{i\}$, $\mathcal{P}_i$ computes
$$\mathbf{H} := \left\{ \sum_{j\in[n]} D_{j,k} \right\}_{k\in[\ell+1]}$$
$$X := \sum_{j\in\mathbf{J}} \lambda^{\mathbf{J}}_j(0) \cdot X_j$$
using any subset $\mathbf{J} \subseteq [n]$ where $|\mathbf{J}| = t$. Finally, $\mathcal{P}_i$ outputs $(\texttt{pub-key}, \mathsf{sid}, (\mathbf{H}, X))$ to $\mathcal{Z}$.[a]

Signing: This phase of the protocol is initiated by the client $\mathcal{C}$ when it receives $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from $\mathcal{Z}$. Clients may be transient or alias with the fixed parties. Note $\mathbf{J} \subseteq [n]$ and $|\mathbf{J}| = t$.

9. $\mathcal{C}$ sends $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to each party $\mathcal{P}_j$ for $j \in \mathbf{J}$.
10. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from some client $\mathcal{C}$, $\mathcal{P}_i$ samples $(e_i, s_i, r_i) \leftarrow \mathbb{Z}_p^3$ uniformly and sends $(\texttt{sample}, \mathsf{sid}, \mathbf{J})$ to $\mathcal{F}_{\mathsf{Zero}}(p)$. Upon receiving $(\texttt{zero-share}, \mathsf{sid}, \alpha_i)$ from $\mathcal{F}_{\mathsf{Zero}}(p)$, $\mathcal{P}_i$ computes the Lagrange coefficient $\lambda^{\mathbf{J}}_i(0)$ for interpolating the polynomial with the parties indexed by $\mathbf{J}$ and sends $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid}, \{\mathcal{P}_j\}_{j\in\mathbf{J}}, (e_i, s_i))$ to $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j, \lambda^{\mathbf{J}}_i(0) \cdot x_i + \alpha_i)$ to $\mathcal{F}_{\mathsf{Mul2P}}(p)$ for every $j \in \mathbf{J} \setminus \{i\}$.
11. Upon receiving $(\texttt{committed}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{bob-ready}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, \mathcal{P}_j, c_{i,j})$ from $\mathcal{F}_{\mathsf{Mul2P}}(p)$ from every $j \in \mathbf{J} \setminus \{i\}$, sends $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid})$ to $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{multiply}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, r_i)$ to $\mathcal{F}_{\mathsf{Mul2P}}(p)$ for every $j \in \mathbf{J} \setminus \{i\}$.
12. Upon receiving both $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid}, (e_j, s_j))$ from $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{product}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sigid}, d_{i,j})$ from $\mathcal{F}_{\mathsf{Mul2P}}(p)$ for every $j \in \mathbf{J} \setminus \{i\}$, $\mathcal{P}_i$ computes
$$e := \sum_{j\in[n]} e_j \bmod p$$
$$s := \sum_{j\in[n]} s_j \bmod p$$
$$B := G_1 + s \cdot H_1 + \sum_{k\in[\ell]} m_k \cdot H_{k+1}$$
$$R_i := r_i \cdot B$$
$$u_i := \begin{pmatrix} r_i \cdot \left(e + \lambda^{\mathbf{J}}_i(0) \cdot x_i\right) + \alpha_i \\ + \sum\limits_{j\in\mathbf{J}\setminus\{i\}} (c_{i,j} + d_{i,j}) \end{pmatrix} \bmod p$$
and sends $(\texttt{output-share}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i, \mathbf{J})$ to $\mathcal{C}$, and halts.
13. Upon receiving $(\texttt{output-share}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i, \mathbf{J})$ from every $\mathcal{P}_i$ for $i \in \mathbf{J}$, the client $\mathcal{C}$ aborts if any two parties disagree on the values of s, e, or pk. Otherwise, $\mathcal{C}$ parses pk as $(\mathbf{H}, X)$, computes B via the same equation as in Step 12, computes
$$A := \frac{\sum_{i\in\mathbf{J}} R_i}{\sum_{i\in\mathbf{J}} u_i}$$
and verifies that $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s)) = 1$. If so, then $\mathcal{C}$ outputs $(\texttt{output}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk})$ to $\mathcal{Z}$, and halts, but if the equality does not hold, then $\mathcal{C}$ aborts.

Weak Partially-Blind Signing: This phase of the protocol is initiated by the client $\mathcal{C}$ when it receives $(\texttt{wblind-sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from $\mathcal{Z}$. Clients may be transient or alias with the fixed parties. Note $\mathbf{J} \subseteq [n]$ and $|\mathbf{J}| = t$.

14. $\mathcal{C}$ samples $s_0 \leftarrow \mathbb{Z}_p$, computes
$$B' := s_0 \cdot H_1 + \sum_{k\in[\ell]} m_k \cdot H_{k+1}$$
and sends $(\texttt{prove}, \mathsf{sid}, \{\mathcal{P}_i\}_{i\in\mathbf{J}}, B', \{s_0, m_1, \ldots, m_\ell\})$ to $\mathcal{F}^{R_{\mathsf{DL}\wedge\phi}}_{\mathsf{ZK}}$. $\mathcal{C}$ sends $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, B', \mathbf{J})$ to each party $\mathcal{P}_j$ for $j \in \mathbf{J}$.
15. On receiving $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, B', \mathbf{J})$ from some client $\mathcal{C}$ and $(\texttt{proof}, \mathsf{sid}, B')$ from $\mathcal{F}^{R_{\mathsf{DL}\wedge\phi}}_{\mathsf{ZK}}$, each party

$\mathcal{P}_i$ runs Steps 10 to 12 from the **Signing** phase, except now B is computed as

$$B := G_1 + s \cdot H_1 + B'$$

As before, each party $\mathcal{P}_i$ sends $(\texttt{output-share}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i, \mathbf{J})$ to $\mathcal{C}$ before halting. If any party receives $(\texttt{abort}, \mathsf{sid}, B')$ from $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}\wedge\phi}}$, then they output $(\texttt{failure}, \mathsf{sid})$ and halt.

16. Upon receiving $(\texttt{output-share}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i, \mathbf{J})$ from every $\mathcal{P}_i$ for $i \in \mathbf{J}$, the client $\mathcal{C}$ aborts if any two parties disagree on the values of s, e, or pk. Otherwise, $\mathcal{C}$ parses pk as $(\mathbf{H}, X)$, computes B via the same equation as in Step 15, computes

$$A := \frac{\sum\limits_{i \in \mathbf{J}} R_i}{\sum\limits_{i \in \mathbf{J}} u_i}$$

$$s' := s_0 + s$$

and verifies that $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s')) = 1$. If so, then $\mathcal{C}$ outputs $(\texttt{output}, \mathsf{sid}, \mathsf{sigid}, (A, e, s'), \mathsf{pk})$ to $\mathcal{Z}$, and halts, but if the equality does not hold, then $\mathcal{C}$ aborts.

a. In the programmable random oracle model, an optimization is available to reduce the public key size: the parties sample and store only H_1 during steps 4 through 8. When they require H_i for $i \neq 1$, in order to provide them as output to the environment, and in the **Signing** and **Weak Partially-Blind Signing** phases, they (locally) calculate $H_i := \mathsf{RO}(i \| H_1)$, where RO is a random oracle. A more thorough disscussion follows this protocol description.

Theorem 4.2. *$\pi_{\mathsf{BBS+}}(n, t, \mathcal{G})$ UC-realizes $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G})$ in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}\wedge\phi}}, \mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}}}, \mathcal{F}_{\mathsf{Zero}}(n, p), \mathcal{F}_{\mathsf{Mul2P}}(p))$-hybrid model with selective abort against a malicious adversary that statically corrupts up to $t-1$ fixed parties and any number of transient clients, where p is the order of the bilinear group described by $\mathcal{G}$.*

Proof is given in the full version.

Optimizing public key length. As previously mentioned in footnote *a* of Protocol 4.1, a programmable random oracle can be used to make the public key size independent of ℓ: in particular, under the optimization described, the public key comprises only two group elements: one each from $\mathbb{G}_1$ and $\mathbb{G}_2$. Note that the functionality $\mathcal{F}_{\mathsf{BBS+}}$ determines the values H_i for $i \neq 1$, and it *must* do so in order for the signature scheme to remain secure; it follows that the random oracle used to calculate these values in the protocol *must* be programmed in the ideal world by the simulator, with the values the functionality supplies. This optimization weakens the security theorem, which does not otherwise incorporate a random oracle. Nevertheless, for the performance analyses in Sections 6 and 7 we assume this optimization is applied.

Proving selective attributes. Credentials produced by this system can be used in a selective fashion. Camenisch et al. [18] construct a concretely efficient proof of the following relation:

$$R_{\mathsf{BBS+}} = \left\{ \begin{array}{l} ((\boldsymbol{\mu}, \mathbf{I}, \mathsf{pk}), (\mathbf{m}, \sigma)) : \\ \quad \bigwedge\limits_{i \in [|\mathbf{I}|]} \mu_i = m_{I_i} \\ \quad \wedge\ \mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, \sigma) = 1 \end{array} \right\}$$

which selectively reveals some subset $\boldsymbol{\mu}$ (indexed by $\mathbf{I}$) of the messages in a BBS+ signature to a verifier. Using this proof, a credential holder can authenticate to a service that only requires the authority of some subset of the issuers, without revealing their relationship with the other issuers. Yet because it is a a proof over only a single signature, it is far more efficient (computationally, and in terms of communication and storage) than the naive solution of proving knowledge of many independently-signed credentials.

4.2. A Simple Application: Credential Coalescing

As we discussed in Section 1, our scheme can be used to thresholdize any single-issuer anonymous credential scheme based upon BBS+. Here we discuss a related application: coalescing of credential from multiple authorities. Suppose a user is known to multiple credential authorities, and wishes to authenticate to a service by proving a joint statement about who these authorities believe the user to be, and what each of them (individually) authorizes the user to do. On the other hand, the user may not wish to reveal to one authority their relationship with another authority. Our weak partially-blind signing protocol allows the user to prove a different predicate privately to each signing server. The user can request a vector of messages to be signed in a weak-blind fashion, each message representing one of the issuing authorities' credentials, and convince each issuing authority independently that its corresponding message is acceptable simply by proving equality with some string the issuing authority has fixed. The result is a single *compact* credential that coalesces the relationship between the user and all of the issuing authorities.

It should be noted for properties only subsets of authorities are authorized to issue, care must be taken to ensure the credentials with these properties are not issued without consent and involvement of the relevant servers. In practice, this can be implemented by having parties check this requirement before participating in signing (in the clear for regular signing or appending to the predicate in weak partially-blind signing) or by requiring $t = n$.

5. Extensions

5.1. Strong Blind Signatures

Our weak partially-blind signing protocol can be modified to achieve strong blindness, without increasing its round count and with only a modest increase in other cost metrics. Under this modification is realizes a version of the $\mathcal{F}_{\mathsf{BBS+}}$ functionality that has no leakage to $\mathcal{S}$ in the blind-signing phase.

Strong blindness requires that the e must also be masked in addition to s and A. To achieve this, the client samples an additional masking nonce $e_0 \leftarrow \mathbb{Z}_p$ and initiates an instance of $\mathcal{F}_{\mathsf{Mul2P}}(p)$ with every signing party, using e_0 as its input. Each signing party $\mathcal{P}_i$ receives $c_{i,0}$ as output from this multiplier, which is added to u_i. When $\mathcal{P}_i$ sends u_i to the client, it also completes the multiplier instance, supplying r_i as its input. Thus the client receives $d_{0,i}$ from $\mathcal{F}_{\mathsf{Mul2P}}(p)$ such that $c_{i,0} + d_{0,i} = e_0 \cdot r_i$ for every $i \in [n]$ along with R_i and u_i, computes

$$A' := \frac{\sum_{i\in[n]} R_i}{\sum_{i\in[n]} (u_i + c_{0,i})} \qquad \text{and} \qquad e' := e_0 + e$$

and takes the signature to be (A', e', s') instead of (A, e, s'). This modification information-theoretically hides the messages and the signature from the signing parties (in the hybrid model), even if all of the signing parties are corrupt, and the proof of security we have given in Section **??** extends naturally.

Note that because the multiplication protocol of Doerner et al. [36] (with which we propose to realize $\mathcal{F}_{\mathsf{Mul2P}}(p)$) requires only two messages, the client can arrange to play the role of Bob, and send the first message along with the client's signature request, and then the signing parties can send the second message along with their outputs. For the sake of computational efficiency, Doerner et al. base their protocol on OT-extension, which requires a one-time setup protocol to be run before the multiplication protocol begins. In our context, this would imply additional rounds for any client who has not previously interacted with the signing parties. Fortunately, it is possible to achieve two-round chosen-input oblivious transfer without advance setup via endemic OT [64], though at noticeably increased computational cost. Replacing OT extension with endemic OT in the protocol of Doerner et al. allows us to achieve fully-blind signing without increasing the round complexity of our protocol.

5.2. Oblivious Threshold VRF Evaluation

Dodis and Yampolskiy [44] proposed a verifiable random function (VRF) of the form $F_x(e) \mapsto \mathsf{e}(G_1, G_2)/(x+e)$ where the proof of correct evaluation is of the form $\pi = G_1/(x + e)$. In its strongly-blind form as described in Section 5.1, our protocol can be used as a threshold *oblivious* evaluation protocol for the DY VRF, which maintains obliviousness even if all key-holders are corrupt, and achieves a composable security guarantee with a clean functionality.

5.3. Proactive Security

Ostrovsky and Yung [47] conceived of the *mobile adversary* model, in which an attacker might corrupt every device throughout the lifetime of the system, while never corrupting more than a threshold number at any given time. Herzberg et al. [65] devised a method to defend against such an adversary, by attempting to rerandomize the state of the system before the adversary corrupts a new party. In our constructions, the state of the system is characterized by additive shares of the secret x, and the OT correlations that are extended for use by the multiplier. This is exactly the same state maintained by the $(2, n)$ ECDSA construction of Kondi et al. [46] and we are able to directly apply their technique to proactivize a $(2, n)$ instantiation of our system. Note that a more general (t, n) proactivization in the UC context requires either relaxing the security definition, or using a more abstract functionality as discussed extensively by Canetti et al. [33].

6. Cost Analysis

In this section we present a closed-form cost analysis of the bandwidth and computational costs associated with our protocol given in Protocol 4.1, where the functionalities are realized as suggested in Section 3.1. We count the total number bits transmitted per signing server, but with respect to computational costs, we focus only on the most computationally-expensive elements of our protocol, which are the operations over the bilinear group. In Section 7, we implement and benchmark our protocol to demonstrate the concrete impact of these costs. Since the blind signing protocol requires an application-dependent predicate to be defined, and this predicate heavily influences the cost of the protocol, we consider only the costs of the non-blind signing protocol.

Building Blocks. We instantiate our multiplication functionality via the multiplication protocol of Doerner *et al.* [36], but to realize the underlying OT-extension, we use the new SoftSpokenOT protocol [66] in place of the KOS protocol they suggested, along with the Endemic OT protocol of Masny and Rindal [64] for the base OTs. We modify SoftspokenOT via the Fiat-Shamir transform to run in two rounds. The average bandwidth cost (that is the number of bits transmitted by any single party, on average) for this modified form of SoftSpokenOT is

$$\mathsf{ROTeCost}(\lambda, \ell) \mapsto \left(\frac{3}{2} + \frac{1}{2k_{\mathsf{SSOT}}}\right) \cdot (\lambda^2 + \lambda) + \frac{\lambda \cdot \ell}{2k_{\mathsf{SSOT}}}$$

where ℓ is the number of OT extensions in the batch [67]. This cost function includes a parameter k_{SSOT} which controls the trade-off between bandwidth and computation cost. For calculating concrete bandwidth numbers, we set $k_{\mathsf{SSOT}} = 2$, since Roy suggested [67] this yields a strict improvement over KOS.

We can write the average bandwidth cost of the Doerner et al. multiplier as follows:

$$\begin{aligned}\mathsf{COTeCost}(\lambda, \ell, n) &\mapsto \ell \cdot n/2 + \mathsf{ROTeCost}(\lambda, \ell)\\ \mathsf{MulCost}(\lambda, \kappa, s) &\mapsto \mathsf{COTeCost}(\lambda, 2\kappa + 2s, 2\kappa)\\ &\quad + \kappa \cdot (2\kappa + 2s + 1)/2\end{aligned}$$

where s is the statistical security parameter. For calculating concrete bandwidth numbers, we let $s = 80$. This function gives the number of bits transmitted per party on average.

In Table 6.2 we report concrete values for several specific parameterizations.

The foregoing multiplication strategy requires a one-time setup protocol comprising λ instances of base OT. The Endemic OT scheme [64] that we choose for base OTs requires a key agreement protocol; using DHKE over an elliptic curve with elements of size $|G_1|$, the average number of bits transmitted per party is

$$\mathsf{MulSetupCost}(\lambda, |G_1|) \mapsto (2\lambda \cdot |G_1|)$$

and furthermore it requires 4λ elliptic curve scalar operations per party. This setup protocol can be run simultaneously with key generation.

Because we use optimistic echo-broadcast to instantiate our broadcast channel, we consider the cost of a broadcast to be equivalent to sending to all parties via point-to-point channels. Thus we consider the cost of a (broadcast) commitment to be 2λ bits per destination party, and the cost of a decommitment to be equal to the size of the committed value times the number of destination parties. We consider the cost of an instance of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, where R_{DL} is over $\mathbb{G}_2$, to be $(2 \cdot |G_2| + \kappa) \cdot \lambda / \log_2 \lambda$ bits; the overhead relative to the normal cost of a sigma protocol is due to the Fischlin transform. The cost of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}}}$ is the cost of committing and decommitting this same number of bits.

Our Protocol. We divided our BBS+ protocol into its components for key generation and signing, and constructed the cost functions for each component from the above sub-protocol costs. Our SetupCost function combines the cost of key generation, multiplier setup, and fixing shared PRF keys for instantiating $\mathcal{F}_{\mathsf{Zero}}$ (as discussed in Section 3.1). We assume the random-oracle-based optimization described in footnote *a* of Protocol 4.1 is applied.

$$\begin{aligned}&\mathsf{SetupCost}(n, \ell, \lambda, \kappa, |G_1|, |G_2|) \mapsto \\ &\quad (n-1) \cdot \begin{pmatrix} 4\lambda + \kappa + (2 \cdot |G_2| + \kappa) \cdot \lambda / \log_2 \lambda \\ + |G_1| + \mathsf{MulSetupCost}(\lambda, |G_1|) \end{pmatrix} \\ &\mathsf{SignCost}(n, t, \ell, \lambda, s, \kappa, G_1, G_2) \mapsto \\ &\quad (n-1) \cdot (3\lambda + \kappa + 2 \cdot \mathsf{MulCost}(\lambda, \kappa, s)) \\ &\quad + |G_1| \cdot (\ell + 2) + |G_2| + 3\kappa + t \log n\end{aligned}$$

Finally, the client's signing request involves transmitting $n \cdot \ell \cdot \kappa$ bits in total. In terms of computation, each signing server must perform $\ell + 2$ scalar multiplications in $\mathbb{G}_1$ in order to create a signature, and the client must perform $\ell + 1$ scalar multiplications in $\mathbb{G}_1$ plus one scalar in $\mathbb{G}_2$ and two pairing operations in order to verify the signature.

The current recommendations [68] of The Internet Engineering Task Force (IETF) for "pairing-friendly" elliptic curves are the BLS12_381 and BN462 [69] curves, corresponding to a 128-bit security level, and the BLS48_581 [70] curve, corresponding to a 256-bit security level. Specifications for these curves are listed in Table 6.1, and for each curve and its associated security parameter, we give concrete bandwidth costs, in bits transmitted per party, in Table 6.2.

	κ	p	$\lvert G_1\rvert$	$\lvert G_2\rvert$	λ
BLS12_381	255	$\sim 2^{255}$	384	768	126
BN462	462	$\sim 2^{462}$	232	232	128
BLS48_581	517	$\sim 2^{517}$	584	1168	256

TABLE 6.1: IETF-Recommended Pairing Curve Specifications: Group order bit-length (κ), group order (p), and group element sizes for curves BLS12_381, BN462, and BLS48_581 and their corresponding security levels (λ).

7. Implementation and Benchmarks

We implemented and benchmarked our protocol in Rust, using the BLS12_381 curve for both for the signature scheme and for the base OT protocol underlying the multiplication protocol. Our implementation took roughly 6400 lines of code including comments and extensive test suites, and it was compiled using `rustc 1.62.0-nightly (3f052d8ee)`. In this section we present the results of our benchmarks, and in Appendix B we compare these results against related works.

Our experiment consists of n server processes and a client; when started, the n server instances establish connections between themselves and then listen on a network port for requests from a client. The client sends a signing request to all servers, then waits for the responses, and assembles the signature. We measure wall-clock time independently for servers and the client, because they have different workloads. Each configuration of the experiment was run at least 150 times to compute aggregate statistics. As all of our experiments involve timings from multiple parties, we always report the statistics from the party that recorded the *maximum* average time in each protocol execution.

We performed experiments on three network environments: local, LAN, and WAN. In the local environment, all n server processes and the client were executed on the same physical machine. This machine had a 16-Core AMD Ryzen 9 7950X processor (model 97, stepping 2) and 64GB of RAM. Our LAN and WAN benchmarks used Google Cloud C2D-STANDARD-4 instances, which at the time had 4 vCPUs partitioned from an AMD EPYC 7B13 processor (model 1, stepping 0) and 16GB of RAM. These instance were running linux kernel 5.10.0. For LAN benchmarks, all instances were colocated in the US-EAST1-C zone. For WAN benchmarks, the first 12 server instances were spread among zones US-EAST1-*, US-EAST4-*, US-CENTRAL1-*, and US-WEST1-* the next 13 servers were spread across EUROPE-WEST1-*, EUROPE-WEST2-*, and EUROPE-WEST4-*, and the remaining 7 were again spread across the US zones. In all cases, the client was located in US-EAST1-C.

We evaluated Local, LAN and WAN setup and signing operations for the n-of-n case for $n \in [2, 32]$. These timings closely reflect the performance for any t-of-n regime where $t \in [2, 32]$. The primary difference between n-of-n and t-of-n is that the identities of the t parties involved in the session must be shared, and each party must locally multiply

	BLS12_381	BN462	BLS48_581
	$\kappa = 255$	$\kappa = 462$	$\kappa = 517$
Multiplication Cost	306739	815027	1117758
Setup Cost	$132205 \cdot (n-1)$	$77531 \cdot (n-1)$	$392429 \cdot (n-1)$
Signing Cost	$(n-1) \cdot (873697 + t \log n)$	$(n-1) \cdot (1884470 + t \log n)$	$(n-1) \cdot (2937983 + t \log n)$

TABLE 6.2: Bandwidth Costs in total bits transmitted per party, for n parties who wish to sign a vector of ℓ messages.

their secret share with a Lagrange coefficient. These steps contribute negligibly to wall-clock time and bandwidth. We believe that in practice the typical number of issuers will be less than 10, but we provide extra datapoints to experimentally confirm scaling behavior.

Signing results. Our results for signing are reported in Figure 2. In the LAN setting, when $n = 6$, the servers incur a wall-clock time from request to response of 5.1ms, whereas the client experiences 11.3ms of latency from input to output and signature verification (regardless of n) requires 5.0ms; this means that the dominant concrete cost for $n \leq 5$ in the LAN setting is actually verification of the signature by the client! No alternative approach can hope to do much better in this regime, unless it avoids verifying the signature that the protocol produces.

Our local experiments exhibited slightly slower times than our LAN experiments, especially when $n > 16$ and there was more than one server process per core. WAN costs seem to be dominated by network latency. The large gap between $n = 12$ and 13 occurs because the 13th server is located in Europe and incurs trans-Atlantic latencies; the one between $n = 5$ and 6 is due to adding a west-coast zone to the experiment. The graph shows that these latencies overwhelm the compute time. We note the cost of running this protocol is comparable to the cost of serving a modern web application, with response times that are measured in the 100s of milliseconds.

Setup results. Our setup protocol is more costly than our signing protocol, because it performs the oblivious transfer and extension operations required to initialize the multiplier protocols. As predicted by the analysis in Section 6, these operations require more network bandwidth. Measurements are shown in Figure 3.

Overhead of MPC. To measure the "overhead of our MPC", we also measured signing and verification operations for the standard BBS+ signature scheme using the same Rust elliptic curve libraries. These times were collected using Rust's built-in test framework, and they are reported in Table 7.1. To help gauge these results on different machines, we also provide micro-benchmarks for scalar curve operations of our implementation.

In moving from the single party implementation to the 2-of-2 threshold case, we see that the overhead of MPC for the signing operation is 3x in the local and LAN environment, and roughly 70x for the WAN environment due mostly to network latencies. In comparison to generic MPC techniques which have overheads in the range of 10^4 or more, these results show that our tailored MPC approach has merit, resulting in overhead that can be tolerated in some applications.

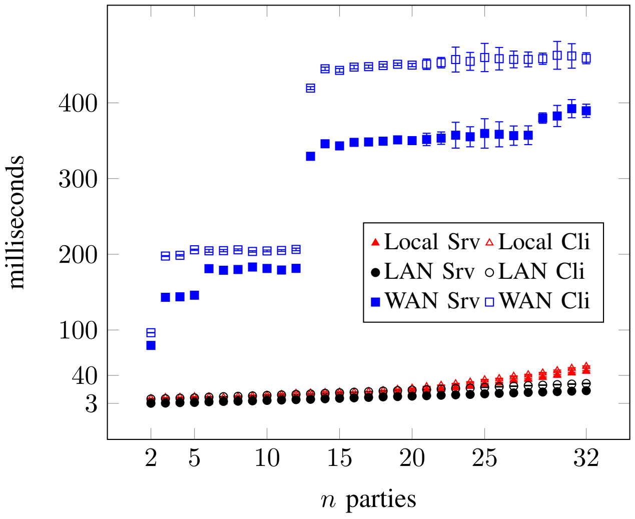

Figure 2: Protocol signing timings for n-out-of-n over Local, LAN and WAN setups, as expected in a practical deployment for credentials. Error bars depict standard deviation over 150+ runs. The server time reflects the backend computation, measured as the max average over the servers from the time a request is received by a server until a response is sent. The client time reflects the time from when a request is sent to all servers until a signature on the message has been reconstructed and verified from the responses.

Operation	Time
Key Generation	1.994ms $\pm$ 5.2μs
Sign	1.185ms $\pm$ 4.3μs
Verify	5.008ms $\pm$ 31.6μs
G_1 scalar multiplication	0.391ms $\pm$ 3.6μs
G_2 scalar multiplication	1.204ms $\pm$ 10.4μs

TABLE 7.1: BBS+ operations using BLS12_381, as measured by the Rust benchmark. The error terms represent standard deviation. Measurements were taken on the GCP instance used for LAN tests.

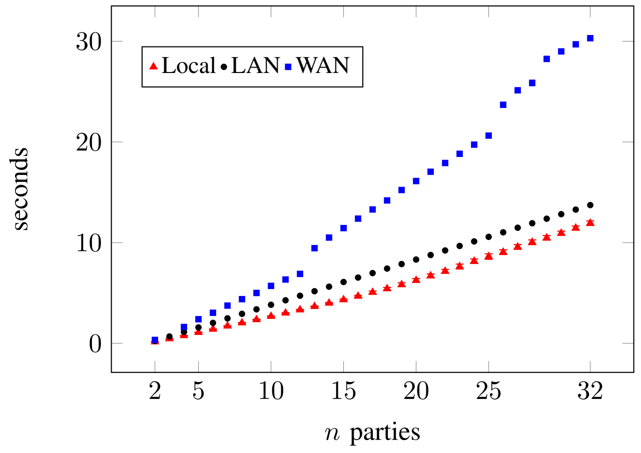

Figure 3: Protocol setup timing for n-out-of-n over Local, LAN and WAN setups. Error bars depict standard deviation over 150+ runs.

References

[1] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Communications of the ACM*, vol. 28, no. 10, p. 1030–1044, oct 1985. [Online]. Available: https://doi.org/10.1145/4372.4373

[2] D. Chaum and J.-H. Evertse, "A secure and privacy-protecting protocol for transmitting personal information between organizations," in *Proceedings on Advances in Cryptology—CRYPTO '86*. Berlin, Heidelberg: Springer-Verlag, 1987, p. 118–167.

[3] L. Chen, "Access with pseudonyms," in *Cryptography: Policy and Algorithms*, E. Dawson and J. Golić, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 232–243.

[4] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf, "Pseudonym systems," in *Selected Areas in Cryptography*, 1999.

[5] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology – EUROCRYPT 2001*. Springer Berlin Heidelberg, 2001.

[6] ——, "Signature schemes and anonymous credentials from bilinear maps," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–72.

[7] J. Camenisch, S. Krenn, A. Lehmann, G. L. Mikkelsen, G. Neven, and M. Ø. Pedersen, "Formal treatment of privacy-enhancing credential systems," in *Selected Areas in Cryptography – SAC 2015*, O. Dunkelman and L. Keliher, Eds. Cham: Springer International Publishing, 2016, pp. 3–24.

[8] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss, "Composable and modular anonymous credentials: Definitions and practical constructions," in *Advances in Cryptology – ASIACRYPT 2015*, T. Iwata and J. H. Cheon, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 262–288.

[9] M. Chase, S. Meiklejohn, and G. Zaverucha, "Algebraic macs and keyed-verification anonymous credentials," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1205–1216. [Online]. Available: https://doi.org/10.1145/2660267.2660328

[10] M. Chase, T. Perrin, and G. Zaverucha, *The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1445–1459. [Online]. Available: https://doi.org/10.1145/3372297.3417887

[11] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Advances in Cryptology - CRYPTO 2006*, C. Dwork, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 78–96.

[12] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *Advances in Cryptology - CRYPTO 2009*, S. Halevi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 108–125.

[13] E. C. Crites and A. Lysyanskaya, "Delegatable anonymous credentials from mercurial signatures," in *Topics in Cryptology – CT-RSA 2019*, M. Matsui, Ed. Cham: Springer International Publishing, 2019, pp. 535–555.

[14] M. Chase, E. Ghosh, S. Setty, and D. Buchner, "Zero-knowledge credentials with deferred revocation checks," https://github.com/decentralized-identity/snark-credentials/blob/master/whitepaper.pdf.

[15] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic k-taa," in *Proceedings of the 5th Conference on Security and Cryptography for Networks (SCN)*, R. De Prisco and M. Yung, Eds., 2006, pp. 111–125.

[16] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology – CRYPTO 2004*, 2004, pp. 41–55.

[17] E. Brickell and J. Li, "Enhanced privacy id from bilinear pairing," *IACR Cryptol. ePrint Arch.*, vol. 2009, p. 95, 2009.

[18] J. Camenisch, M. Drijvers, and A. Lehmann, "Anonymous attestation using the strong diffie hellman assumption revisited," in *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016*. Springer, 2016, pp. 1–20.

[19] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "Blacklistable anonymous credentials: Blocking misbehaving users without ttps," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 72–81. [Online]. Available: https://doi.org/10.1145/1315245.1315256

[20] T. Looker, V. Kalos, A. Whitehead, and M. Lodder, "The BBS Signature Scheme," Internet Engineering Task Force, Internet-Draft draft-irtf-cfrg-bbs-signatures-01, Oct. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/01/

[21] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2001, pp. 136–145.

[22] D. Beaver, "Precomputing oblivious transfer," 1995, pp. 97–109.

[23] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," 1986.

[24] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols (extended abstract)," 1990, pp. 503–513.

[25] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," 2017, pp. 21–37.

[26] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," 1987.

[27] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," 2012, pp. 643–662.

[28] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," 2016, pp. 830–842.

[29] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent OT extension and more," 2019, pp. 489–518.

[30] ——, "Correlated pseudorandom functions from variable-density LPN," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, S. Irani, Ed. IEEE, 2020, pp. 1069–1080. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00103

[31] J. Bar-Ilan and D. Beaver, "Non-cryptographic fault-tolerant computing in constant number of rounds of interaction," in *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1989.

[32] J. Aumasson, A. Hamelink, and O. Shlomovits, "A survey of ECDSA threshold signing," *IACR Cryptol. ePrint Arch.*, p. 1390, 2020. [Online]. Available: https://eprint.iacr.org/2020/1390

[33] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *CCS '20*. ACM, 2020.

[34] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold EC-DSA," in *Public-Key Cryptography - PKC 2020*, 2020.

[35] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," 2018, pp. 1837–1854.

[36] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *Proceedings of the 39th IEEE Symposium on Security and Privacy, (S&P)*, 2018, pp. 980–997.

[37] ——, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *Proceedings of the 40th IEEE Symposium on Security and Privacy, (S&P)*, 2019.

[38] A. P. K. Dalskov, C. Orlandi, M. Keller, K. Shrishak, and H. Shulman, "Securing DNSSEC keys via threshold ECDSA from generic MPC," in *ESORICS 2020*, 2020.

[39] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*. Berlin, Heidelberg: Springer-Verlag, 2012, p. 643–662. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_38

[40] N. P. Smart and Y. T. Alaoui, "Distributing any elliptic curve based protocol," in *IMACC 2019*, M. Albrecht, Ed., 2019.

[41] I. Haitner, N. Makriyannis, S. Ranellucci, and E. Tsfadia, "Highly efficient ot-based multiplication protocols," *EUROCRYPT '22*, 2022.

[42] Y. Lindell, "Simple three-round multiparty schnorr signing with full simulatability," *IACR Cryptol. ePrint Arch.*, p. 374, 2022. [Online]. Available: https://eprint.iacr.org/2022/374

[43] T. Okamoto, "Efficient blind and partially blind signatures without random oracles," in *Proceedings of the Third Theory of Cryptography Conference, TCC 2006*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 80–99.

[44] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *Public Key Cryptography - PKC 2005*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 416–431.

[45] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," 2003.

[46] Y. Kondi, B. Magri, C. Orlandi, and O. Shlomovits, "Refresh when you wake up: Proactive threshold wallets with offline devices," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 608–625. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00067

[47] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," 1991, pp. 51–59.

[48] R. Gennaro, S. Goldfeder, and B. Ithurburn, "Fully distributed group signatures," 2019.

[49] N. Makriyannis and U. Peled, "A note on the security of gg18," https://info.fireblocks.com/hubfs/A_Note_on_the_Security_of_GG.pdf.

[50] D. Tymokhanov and O. Shlomovits, "Alpha-rays: Key extraction attacks on threshold ecdsa implementations," Cryptology ePrint Archive, Paper 2021/1621, 2021.

[51] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[52] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*. Berlin, Heidelberg: Springer-Verlag, 2016, p. 111–126. [Online]. Available: https://doi.org/10.1007/978-3-319-29485-8_7

[53] A. Rial and A. M. Piotrowska, "Security analysis of coconut, an attribute-based credential scheme with threshold issuance," Cryptology ePrint Archive, Paper 2022/011, 2022, https://eprint.iacr.org/2022/011. [Online]. Available: https://eprint.iacr.org/2022/011

[54] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.

[55] D. Boneh and X. Boyen, "Short signatures without random oracles," in *Advances in Cryptology – EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–73.

[56] ——, "Short signatures without random oracles and the sdh assumption in bilinear groups," *Journal of Cryptology*, vol. 21, no. 2, pp. 149–177, 2008.

[57] T. Okamoto, "Efficient blind and partially blind signatures without random oracles," Cryptology ePrint Archive, Report 2006/102, 2006, https://ia.cr/2006/102.

[58] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002, pp. 494–503.

[59] S. Goldwasser and Y. Lindell, "Secure multi-party computation without agreement," *Journal of Cryptology*, vol. 18, no. 3, pp. 247–287, 2005.

[60] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *Advances in Cryptology – CRYPTO 2005*, 2005, pp. 152–168.

[61] C. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology – CRYPTO 1989*, 1989, pp. 239–252.

[62] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, p. 637–647, jun 1985. [Online]. Available: https://doi.org/10.1145/3812.3818

[63] N. Gilboa, "Two party RSA key generation," in *Advances in Cryptology – CRYPTO 1999*, 1999, pp. 116–129.

[64] D. Masny and P. Rindal, "Endemic oblivious transfer," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 309–326. [Online]. Available: https://doi.org/10.1145/3319535.3354210

[65] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," 1995, pp. 339–352.

[66] L. Roy, "Softspokenot:communication-computation tradeoffs in ot extension," in *Advances in Cryptology – CRYPTO 2022*, 2022.

[67] ——, personal communication, 2022.

[68] Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby, "Pairing-friendly curves," 2020.

[69] S. Bowe, "Bls12-381: New zk-snark elliptic curve construction," 2017.

[70] Y. Kiyomura, A. Inoue, Y. Kawahara, M. Yasuda, T. Takagi, and T. Kobayashi, "Secure and efficient pairing at 256-bit security level," in *Applied Cryptography and Network Security*, D. Gollmann, A. Miyaji, and H. Kikuchi, Eds. Springer International Publishing, 2017, pp. 59–79.

[71] Nymtech, 2022, https://github.com/nymtech/nym/tree/develop/common/nymcoconut.

Appendix A. Functionalities

Functionality A.1. $\mathcal{F}_{\mathsf{Com}}$ [58].

This functionality interacts with parties $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots\}$.

Commitment: On receiving $(\texttt{commit}, \mathsf{sid}, \boldsymbol{\mathcal{P}}, x)$ from some party $\mathcal{P}$, where sid is fresh, store $(\texttt{commitment}, \mathsf{sid}, \mathcal{P}, \boldsymbol{\mathcal{P}}, x)$ in memory and send $(\texttt{committed}, \mathsf{sid}, \mathcal{P})$ to all of the parties identified by $\boldsymbol{\mathcal{P}}$.

Decommitment: On receiving $(\texttt{decommit}, \mathsf{sid})$ from $\mathcal{P}$, if $(\texttt{commitment}, \mathsf{sid}, \mathcal{P}, \boldsymbol{\mathcal{P}}, x)$ exists in memory, then send $(\texttt{decommitment}, \mathsf{sid}, x)$ to all of the parties identified by $\boldsymbol{\mathcal{P}}$.

Functionality A.2. $\mathcal{F}^{\mathcal{R}}_{\mathsf{ZK}}$ [58].

This functionality interacts with an a-priori-unspecified number of parties, designated by $\mathcal{P}$ and $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$. It also has black-box access to the decider for NP-relation $\mathcal{R}$.

Proof: On receiving $(\texttt{prove}, \mathsf{sid}, \boldsymbol{\mathcal{V}}, x, w)$ from $\mathcal{P}$, where sid is fresh and $\boldsymbol{\mathcal{V}}$ is a set of party identifiers, check whether $\mathcal{R}(x, w) = 1$, and send $(\texttt{proof}, \mathsf{sid}, x)$ to all of the parties identified by $\boldsymbol{\mathcal{V}}$ if so. If $\mathcal{R}(x, w) \neq 1$, then send $(\texttt{abort}, \mathsf{sid}, x)$ to the same set of parties.

Functionality A.3. $\mathcal{F}^{\mathcal{R}}_{\mathsf{Com\text{-}ZK}}$ [58].

This functionality interacts with a prover $\mathcal{P}$ and a set of verifiers $\boldsymbol{\mathcal{V}} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$. It also has black-box access to the decider for NP-relation $\mathcal{R}$.

Commitment: On receiving $(\texttt{commit}, \mathsf{sid}, \boldsymbol{\mathcal{V}}, x, w)$ from some party $\mathcal{P}$, where sid is fresh, store $(\texttt{commitment}, \mathsf{sid}, \mathcal{P}, \boldsymbol{\mathcal{V}}, x, w)$ in memory and send $(\texttt{committed}, \mathsf{sid}, \mathcal{P})$ to all of the parties identified by $\boldsymbol{\mathcal{V}}$.

Proof: On receiving $(\texttt{prove}, \mathsf{sid})$ from $\mathcal{P}$, if $(\texttt{commitment}, \mathsf{sid}, \mathcal{P}, \boldsymbol{\mathcal{V}}, x, w)$ exists in memory, then check whether $\mathcal{R}(x, w) = 1$, and send $(\texttt{accept}, \mathsf{sid}, x)$ to all of the parties identified by $\boldsymbol{\mathcal{V}}$ if so. If $\mathcal{R}(x, w) \neq 1$, then send $(\texttt{reject}, \mathsf{sid}, x)$ to the same set of parties.

Functionality A.4. $\mathcal{F}_{\mathsf{Zero}}(p)$

Sample: Upon receiving $(\texttt{sample}, \mathsf{sid}, \mathbf{J})$ from all parties $\mathcal{P}_i$ for $i \in \mathbf{J}$, where sid is a fresh, agreed-upon value, uniformly sample $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^{|\mathbf{J}|}$ conditioned on $\sum_{i \in \mathbf{J}} \alpha_i \equiv 0 \pmod p$ and send $(\texttt{zero-share}, \mathsf{sid}, \alpha_i)$ to each party $\mathcal{P}_i$ as adversarially delayed private output.

Functionality A.5. $\mathcal{F}_{\mathsf{Mul2P}}(p)$ [36].

This functionality interacts with two parties, who we refer to as Alice and Bob. It is parameterized by a prime p that determines the order of the field over which multiplications are performed.

Bob-input: On receiving $(\texttt{input}, \mathsf{sid}, \mathcal{P}_{\mathsf{B}}, b)$ from Bob, if $b \in \mathbb{Z}_p$ and no record of the form $(\texttt{bob-input}, \mathsf{sid}, *, *)$ exists in memory, then sample $c \leftarrow \mathbb{Z}_p$ uniformly, store $(\texttt{bob-input}, \mathsf{sid}, b, c)$ in memory, and send $(\texttt{bob-ready}, \mathsf{sid}, \mathcal{P}_{\mathsf{B}}, c)$ to $\mathcal{P}_{\mathsf{A}}$ (a.k.a. Alice).

Multiplication: On receiving $(\texttt{multiply}, \mathsf{sid}, a)$ from Alice, if $a \in \mathbb{Z}_p$ and there exists a message of the form $(\texttt{bob-input}, \mathsf{sid}, b, c)$ in memory, and if $(\texttt{complete}, \mathsf{sid})$ does not exist in memory, then compute $d := a \cdot b - c \bmod p$, send $(\texttt{product}, \mathsf{sid}, d)$ to Bob, and store $(\texttt{complete}, \mathsf{sid})$ in memory.

Appendix B. Benchmark Comparisons.

Comparison with Threshold ECDSA [37]. As noted in Section 1, our threshold protocol for BBS+ requires fewer operations than the threshold ECDSA signing protocol of Doerner et al., and yet our benchmarks appear to be slower than theirs. To explain, we note that their implementation of ECDSA uses a highly optimized elliptic curve and modular arithmetic library. For example, by employing a scalar multiplication optimization that exploits precomputation, their elliptic-scalar operation requires only 34.2μs versus the 390μs we require for BLS12_381 when measured on the same platform (i.e., their elliptic curve operations are roughly 11x faster). Our multipliers use the slower BLS12_381 curve, although in principle, we could use the faster secp256k1 curve at the cost of introducing an extra security assumption in our protocol. Finally, their ECDSA implementation uses hardware SHA256 accelerations for computing 8 hashes at once.

Comparison with Goldfeder, Gennaro, and Ithurburn [48]. Because no implementation of the Goldfeder, Gennaro, and Ithurburn protocol is available, we attempt to approximate its signing times. Phase 3 of their protocol requires invoking a multiplier between every pair of parties; the most expensive steps in their multiplier requires Alice to encrypt a share, and provide a zero-knowledge range proof on the ciphertext, Bob to perform a scalar multiplication and addition on the ciphertext while also providing a zero-knowlege proof of correctness, and then Alice to decrypt the ciphertext (ignoring commitments, and exponentiations in elliptic curve group). We were implemented the basic encryption operations described above using the `libpaillier` rust library. One such multiplier requires 27.6ms $\pm$ 44μs of compute time. This step alone costs 9x more than full 2-of-2 server latency in our protocol, and since each server in their protocol needs to perform 2 of these multipications with every other server, we expect the computational burden of their protocol to grow quickly into hundreds of milliseconds as n increases. Moreover, the Paillier operations just described are *not* the most expensive component of their protocol: that distinction belongs to the zero-knowledge proofs, and we expect that if they were implemented, the total protocol time is likely to be on the order of several seconds.

Comparison with RP-Coconut [53]. We used Nym's implementation [71] of the RP-Coconut protocol to compare performance. By using an interactive security assumption, the RP-Coconut scheme is simpler and does not require an expensive multiplier operation. However, the protocol does require each server to perform a pairing operation and the client to perform several in order to aggregate results, and thus the performance relationships are not immediately clear.

The RP-Coconut protocol involves three stages that correspond roughly to our protocol: (a) first, the client must prepare a request for message signing and send this to the servers, (b) the servers must run the sign operation, and (c) the client must aggregate each of the received messages into a final signature. Step (a) corresponds almost exactly to the steps we require in our protocol: a commitment and a proof of knowledge of the committed values to be signed; thus we do not benchmark it. The RP-Coconut implementation was incapable of running a full experiment with n servers communicating via a network. Instead, we benchmarked a single server running (with each parameterization) in isolation, and then benchmarked the time required for the client to aggregate. This means that while the client latency for our protocol *includes* the server time, the client time for their protocol *excludes it*. These facts combine to favor their protocol over ours.

As expected, Figure 4 shows that RP-Coconut's server performance remains roughly the same as n increases because each server only performs linear operations on its secret. While our protocol's server performance grows faster than that of RP-Coconut, our protocol's client performance grows much slower than theirs. This implies that when the application context requires the signature to be reconstructed to make progress (e.g., many blockchain settings), our protocol's overall time to create a signature (client + server) is lower than that of RP-Coconut.

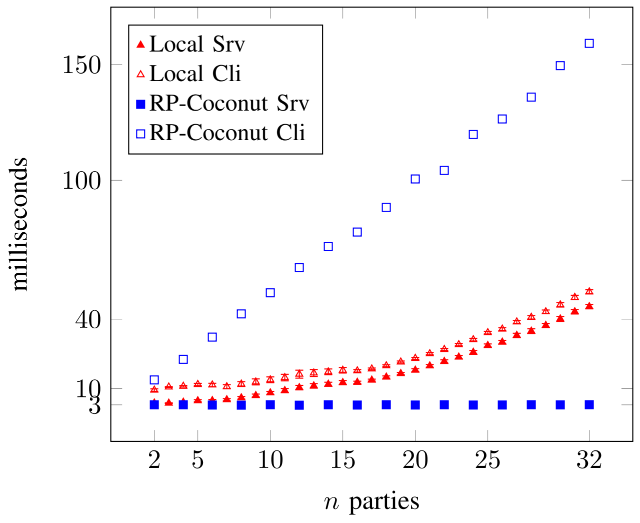

Figure 4: Protocol running times for n-out-of-n signing 3 messages over Local network environment for our protocol versus RP-Coconut. Timings for RP-Coconut were taken by the criterion package, with 100 samples.