Threshold BBS+ Signatures for Distributed Anonymous Credential Issuance

Jack Doerner
j@ckdoerner.net
Technion

Yashvanth Kondi
ykondi@cs.au.dk
Aarhus University

Eysa Lee
lee.ey@northeastern.edu
Northeastern University

abhi shelat
abhi@neu.edu
Northeastern University

LaKyah Tyner
tyner.l@northeastern.edu
Northeastern University

April 27, 2023

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.

Contents

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 [Cha85], and has been refined by a long line of follow-up works [CE87, Che96, LRSW99, CL01, CL04, CKL$^+$16, CDHK15]. Anonymous credentials satisfy two basic security properties: the first is *unlinkability*, which guarantees that no verifier can correlate multiple uses of the same credential (even under arbitrary collusion), and the second is *unforgeability*, which guarantees that no valid credential can be generated without the consent of the issuer. These properties are essential, but a number of additional properties have been defined and realized, such as keyed-verifiability [CMZ14, CPZ20] and delegatability [CL06, BCC$^+$09, CL19].

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 [CL01, CGSB]. 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 [ASM06] and derives its name from the group signature scheme of Boneh, Boyen, and Shacham [BBS04], 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) [BL09, CDL16], k-Times Anonymous Authentication [ASM06], and blacklistable anonymous credentials [TAKS07]. Of particular note is the Enhanced Privacy ID (EPID) of Brickell and Li [BL09], 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+ [LKWL22], 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 curve group. 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 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

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

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 [Can01], 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 1.1.

We would also like to avoid the so-called *preprocessing* model [Bea95] 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 [Yao86, BMR90, WRK17], arithmetic MPC systems where round complexity depends on the multiplicative depth of the circuit [GMW87, DPSZ12, KOS16], and, recently, Pseudorandom-Correlation-based schemes [BCG$^+$19, BCG$^+$20]. The BBS+ scheme is defined over a large finite field. The garbled circuit approach incurs substantial overhead for arithmetic circuits over large fields and so we do not pursue it further. While the Pseudorandom Correlation paradigm is promising, known constructions are either in the preprocessing model [BCG$^+$19] or require non-standard assumptions

[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.

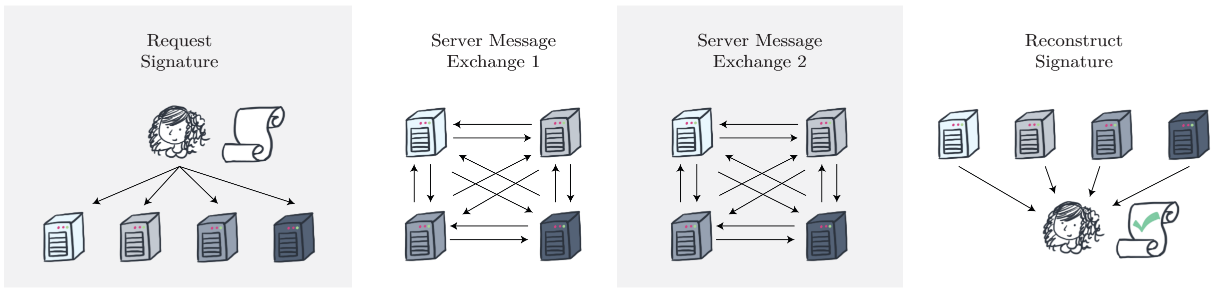

Figure 1.1: We consider a client (i.e. user) requesting a signature from a group of signing servers (i.e. credential issuers). The servers execute a protocol requiring two exchanges of messages before sending responses to the client. From these responses the client is able to reconstruct a signature.

or heavy machinery [BCG+20]. The arithmetic MPC approach is known to be efficient in terms of computation and bandwidth for large fields [KOS16], 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 [BB89] provided an elegant template for solving this problem, which uses only 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. [AHS20] 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 [BB89], which is typically instantiated via either Additively Homomorphic Encryption (AHE) [CGG+20, CCL+20, LN18], or Oblivious Transfer (OT) [DKLs18, DKLs19, DOK+20]. 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 Dal-

skov et al. [DOK$^+$20] 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 [DPSZ12], and equality checks in the ECDSA curve group. The works of Dalskov et al. [DOK$^+$20] and Smart and Alaoui [SA19] show how to implement consistency checking for any secure computation within the 'arithmetic black box' framework [KOS16] 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. [DKLs18, DKLs19] 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. [DKLs18], 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. [HMRT22] 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 [Can01], 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 $A := \sum_i R_i / (\sum_i u_i)$. 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 [BB89]. 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 [SA19, DOK+20]—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 [Lin22], requiring only commitments and proofs of knowledge. ECDSA requires secure multiplication; specifically, it requires multiple products to be securely computed, and then requires some mechanism to ensure consistency between them. Enforcing consistency involves checking implicit (and possibly computational) MACs in the curve group [DKLs19], homomorphically evaluating an encrypted form of the signature scheme [?], or performing zero-knowledge proofs [?]. 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 a corresponding information-theoretic implicit MAC check with no need for additional consistency checks.

Extensions. Okamoto's signature scheme [Oka06a] 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 [DY05]. In addition, since the state that our servers are required to maintain comprises additive secret key shares and base OTs for the OT extension [IKNP03] used by the multiplier, we can use the proactivization scheme of Kondi et al. [KMOS21] to refresh the state of the system and defend it against mobile attackers [OY91].

1.3 Prior Works

Dodis and Yampolskiy [DY05] 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 [BB89]; 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 [GGI19]. 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 [GGI19] have been shown explicitly to be insecure in the context of threshold ECDSA signing [MP, TS21]; the impact of this on the BBS+ protocol is at present unclear. Finally, unlike the [GGI19] 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 [SAB+19]. This scheme, Coconut, is primarily based upon the signature scheme of Pointcheval and Sanders [PS16] (PS signatures), which base their security on an interactive assumption similar but not equivalent to LRSW [LRSW99]. 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 [RP22] (RP-Coconut), however, identifies security problems with the proof sketch of Sonnino *et al.* [SAB+19] and provides a patch. In order to prove unforgeability, RP-Coconut requires increasing the size of the public key to ℓ elements of $\mathbb{G}_1$ plus $\ell + 1$ elements of $\mathbb{G}_2$, where ℓ is the maximal number of elements in any vector of messages that can be signed. 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 [PS16] 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 q, such that $|q| = \kappa$, 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 $\mathsf{sans\text{-}serif}$, 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. We let $\mathsf{lagrange}(\mathbf{J}, j, 0)$ denote party $\mathcal{P}_j$'s Lagrange coefficient for Shamir-reconstruction with the parties indexed by $\mathbf{J}$; that is, the coefficient for the one point that it knows on a polynomial, such that the sum over all the parties indexed by $\mathbf{J}$ interpolates the polynomial at location 0.

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 $x, \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$, $\mathsf{e}(x \cdot G_1, y \cdot G_2) = x \cdot y \cdot \mathsf{e}(G_1, G_2)$. We define $\mathsf{BilinGen}$ to be an efficient algorithm $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q) = \mathcal{G} \leftarrow \mathsf{BilinGen}(1^\lambda)$, which samples a description $\mathcal{G}$ of the group (with λ bits of security). There are three types of pairings [GPS08]: 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 $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q) := \mathcal{G}$.
2. Sample a vector of $\ell + 1$ random group elements $\mathbf{H} \leftarrow \mathbb{G}_1^{\ell+1}$.
3. Uniformly choose secret key $x \leftarrow \mathbb{Z}_q^*$.
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. $\mathsf{BBS{+}Sign}(\mathsf{sk}, \mathbf{m} \in \mathbb{Z}_q^\ell)$

1. Parse sk as $(\mathbf{H}, x)$.
2. Uniformly sample nonces $e \leftarrow \mathbb{Z}_q$ and $s \leftarrow \mathbb{Z}_q$.
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 pk as $(\mathbf{H}, X)$ and σ 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.* [ASM06] introduced BBS+ and proved it secure for type-1 and type-2 pairings, using the original "conference version" of the qSDH assumption [BB04].

Lemma 2.4 ([ASM06])**.** *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.* [CDL16], under the updated "journal version" of the qSDH assumption [BB08].

Lemma 2.5 ([CDL16])**.** *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* require qSDH or any other specific computational assumption.

Comparison to Okamoto Signatures Okamoto's signature scheme [Oka06a] was originally introduced in the context of constructing blind signatures. As Au *et al.* [ASM06] 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 [Oka06b] 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.* [ASM06] and Camenisch *et al.* [CDL16] 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 focus mainly on 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. In section 5.1, we discuss an extension of our scheme that achieved strong partially-blind signing.

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 [Can01] 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 $\textsc{Real}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$ 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 $\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$ 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 "admissible" environment $\mathcal{Z}$,[4]

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

3 Functionalities

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

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

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 $t \le n$, a message count ℓ, and the description of a bilinear group, $\mathcal{G}$. During setup, $\mathcal{S}$ may instruct the functionality to abort. During any signing phase, $\mathcal{S}$ may instruct the functionality to output a "failure" to client instead of a signature. This is similar to an abort, except that the functionality does not halt, but continues accepting inputs.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from some party $\mathcal{P}_i$ for $i \in [n]$ where sid is fresh, send $(\texttt{init-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{init}, \mathsf{sid})$ from every party $\mathcal{P}_i$ for $i \in [n]$, where 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 $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ in memory, and send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{S}$ and 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 $\mathbf{m} \in \mathbb{Z}_q^\ell$, $\mathbf{J} \subseteq [n]$, $|\mathbf{J}| = t$, and sigid is fresh, if a record of the form $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk}, \ell)$ exists in memory, then send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \mathbf{m}, \mathbf{J})$ to every party $\mathcal{P}_j$ for $j \in \mathbf{J}$ as adversarially-delayed private output.

On receiving $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ or $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, send $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ or $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid}, j)$ to $\mathcal{S}$, respectively.

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 $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk})$ 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.

[4] Admissibility is a minimal well-formedness condition that ensures all entities in the experiment are able to run for an appropriate amount of time. Per standard practice, we will only consider admissible environments, but leave their admissibility implicit.

Weak Partially-Blind Signing: Upon receiving $(\texttt{wb-sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from $\mathcal{C}$, where $\mathbf{m} \in \mathbb{Z}_q^\ell$, $\mathbf{J} \subseteq [n]$, $|\mathbf{J}| = t$, and sigid is fresh, if a record of the form $(\texttt{key}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk}, \ell)$ exists in memory, then wait to receive $(\texttt{wb-pred}, \mathsf{sid}, \mathsf{sigid}, j, \phi_j)$ from $\mathcal{C}$, where ϕ_j is the description of a predicate on $\mathbf{m}$ such that $\phi_j(\mathbf{m}) = 1$, and upon receiving such a message for some $j \in \mathbf{J}$, send $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \phi_j, \mathbf{J})$ to party $\mathcal{P}_j$.

On receiving $(\texttt{wb-pred}, \mathsf{sid}, \mathsf{sigid}, j, \phi_j)$ from $\mathcal{C}$ and either $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ or $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, send $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ or $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid}, j)$ to $\mathcal{S}$, respectively.

On receiving $(\texttt{wb-pred}, \mathsf{sid}, \mathsf{sigid}, j, \phi_j)$ from $\mathcal{C}$ and $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ from every $\mathcal{P}_j$ for every $j \in \mathbf{J}$, 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 $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk})$ 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 5.1.

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 [Can01, CLOS02]). 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 [GL05]. 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 [CLOS02]. The parties also use a functionality $\mathcal{F}_{\mathsf{Zero}}$ that produces secret sharings of zero in a particular group.

Functionality 3.2. $\mathcal{F}_{\mathsf{Com}}$ **[CLOS02]**

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 3.3. $\mathcal{F}^{\mathcal{R}}_{\mathsf{ZK}}$ [CLOS02]

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 $(\texttt{accepted}, \mathsf{sid}, \mathcal{P}, x)$ to all of the parties identified by $\boldsymbol{\mathcal{V}}$ if so. If $\mathcal{R}(x, w) \neq 1$, then send $(\texttt{rejected}, \mathsf{sid}, \mathcal{P}, x)$ to the same set of parties.

Functionality 3.4. $\mathcal{F}^{\mathcal{R}}_{\mathsf{Com\text{-}ZK}}$ [CLOS02]

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 $(\texttt{accepted}, \mathsf{sid}, x)$ to all of the parties identified by $\boldsymbol{\mathcal{V}}$ if so. If $\mathcal{R}(x, w) \neq 1$, then send $(\texttt{rejected}, \mathsf{sid}, x)$ to the same set of parties.

Functionality 3.5. $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{G})$

This functionality is parameterized by the party count n and a group $\mathbb{G}$.

Sample: Upon receiving $(\texttt{sample}, \mathsf{sid}, \mathbf{J})$ from some party $\mathcal{P}_i$ for $i \in \mathbf{J}$, where $\mathbf{J} \subseteq [n]$ and sid is fresh, uniformly sample $\mathbf{x} \leftarrow \mathbb{G}^{|\mathbf{J}|}$ conditioned on $\sum\limits_{i \in \mathbf{J}} x_i \equiv 0_{\mathbb{G}}$ and send $(\texttt{zero-share}, \mathsf{sid}, x_i)$ to $\mathcal{P}_i$. Upon receiving $(\texttt{sample}, \mathsf{sid}, \mathbf{J})$ from any other $\mathcal{P}_j$ for $\in \mathbf{J} \setminus \{i\}$, send $(\texttt{zero-share}, \mathsf{sid}, x_j)$ to $\mathcal{P}_j$.

$\mathcal{F}_{\mathsf{Com}}$ 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 twice the length of the security parameter, 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 the party pairs in which it is lower-indexed, and adds oracle outputs for the party pairs 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

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

$\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ can be realized via the Fischlin [Fis05] or Kondi-shelat [KS22] transforms applied to the Schnorr protocol [Sch89]. 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 $\mathcal{F}_{\mathsf{Com}}$.

Threshold discrete log key sampling. As specified in section 2.1, the public key of the BBS+ signature scheme over $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q) = \mathcal{G}$ comprises $\ell + 1$ random elements of $\mathbb{G}_1$, plus a single element X of $\mathbb{G}_2$. To sign a message, only the discrete logarithm x of X is required; thus to thresholdize the BBS+ scheme, x must be sampled and stored in threshold-secret-shared form. This is precisely the same requirement as exists for the secret key in many other threshold signature schemes, including Schnorr and ECDSA. We use the following functionality to accomplish it in a modular fashion.

Functionality 3.6. $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The adversary $\mathcal{S}$ may corrupt up to $t - 1$ parties that are indexed by $\mathbf{P}^*$, and if $|\mathbf{P}^*| \geq 1$, then the adversary $\mathcal{S}$ may instruct the functionality to abort.

Key Generation: On receiving $(\texttt{keygen}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that sid is fresh, send $(\texttt{keygen-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{keygen}, \mathsf{sid})$ from all parties, if sid is agreed upon, then

1. Sample $x \leftarrow \mathbb{Z}_q$.
2. Compute $X := x \cdot G$.
3. Receive $(\texttt{poly-points}, \mathsf{sid}, \{p(i)\}_{i \in \mathbf{P}^*})$ from $\mathcal{S}$.
4. Sample a random polynomial p of degree $t-1$ over $\mathbb{Z}_q$, consistent with the values $p(i)$ for $i \in \mathbf{P}^*$ that were sent by $\mathcal{S}$, and subject to $p(0) = x$.
5. For $i \in [n]$, compute $P(i) := p(i) \cdot G$.
6. Send $(\texttt{public-key}, \mathsf{sid}, X, \{P(1), \ldots, P(n)\})$ directly to $\mathcal{S}$.
7. Send $(\texttt{key-pair}, \mathsf{sid}, X, p(i))$ to $\mathcal{P}_i$ for $i \in [n]$ as adversarially-delayed private output.

The realization of $\mathcal{F}_{\mathsf{DLKeyGen}}$ is a well-studied problem. We choose to realize it using the method of Doerner et al. [DKLs19]. While their protocol is simple and self contained, they did not prove that it realizes any standalone functionality, so, for completeness, we do so in Section 9.

Multiplication. 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 is also sometimes known as Oblivious Linear Evaluation (OLE), and it is encapsulated by the following functionality.

Functionality 3.7. $\mathcal{F}_{\mathsf{Mul2P}}(q)$

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

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

Multiplication: On receiving $(\texttt{multiply}, \mathsf{sid}, a)$ from Alice, if $a \in \mathbb{Z}_q$ 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 $d := a \cdot b - c \bmod q$, send $(\texttt{product}, \mathsf{sid}, d)$ to Bob, and store $(\texttt{complete}, \mathsf{sid})$ in memory.

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 $\mathcal{F}_{\mathsf{Mul2P}}(q)$ via the two-round protocol of Doerner et al. [DKLs18]. Their protocol is based upon Oblivious Transfer (OT) [EGL85], and can be seen as a malicious extension of the classic semi-honest distributed multiplication technique of Gilboa [Gil99]. 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.

We note that because the protocol of Doerner et al. uses concretely efficient OT Extensions instead of plain OT, their protocol performs a one-time setup and reuses it for many multiplication operations. This implies that it does not *quite* realize the above functionality: instead it realizes a slightly more complex functionality with an initialization phase followed by many reactive subsessions, each of which performs one multiplication. We elide this detail for the sake of simplicity; all of our protocols and proofs can be adapted to use the true functionality of Doerner et al. in the obvious way, and the required initialization can be performed at the same time as key generation.

4 Threshold BBS+ Protocol

Before we give the formal description of our signing protocol, we give an overview of its subprotocols. These techniques were also overviewed in Section 1.2.

Key Generation. Before servers can sign messages, they must first run a one-time setup phase to jointly generate keys. To generate $X \in \mathbb{G}_2$ along with a Shamir sharing of its discrete logarithm x (that is, a degree-$(t-1)$ polynomial p such that $p(0) = x$ and $x \cdot G_2 = X$), any standard protocol for threshold discrete-log key generation will do. Here we specify that the parties use an ideal functionality $\mathcal{F}_{\mathsf{DLKeyGen}}$, and we give the Doerner et al. [DKLs19] protocol for realizing this functionality in section 9. The servers 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 concurrently with the invocation of $\mathcal{F}_{\mathsf{DLKeyGen}}$ 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

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

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 $r_i\cdot\mathsf{lagrange}(\mathbf{J}, j, 0)\cdot p(j)$ 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 $r_i\cdot e$. For technical reasons, the inputs to the multipliers must be rerandomized by adding secret-sharings of 0 to them.[6] It is possible for the parties to sample secret-sharings of 0 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 $s_0 \leftarrow \mathbb{Z}_q$ and computes $B' \coloneqq s_0\cdot H_1 + \sum_{k\in[\ell]} m_k\cdot H_{k+1}$. 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 $B \coloneqq G_1 + s\cdot H_1 + B'$, and when the client receives the signature, it computes $s' \coloneqq s_0 + s$, 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. [DKLs18, DKLs19]. Thereafter we give protocols for threshold signing and for weak partially-blind threshold signing.

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

[6]This ensures that if the corrupt parties use incorrect inputs for some or all of the multiplication protocol instances, then the offsets induced into the outputs of the honet parties are independent of those parties secret inputs.

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

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 $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2)$, $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q))$, and $\mathcal{F}_{\mathsf{Mul2P}}(q)$. The protocol is parameterized by the threshold t, a message count ℓ, and the description of a bilinear group, $(\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q) = \mathcal{G}$. If at any point during this protocol, a functionality aborts, any party that observes the abort also aborts to the environment. Clients do not abort, but rather output a `failure` message which does not cause the other parties to halt or ignore further instructions from the environment.

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

1. $\mathcal{P}_i$ sends $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q))$.
2. $\mathcal{P}_i$ 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 $\mathcal{F}_{\mathsf{Com}}$.
3. Upon being notified of all other parties' commitments, $\mathcal{P}_i$ sends $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{Com}}$.
4. If $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q))$ aborts, then $\mathcal{P}_i$ aborts.
5. On receiving $(\texttt{key-pair}, \mathsf{sid}, X, p(i))$ from $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q))$ and $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_j, \mathbf{D}_j)$ from $\mathcal{F}_{\mathsf{Com}}$ for every $j \in [n] \setminus \{i\}$, $\mathcal{P}_i$ computes
$$\mathbf{H} := \left\{ \sum_{j\in[n]} D_{j,k} \right\}_{k\in[\ell+1]}$$
and then $\mathcal{P}_i$ 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 $|\mathbf{J}| = t$.

6. $\mathcal{C}$ broadcasts[b] $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to every $\mathcal{P}_j$ for $j \in \mathbf{J}$. On receiving this message, each party $\mathcal{P}_j$ for $j \in \mathbf{J}$ outputs $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \mathbf{m}, \mathbf{J})$ to the environment.
7. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from some client $\mathcal{C}$ and $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$ from the environment, $\mathcal{P}_i$ sends $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{C}$ and to $\mathcal{P}_j$ for $j \in \mathbf{J} \setminus \{i\}$.

8. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, $\mathcal{P}_i$ ignores all further instructions related to this sigid. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, $\mathcal{C}$ outputs $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to the environment and ignores all further instructions related to this sigid.

9. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from some client $\mathcal{C}$ and $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ from the environment, $\mathcal{P}_i$ samples $(e_i, s_i, r_i) \leftarrow \mathbb{Z}_q^3$ and sends $(\texttt{sample}, \mathsf{sid}\|\mathsf{sigid}, \mathbf{J})$ to $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2)$. On receiving $(\texttt{zero-share}, \mathsf{sid}\|\mathsf{sigid}, (\alpha_i, \beta_i))$ from $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2)$, $\mathcal{P}_i$ computes $x_i := \mathsf{lagrange}(\mathbf{J}, i, 0) \cdot p(i)$, and then $\mathcal{P}_i$ 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 $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j, x_i + \alpha_i)$ to $\mathcal{F}_{\mathsf{Mul2P}}(q)$ and $(\texttt{sync}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m})$ to $\mathcal{P}_j$ for every $j \in \mathbf{J} \setminus \{i\}$.

10. Upon receiving $(\texttt{committed}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{ready}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, \mathcal{P}_j, c_{i,j})$ from $\mathcal{F}_{\mathsf{Mul2P}}(q)$ for every $j \in \mathbf{J} \setminus \{i\}$, sends $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid})$ to $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{multiply}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, r_i + \beta_i)$ to $\mathcal{F}_{\mathsf{Mul2P}}(q)$ for every $j \in \mathbf{J} \setminus \{i\}$.

11. 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 $\mathcal{F}_{\mathsf{Mul2P}}(q)$ for every $j \in \mathbf{J} \setminus \{i\}$, $\mathcal{P}_i$ computes

$$\begin{aligned}
e &:= \sum_{j\in[n]} e_j \\
s &:= \sum_{j\in[n]} s_j \\
B &:= G_1 + s \cdot H_1 + \sum_{k\in[\ell]} m_k \cdot H_{k+1} \\
R_i &:= r_i \cdot B \\
u_i &:= (r_i + \beta_i) \cdot (e + x_i + \alpha_i) + \sum_{j\in\mathbf{J}\setminus\{i\}} (c_{i,j} + d_{i,j})
\end{aligned}$$

and sends $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i)$ to $\mathcal{C}$, and halts.

12. Upon receiving $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i)$ from every $\mathcal{P}_i$ for $i \in \mathbf{J}$, the client $\mathcal{C}$ outputs $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ 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 11, 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 $(\mathtt{signature}, \mathsf{sid}, \mathsf{sigid}, (A, e, s), \mathsf{pk})$ to $\mathcal{Z}$, and halts, but if the equality does not hold, then $\mathcal{C}$ outputs $(\mathtt{failure}, \mathsf{sid}, \mathsf{sigid})$.

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

13. $\mathcal{C}$ samples $s_0 \leftarrow \mathbb{Z}_q$, computes

$$B' := s_0 \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1}$$

and sends $(\mathtt{prove}, \mathsf{sid}\|\mathsf{sigid}, \{\mathcal{P}_j\}, \{B', H_1, \ldots, H_{\ell+1}\}, \{s_0, m_1, \ldots, m_\ell\})$ to $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_j}$ for $j \in \mathbf{J}$. $\mathcal{C}$ also broadcasts[b] $(\mathtt{wb\text{-}sig\text{-}req}, \mathsf{sid}, \mathsf{sigid}, B', \mathbf{J})$ to every party $\mathcal{P}_j$ for $j \in \mathbf{J}$. On receiving this message and recieving $(\mathtt{accepted}, \mathsf{sid}\|\mathsf{sigid}, \mathcal{C}, \{B', H_1, \ldots, H_{\ell+1}\})$ from $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_j}$, each party $\mathcal{P}_j$ for $j \in \mathbf{J}$ outputs $(\mathtt{wb\text{-}sig\text{-}req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \phi_j, \mathbf{J})$ to the environment.

14. On receiving $(\mathtt{wb\text{-}sig\text{-}req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{J})$ from some client $\mathcal{C}$ and $(\mathtt{accepted}, \mathsf{sid}, \mathcal{C}, \{B', H_1, \ldots, H_{\ell+1}\})$ from $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_i}$ and $(\mathtt{reject}, \mathsf{sid}, \mathsf{sigid})$ from the environment, $\mathcal{P}_i$ sends $(\mathtt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{C}$ and to $\mathcal{P}_j$ for $j \in \mathbf{J} \setminus \{i\}$.

15. On receiving $(\mathtt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, $\mathcal{P}_i$ ignores all further instructions related to this sigid. On receiving $(\mathtt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_j$ for $j \in \mathbf{J}$, $\mathcal{C}$ outputs $(\mathtt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to the environment and ignores all further instructions related to this sigid.

16. On receiving $(\mathtt{wb\text{-}sig\text{-}req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{J})$ from some client $\mathcal{C}$ and $(\mathtt{accepted}, \mathsf{sid}, \mathcal{C}, \{B', H_1, \ldots, H_{\ell+1}\})$ from $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_i}$ and $(\mathtt{accept}, \mathsf{sid}, \mathsf{sigid})$ from the environment, each party $\mathcal{P}_i$ runs Steps 9 to 11 from the **Signing** phase, except that B is computed as

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

As before, each party $\mathcal{P}_i$ sends $(\mathtt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i)$ to $\mathcal{C}$ before halting.

17. Upon receiving $(\mathtt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R_i, u_i)$ from every $\mathcal{P}_i$ for $i \in \mathbf{J}$, the client $\mathcal{C}$ outputs $(\mathtt{failure}, \mathsf{sid}, \mathsf{sigid})$ 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 16, computes

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

$$s' := s_0 + s$$

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

[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 2 through 5. 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.

[b] As we mentioned in section 3.1, our broadcast channel can be realized via the echo-broadcast technique [GL05], with no loss in security and no significant loss in efficiency.

Theorem 4.2 (Main Security Theorem)**.** *Let $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$ be the description of a bilinear group. $\pi_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ statistically UC-realizes $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}\wedge\phi}}, \mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q)), \mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2), \mathcal{F}_{\mathsf{Mul2P}}(q))$-hybrid model with selective abort against a malicious adversary that statically corrupts up to $t-1$ fixed parties and any number of transient clients.*

Proof is given in Section 8.

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. [CDL16] construct a concretely efficient proof of the following relation:

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

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 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 via an appropriate choice of predicate in weak paritially-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 $e_0 \leftarrow \mathbb{Z}_q$ and initiates an instance of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ 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 and compute $u'_i := u_i + c_{i,0}$. $\mathcal{P}_i$ sends u'_i to the client in place of u_i, and completes the multiplier instance at the same time, supplying r_i as its input. Thus the client receives $d_{0,i}$ from $\mathcal{F}_{\mathsf{Mul2P}}(q)$ such that $c_{i,0} + d_{0,i} = e_0 \cdot r_i$ for every $i \in \mathbf{J}$

along with R_i and u'_i, computes

$$A' := \frac{\sum\limits_{i \in \mathbf{J}} R_i}{\sum\limits_{i \in \mathbf{J}} (u'_i + d_{0,i})} \qquad \text{and} \qquad e' := e_0 + e$$

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 8 extends naturally.

Note that because the multiplication protocol of Doerner et al. [DKLs18] (with which we propose to realize $\mathcal{F}_{\mathsf{Mul2P}}(q)$) 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 [MR19], 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 Shorter Threshold BBS+ Signatures

The recent work of Tessaro and Zhu [TZ23] proved the security of a shorter variant of BBS+ that eliminates the nonce s and reduces the public key from $(\{H_1, \dots, H_{\ell+1}\}, X)$ to $(\{H_1, \dots, H_\ell\}, X)$. A short BBS+ signature on messages $m_1, \dots, m_\ell$ is of the form (A, e) where

$$A := \frac{G_1 + \sum_{k \in [\ell]} m_k \cdot H_k}{x + e}$$

Our protocol can easily be adjusted to accommodate this shorter BBS+ scheme. For plain signing, the adjustment is direct. For weak partially-blind signing, the client must sample a masking nonce $r_0 \leftarrow \mathbb{Z}_q$ and compute

$$B' := r_0 \cdot \Big(G_1 + \sum_{k \in [\ell]} m_k \cdot H_k\Big)$$

and adjust the language of $\mathcal{F}_{\mathsf{ZK}}$ as necessary. The signing parties skip the steps related to s. The client, upon receiving e and shares R_i and u_i from the signers, computes

$$A := \frac{\sum\limits_{i \in \mathbf{J}} R_i}{r_0 \cdot \sum\limits_{i \in \mathbf{J}} u_i}$$

and takes the signature to be (A, e). Strong partially-blind signing can be achieved as described in Section 5.1 using pairwise instances of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ among the client and each of the signing servers.

5.3 Oblivious Threshold VRF Evaluation

Dodis and Yampolskiy [DY05] 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 $\pi = G_1/(x + e)$. Starting from its strongly-blind form as described in Section 5.1, our protocol can be lightly modified to serve 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.

Specifically, the client receives e as input from the environment, and then samples $e_0, e_{\mathbf{J}_1} \ldots, e_{\mathbf{J}_t}$ to be a uniform additive secret sharing of e, and communicates each e_i privately to $\mathcal{P}_i$ for $i \in \mathbf{J}$ along with the evaluation request. The servers use G_1 in place of B, and no coin tossing need be done for either s or e. The resulting protocol realizes a functionality that simply computes the VRF F_x on the client's input, much as our unmodified protocol simply computes a signature on the client's messages.

5.4 Proactive Security

Ostrovsky and Yung [OY91] 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. [HJKY95] 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-of-n ECDSA construction of Kondi et al. [KMOS21] and we are able to apply their technique directly to proactivize our scheme when $t = 2$. For general t-of-n proactivization, the parties can simply re-run the setup phase of the OT extension protocol to create fresh OT correlations, and refresh their shares of the signing key via the standard technique of jointly sampling Shamir shares of 0 and adding these to the shares with which they started.

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.* [DKLs18], but to realize the underlying OT-extension, we use the new SoftSpokenOT protocol [Roy22b] in place of the KOS protocol they suggested, along with the Endemic OT protocol of Masny and Rindal [MR19] 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 [Roy22a]. 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 [Roy22a] this yields a strict improvement over KOS.

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

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

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 [MR19] 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 [Fis05] or Kondi-shelat transform [KS22]. The cost of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ 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 subprotocol costs. We did not calculate the cost of weak partially-blind signing, since it depends heavily on the predicates that the client proves, and the protocol used to prove them. Our SetupCost function combines the cost of key generation, multiplier setup, and fixing shared 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, and that all commitments are coalesced where possible.

$$\begin{aligned}
&\mathsf{SetupCost}(n,\ell,\lambda,\kappa,|G_1|,|G_2|)\mapsto\\
&\quad(n-1)\cdot(8\lambda+\kappa+(2\cdot|G_2|+\kappa)\cdot\lambda/\log_2\lambda+|G_1|+\mathsf{MulSetupCost}(\lambda,|G_1|))\\
&\mathsf{SignCost}(n,t,\ell,\lambda,s,\kappa,|G_1|,|G_2|)\mapsto\\
&\quad(t-1)\cdot(4\lambda+2\kappa+2\cdot\mathsf{MulCost}(\lambda,\kappa,s))+3\kappa+|G_2|+|G_1|\cdot(\ell+2)+t\log n
\end{aligned}$$

Finally, the client's signing request involves transmitting $t\cdot(\ell\cdot\kappa+\log n)$ 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 [SKSW20] of The Internet Engineering Task Force (IETF) for "pairing-friendly" elliptic curves are the BLS12_381 and BN462 [Bow17] curves, corresponding to a 128-bit security level, and the BLS48_581 [KIK+17] 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.

Curve	κ	q	$\|G_1\|$	$\|G_2\|$	Security Level
BLS12_381	256	$\sim 2^{255}$	384	768	126 [GMT20]
BN462	464	$\sim 2^{461}$	464	928	134 [GMT20]
BLS48_581	520	$\sim 2^{517}$	584	4672	256

Table 6.1: IETF-Recommended Pairing Curve Specifications: Group order bit-length (κ), group order (q), and group element sizes for curves BLS12_381, BN462, and BLS48_581 and their corresponding security levels. Sizes are rounded up to the nearest even byte.

Curve	λ	Mul Cost	Setup Cost	Signing Cost	Response Size
BLS12_381	128	308576	$(n-1)\cdot 132736$	$(t-1)\cdot 618176$	$1536+(\ell+2)\cdot 384$
BN462	128	821192	$(n-1)\cdot 163159$	$(t-1)\cdot 1643820$	$2320+(\ell+2)\cdot 464$
BLS48_581	256	1128196	$(n-1)\cdot 617808$	$(t-1)\cdot 2258456$	$6232+(\ell+2)\cdot 584$

Table 6.2: Bandwidth Costs in total bits transmitted per party, for t parties (out of n) who wish to sign a vector of ℓ messages. Note that we use the standard computational security parameters with values closest to those of the chosen curve, and that in all cases, the statistical parameter $s = 80$. The *Signing Cost* is the number of bits sent per signer to the other signers, whereas the *Response Size* is the number of bits sent per signer to the client.

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)`.

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 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 7.1. 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 7.2.

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 7.3. 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.

Comparison with Doerner et al.'s Threshold ECDSA [DKLs19]. 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

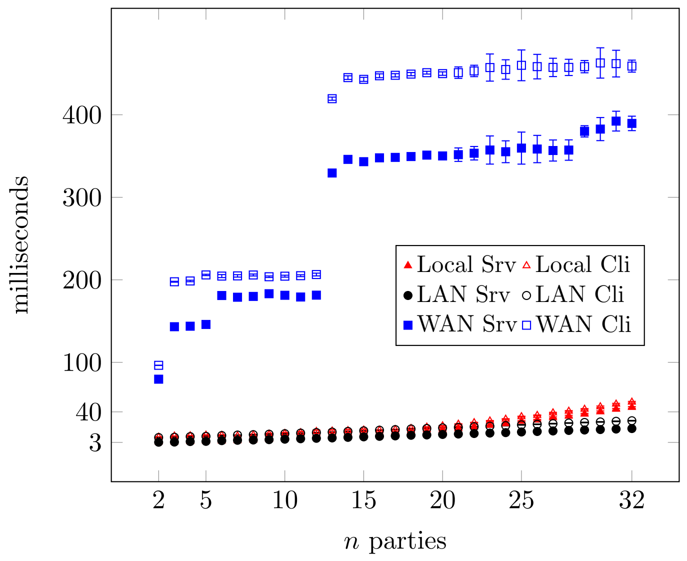

Figure 7.1: Protocol signing timings for n-out-of-n over Local, LAN and WAN setups, as expected in a practical deployment for credentials. Error bars depict standard deviation over 150+ runs. The server time reflects the back-end computation, measured as the max average over the servers from the time 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.

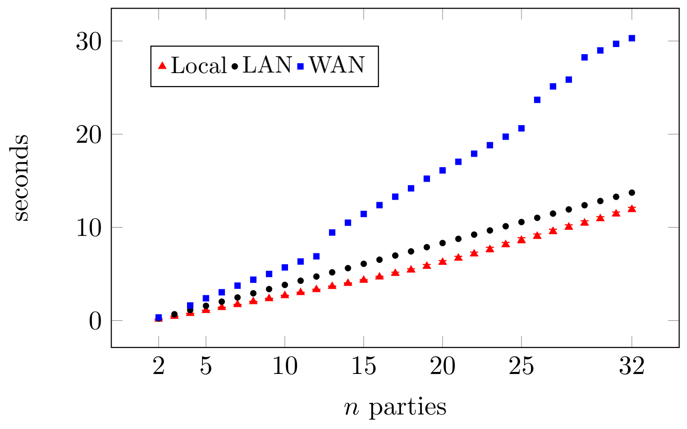

Figure 7.2: Protocol setup timing for n-out-of-n over Local, LAN and WAN setups. Error bars depict standard deviation over 150+ runs.

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 mul-

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 7.3: 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.

tipliers 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 [GGI19]. 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 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 [RP22]. We used Nym's implementation [Nym22] 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*. This favors their protocol over ours.

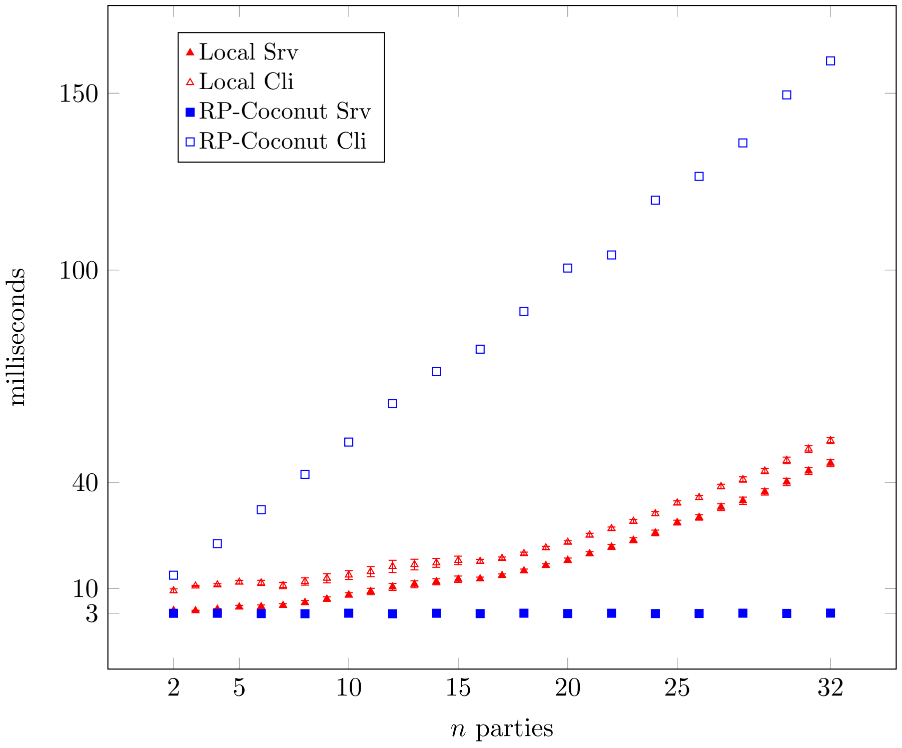

Figure 7.4: Protocol running times for n-out-of-n signing 3 messages over Local network environment for our protocol versus RP-Coconut. Timings for RP-Coconut were taken by the criterion package, with 100 samples.

As expected, Figure 7.4 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.

8 Proof of Security for t-of-n Signing

Theorem 4.2. *Let $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$ be the description of a bilinear group. $\pi_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ statistically UC-realizes $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi}, \mathcal{F}_{\mathsf{DLKeyGen}}(n, t, (\mathbb{G}_2, G_2, q)), \mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2), \mathcal{F}_{\mathsf{Mul2P}}(q))$-hybrid model with selective abort against a malicious adversary that statically corrupts up to $t-1$ fixed parties and any number of transient clients.*

Our proof relies on a few crucial details of the underlying signature scheme and usage model. In particular, we use the fact that the inversion nonce of the Bar-Ilan and Beaver protocol can be interpreted as a MAC on the adversary's shares of the secret key, which is checked via the reconstruction procedure.

The high-level idea is to abstract away the details of the adversarial behavior in terms of our "offset" values that the adversary *induces* on the inputs to the multipliers in the protocol. Because the adversary's input relates to x_i, r_i, R_i, and u_i, we denote the adversary's offsets that the corrupt parties cumulatively inject into these values by $\boldsymbol{\delta}^x$, $\boldsymbol{\delta}^r$, Δ^R, and Δ^u respectively. While the structure of our protocol makes it easy for us to identify these offsets, the difficulty lies in analyzing how these offsets should be used to simulate the protocol outcome.

We must discriminate between offsets that are malicious and those that are benign, i.e. those that cancel each other out in the real protocol and do not cause an abort. We demonstrate how the signature verification algorithm can be used to check whether the adversary's cheating offsets *cancel one another*, even without knowledge of the correct signature value.

The most subtle aspect of our simulation involves programming a *different* (but identically distributed) signature into the view of a corrupt client, relative to the one sampled by the functionality. This step is necessary in our proof because the simulator is not able to calculate the offset necessary to program the functionality's signature, and would be trivial to detect if the functionality sent the signature to the servers in addition to the client.

Proof. Formally, we show that for every malicious adversary $\mathcal{A}$ that statically corrupts up to $t-1$ parties, there exists a simulator $\mathcal{S}_{\mathsf{BBS+}}^{\mathcal{A}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$ it holds that

$$\begin{aligned} &\left\{\textsc{Real}_{\pi_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in\mathbb{N}:n>1, t\in[2,n], \ell\in\mathbb{N}:\ell>0,\\ \mathcal{G}\leftarrow\mathsf{BilinGen}(1^\lambda), z\in\{0,1\}^{\mathsf{poly}(\lambda)}}} \\ &\quad\approx_{\mathsf{s}} \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell),\mathcal{S}_{\mathsf{BBS+}}^{\mathcal{A}}(n,t,\mathcal{G},\ell),\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N}, n\in\mathbb{N}:n>1, t\in[2,n], \ell\in\mathbb{N}:\ell>0,\\ \mathcal{G}\leftarrow\mathsf{BilinGen}(1^\lambda), z\in\{0,1\}^{\mathsf{poly}(\lambda)}}} \end{aligned} \tag{1}$$

We begin by giving a description of $\mathcal{S}_{\mathsf{BBS+}}^{\mathcal{A}}(n, t, \mathcal{G}, \ell)$ and proceed via a sequence of hybrid experiments.

Simulator 8.1. $\mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$

This simulator is parameterized by a party count n, a threshold t, a message count ℓ, and the description of a bilinear group, $(\mathbb{G}_1,\mathbb{G}_2,G_1,G_2,q)=\mathcal{G}$. It has black-box access to an adversary $\mathcal{A}$ for the protocol $\pi_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$, which statically corrupts up to $t-1$ of the fixed parties $\mathcal{P}_1,\ldots,\mathcal{P}_n$. $\mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$ simulates an instance of the real-world experiment involving $\pi_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$ to $\mathcal{A}$, and forwards all messages that it receives from $\mathcal{Z}$ to $\mathcal{A}$ and vice versa. When $\mathcal{A}$ announces which parties it would like to corrupt, the simulator corrupts the same parties in its own ideal experiment. If $\mathcal{A}$ corrupts fewer than $t-1$ of the fixed parties in the simulated experiment, then the simulator arbitrarily chooses $n-(t-1)$ of the honest parties, denoted by $\overline{\mathbf{P}^*}$, and corrupts *all* other fixed parties in the ideal world. The simulator simulates honest parties other than those indexed by $\overline{\mathbf{P}^*}$ to $\mathcal{A}$ by running their protocol code honestly and forwarding any environment interactions they may produce to $\mathcal{Z}$ on their behalf. Thus, in the simulated experiment, we can let $\mathbf{P}^* := [n]\setminus\overline{\mathbf{P}^*}$.

Setup:

1. On receiving either $(\texttt{keygen},\mathsf{sid})$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n,t,\mathcal{G})$ or $(\texttt{commit},\mathsf{sid}\|\mathcal{P}_i,\{\mathcal{P}_1,\ldots,\mathcal{P}_n\},\mathbf{D}_i)$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ from $\mathcal{P}_i$ for some $i\in\mathbf{P}^*$, send $(\texttt{init},\mathsf{sid})$ to $\mathcal{F}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$ on behalf of $\mathcal{P}_i$.

2. On receiving $(\texttt{keygen},\mathsf{sid})$ from $\mathcal{P}_i$ for some $i\in\mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G},n,t)$, send $(\texttt{keygen-req},\mathsf{sid},i)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n,t,\mathcal{G})$.

3. On receiving $(\texttt{init-req},\mathsf{sid},j)$ for some $j\in\overline{\mathbf{P}^*}$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$, send $(\texttt{keygen-req},\mathsf{sid},j)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n,t,\mathcal{G})$ and send $(\texttt{committed},\mathsf{sid}\|\mathcal{P}_j,\mathcal{P}_j)$ to $\mathcal{P}_i$ for every $i\in\mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}$.

4. On receiving $(\texttt{public-key},\mathsf{sid},\mathsf{pk})$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell)$ and $(\texttt{keygen},\mathsf{sid})$ from $\mathcal{P}_i$ for every $i\in\mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G},n,t)$,

   a. Parse $(\mathbf{H},X) := \mathsf{pk}$.

   b. Wait to receive $(\texttt{poly-points},\mathsf{sid},\{p(i)\}_{i\in\mathbf{P}^*})$ directly from $\mathcal{A}$ (and if $\mathcal{A}$ has corrupted fewer than $t-1$ parties, then sample the missing values uniformly).

   c. For every $i\in\mathbf{P}^*$, compute $P(i) := p(i)\cdot G_2$.

   d. For every $j\in\overline{\mathbf{P}^*}$, compute

$$P(j) := \frac{X-\sum_{i\in\mathbf{P}^*}\mathsf{lagrange}(\mathbf{P}^*\cup\{j\},i,0)\cdot P(i)}{\mathsf{lagrange}(\mathbf{P}^*\cup\{j\},j,0)}$$

e. Send $(\texttt{public-key}, \mathsf{sid}, X, \{P(1), \ldots, P(n)\})$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$.

5. On receiving $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ and $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \mathbf{D}_i)$ from $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}$, parse $(\mathbf{H}, X) := \mathsf{pk}$ and sample $\mathbf{D}_j \leftarrow \mathbb{Z}_q^{\ell+1}$ for every $j \in \overline{\mathbf{P}^*}$ subject for every $k \in [\ell + 1]$ to
$$\sum_{j \in [n]} D_{j,k} = H_k$$
and send $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_j, \mathbf{D}_j)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$ and $j \in \overline{\mathbf{P}^*}$ on behalf of $\mathcal{F}_{\mathsf{Com}}$.

6. On being instructed by $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ to release the output to $\mathcal{P}_i$ for some $i \in \mathbf{P}^*$, send $(\texttt{key-pair}, \mathsf{sid}, X, p(i))$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$

7. On being instructed by $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ to release the output to $\mathcal{P}_j$ for some $j \in \overline{\mathbf{P}^*}$, and receiving $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_i)$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ from $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$, instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to release the public key to $\mathcal{P}_j$.

8. On being instructed by $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ to abort, instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to abort and ignore all future instructions with the same sid.

Signing, with an honest Client $\mathcal{C}$:

9. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \mathbf{m}, \mathbf{J})$ from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$ for some $i \in \mathbf{J}^*$, send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to $\mathcal{P}_i$ on behalf of $\mathcal{C}$. Let $\mathbf{J}^* := \mathbf{J} \cap \mathbf{P}^*$ and let $\overline{\mathbf{J}^*} := \mathbf{J} \cap \overline{\mathbf{P}^*}$.

10. When $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ attempts to send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \mathbf{m}, \mathbf{J})$ to some honest party, instruct it to release its message immediately.

11. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from some $\mathcal{P}_i$ for $i \in \mathbf{J}^*$, send $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$ and ignore future messages from $\mathcal{P}_i$ with this sigid.

12. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid}, j)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, send $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$ on behalf of $\mathcal{P}_j$ and ignore future messages with this sigid.

13. On receiving $(\texttt{sample}, \mathsf{sid}\|\mathsf{sigid}, \mathbf{J})$ from $\mathcal{P}_i$ for $i \in \mathbf{J}^*$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2)$, sample $(\alpha_i, \beta_i) \leftarrow \mathbb{Z}_q^2$ and respond with $(\texttt{zero-share}, \mathsf{sid}\|\mathsf{sigid}, (\alpha_i, \beta_i))$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(q)$.

14. On receiving $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid}, \{\mathcal{P}_j\}_{j\in\mathbf{J}}, (e_i, s_i))$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j, x'_{i,j})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ from $\mathcal{P}_i$ for some $i \in \mathbf{J}^*$ for every $j \in \overline{\mathbf{J}^*}$, send $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$.

15. On receiving $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, sample $c_{i,j} \leftarrow \mathbb{Z}_q$ uniformly and send $(\texttt{committed}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j)$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{ready}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, \mathcal{P}_j, c_{i,j})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ to $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$.

16. On receiving $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ for every $j \in \mathbf{J}^*$, and receiving $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid}, \{\mathcal{P}_j\}_{j\in\mathbf{J}}, (e_i, s_i))$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_k\|\mathsf{sigid}, \mathcal{P}_k, x'_{i,k})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ from $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$ and some $k \in \overline{\mathbf{J}^*}$,

- If $\mathcal{P}_k$ is not the last honest party for which these conditions hold, then sample $e_k \leftarrow \mathbb{Z}_q$ and $s_k \leftarrow \mathbb{Z}_q$ uniformly.
- If $\mathcal{P}_k$ is the last honest party for which these conditions hold, then wait to receive $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, e, s)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, and then compute

$$e_k := e - \sum_{j\in\mathbf{J}\setminus\{k\}} e_j \qquad \text{and} \qquad s_k := s - \sum_{j\in\mathbf{J}\setminus\{k\}} s_j$$

and then for every $i \in \mathbf{J}^*$ sample $d_{i,k} \leftarrow \mathbb{Z}_q$ uniformly and send to $\mathcal{P}_i$ $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_k\|\mathsf{sigid}, (e_k, s_k))$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{product}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_k\|\mathsf{sigid}, d_{i,k})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$.

17. On receiving $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{multiply}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, r'_{i,j})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ and $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, e, s, R'_i, u'_i)$ on behalf of $\mathcal{C}$ from $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$ and $j \in \overline{\mathbf{J}^*}$,

   a. Choose $h \in \overline{\mathbf{J}^*}$ arbitrarily then for every $j \in \overline{\mathbf{J}^*}$ compute

$$\delta^r_j := \sum_{i\in\mathbf{J}^*} (r'_{i,j} - r'_{i,h})$$
$$\delta^x_j := \sum_{i\in\mathbf{J}^*} \left(x'_{i,j} - \mathsf{lagrange}(\mathbf{J}, i, 0) \cdot p(i) - \alpha_i\right)$$

   b. Compute B via the same equation as in Step 11 of $\pi_{\mathsf{BBS+}}$, and then compute

$$\begin{aligned}\delta^u &:= \sum_{i\in\mathbf{J}^*}\Big(u'_i - \big(\sum_{j\in\overline{\mathbf{J}^*}} c_{i,j} + d_{i,j}\big)\Big)\\ &\quad - \Big(e + \sum_{i\in\mathbf{J}^*}(\alpha_i + \mathsf{lagrange}(\mathbf{J}, i, 0)\cdot p(i))\Big)\cdot\sum_{i\in\mathbf{J}^*} r'_{i,h}\\ \Delta^R &:= \sum_{i\in\mathbf{J}^*}\left(R'_i + \beta_i\cdot B - r'_{i,h}\cdot B\right)\\ \Delta^A &:= \frac{\Delta^R}{\delta^u}\end{aligned}$$

c. If only one of δ^u and Δ^R is nonzero, or if both are nonzero and $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (\Delta^A, e, s)) \neq 1$, or if there is any $j \in \overline{\mathbf{J}^*}$ such that $\delta^r_j \neq 0$ or $\delta^x_j \neq 0$, or if there is any disagreement among the corrupt parties on the values of pk, e, or s, then instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to output a failure to $\mathcal{C}$. Otherwise, instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to release its output to $\mathcal{C}$.

Signing, against a corrupt Client $\mathcal{C}^*$:

18. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ from $\mathcal{C}^*$ on behalf of some honest party $\mathcal{P}_j$ for $j \in \mathbf{J}\cap\overline{\mathbf{P}^*}$, if this is the first honest party on behalf of whom such a message was received, then send $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{C}^*$; regardless, instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to release its message $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \mathbf{m}, \mathbf{J})$ to $\mathcal{P}_j$. Let $\mathbf{J}^* := \mathbf{J}\cap\mathbf{P}^*$ and let $\overline{\mathbf{J}^*} := \mathbf{J}\cap\overline{\mathbf{P}^*}$.

19. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ from $\mathcal{P}_i$ for $i \in \mathbf{J}^*$ on behalf of some $\mathcal{P}_j$ for $j \in \overline{\mathbf{J}^*}$, send $(\texttt{reject}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$, and ignore future messages with this sigid from $\mathcal{P}_i$.

20. On receiving $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid}, j)$ from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ for some $j \in \overline{\mathbf{J}^*}$, send $(\texttt{rejected}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{C}^*$ and to $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$, and ignore future messages with this sigid.

21. On receiving $(\texttt{sample}, \mathsf{sid}\|\mathsf{sigid}, \mathbf{J})$ from $\mathcal{P}_i$ for $i \in \mathbf{J}^*$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(n, \mathbb{Z}_q^2)$, sample $(\alpha_i, \beta_i) \leftarrow \mathbb{Z}_q^2$ and respond with $(\texttt{zero-share}, \mathsf{sid}\|\mathsf{sigid}, (\alpha_i, \beta_i))$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(q)$.

22. On receiving $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid}, \{\mathcal{P}_j\}_{j\in\mathbf{J}}, (e_i, s_i))$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j, x'_{i,j})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ from $\mathcal{P}_i$ for some $i \in \mathbf{J}^*$ for every $j \in \overline{\mathbf{J}^*}$, send $(\texttt{accept}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$.

23. On receiving $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ for some $j \in \overline{\mathbf{J}^*}$, sample $c_{i,j} \leftarrow \mathbb{Z}_q$ uniformly and send $(\texttt{committed}, \mathsf{sid}\|\mathcal{P}_j\|\mathsf{sigid}, \mathcal{P}_j)$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{ready}, \mathsf{sid}\|\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sigid}, \mathcal{P}_j, c_{i,j})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ to $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$.

24. On receiving $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, j)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ for every $j \in \mathbf{J}^*$, and receiving $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid}, \{\mathcal{P}_j\}_{j\in\mathbf{J}}, (e_i, s_i))$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{input}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_k\|\mathsf{sigid}, \mathcal{P}_k, x'_{i,k})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ from $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$ and some $k \in \overline{\mathbf{J}^*}$,

   - If $\mathcal{P}_k$ is not the last honest party for which these conditions hold, then sample $e_k \leftarrow \mathbb{Z}_q$ and $s_k \leftarrow \mathbb{Z}_q$ uniformly.
   - If $\mathcal{P}_k$ is the last honest party for which these conditions hold, then wait to receive $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, e, s)$ directly from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, and then

     a. For every $j \in \overline{\mathbf{J}^*}$, compute

     $$\delta^x_j := \sum_{i\in\mathbf{J}^*} (\mathsf{lagrange}(\mathbf{J}, i, 0) \cdot p(i) + \alpha_i - x'_{i,j})$$

     b. Let $\hat{e} := e - \delta^x_k$.

     c. Compute

     $$e_k := \hat{e} - \sum_{j\in\mathbf{J}\setminus\{k\}} e_j \qquad \text{and} \qquad s_k := s - \sum_{j\in\mathbf{J}\setminus\{k\}} s_j$$

   and then for every $i \in \mathbf{J}^*$ sample $d_{i,k} \leftarrow \mathbb{Z}_q$ uniformly and send to $\mathcal{P}_i$ $(\texttt{decommitment}, \mathsf{sid}\|\mathcal{P}_k\|\mathsf{sigid}, (e_k, s_k))$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{product}, \mathsf{sid}\|\mathcal{P}_i\|\mathcal{P}_k\|\mathsf{sigid}, d_{i,k})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$.

25. On receiving $(\texttt{decommit}, \mathsf{sid}\|\mathcal{P}_i\|\mathsf{sigid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and $(\texttt{multiply}, \mathsf{sid}\|\mathcal{P}_k\|\mathcal{P}_i\|\mathsf{sigid}, r'_{i,k})$ on behalf of $\mathcal{F}_{\mathsf{Mul2P}}(q)$ from $\mathcal{P}_i$ for every $i \in \mathbf{J}^*$ and some $k \in \overline{\mathbf{J}^*}$,

   - If $\mathcal{P}_k$ is not the last honest party for which these conditions hold, then sample $R_k \leftarrow \mathbb{G}_1$ and $\hat{u}_k \leftarrow \mathbb{Z}_q$ uniformly.
   - If $\mathcal{P}_k$ is the last honest party for which these conditions hold, then

     a. Compute B via the same equation as in step 11 of $\pi_{\mathsf{BBS+}}$.

     b. For every $j \in \overline{\mathbf{J}^*}$, compute

     $$\delta^r_j := \sum_{i\in\mathbf{J}^*} (r'_{i,k} - r'_{i,j})$$

c. Sample $u \leftarrow \mathbb{Z}_q$ uniformly and compute

$$R_k := u \cdot A - \sum_{i \in \mathbf{J}^*} (r'_{i,k} - \beta_i) \cdot B - \sum_{j \in \overline{\mathbf{J}^*} \setminus \{k\}} R_j$$

d. If there exist some $j, j' \in \overline{\mathbf{J}^*}$ such that $\delta^x_j \neq \delta^x_{j'}$ or $\delta^r_j \neq \delta^r_{j'}$ then sample $u_k \leftarrow \mathbb{Z}_q$ uniformly.

e. If $\delta^x_j = \delta^x_{j'}$ and $\delta^r_j = \delta^r_{j'}$ for every $j, j' \in \overline{\mathbf{J}^*}$, then compute

$$u_k := u - \sum_{j \in \overline{\mathbf{J}^*} \setminus \{k\}} u_j - \sum_{i \in \mathbf{J}^*} \sum_{j \in \overline{\mathbf{J}^*}} (c_{i,j} + d_{i,j}) - \Big(e + \sum_{i \in \mathbf{J}^*} (\alpha_i + \mathsf{lagrange}(\mathbf{J}, i, 0) \cdot p(i))\Big) \cdot \sum_{i \in \mathbf{J}^*} r'_{i,k}$$

and send $(\texttt{accepted}, \mathsf{sid}, \mathsf{sigid}, \mathsf{pk}, \hat{e}, s, R_k, u_k)$ to $\mathcal{C}^*$ on behalf of $\mathcal{P}_k$.

Weak Partially-Blind Signing, with honest Client $\mathcal{C}$:

26. On receiving $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \phi_i, \mathbf{J})$ from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{P}_i$ for some $i \in \mathbf{J}^*$, if $\mathcal{P}_i$ is the first corrupt party on behalf of whom such a message has been received, then sample $B' \leftarrow \mathbb{G}_1$ unformly, and broadcast $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, B', \mathbf{J})$ to the parties indexed by $\mathbf{J}$ on behalf of $\mathcal{C}$. Regardless, send $(\texttt{accepted}, \mathsf{sid}\|\mathsf{sigid}, \mathcal{C}, \{B', H_1, \ldots, H_{\ell+1}\})$ on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_i}$ to $\mathcal{P}_i$. Let $\mathbf{J}^* := \mathbf{J} \cap \mathbf{P}^*$ and let $\overline{\mathbf{J}^*} := \mathbf{J} \cap \overline{\mathbf{P}^*}$.

27. When $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ attempts to send $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, \mathcal{C}, \phi_j, \mathbf{J})$ to some honest party $\mathcal{P}_j$, instruct it to release its message immediately.

28. Run steps 11 through 17b of this simulator (from the **Signing, with an honest Client** $\mathcal{C}$ phase), but compute $B := G_1 + s \cdot H_1 + B'$ as specified in step 16 of $\pi_{\mathsf{BBS+}}$ (from the **Weak Partially-Blind Signing** phase). If $\mathsf{e}(\Delta^A,\ X + e \cdot G_2) \neq \mathsf{e}(B,\ G_2)$ or there is any $j \in \overline{\mathbf{J}^*}$ such that $\delta^r_j \neq 0$ or $\delta^x_j \neq 0$ or if there is any disagreement among the corrupt parties on the values of pk, e, or s, then instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to abort. Otherwise, instruct $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to release its output to $\mathcal{C}$.

Weak Partially-Blind Signing, against corrupt Client $\mathcal{C}^*$:

29. On receiving $(\texttt{wb-sig-req}, \mathsf{sid}, \mathsf{sigid}, B', \mathbf{J})$ from $\mathcal{C}^*$ on behalf of party $\mathcal{P}_j$ and $(\texttt{prove}, \mathsf{sid}\|\mathsf{sigid}, \{\mathcal{P}_j\}, \{B', H_1, \ldots, H_{\ell+1}\}, \{s_0, m_1, \ldots, m_\ell\})$ on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}} \wedge \phi_j}$ for some $j \in \mathbf{J} \cap \overline{\mathbf{P}^*}$, such that

$$B' = s_0 \cdot H_1 + \sum_{k \in [\ell]} m_k \cdot H_{k+1}$$

and such that $\phi_j(\mathbf{m}) = 1$, if $\mathcal{P}_j$ is the first honest party for whom these conditions are met, then send $(\texttt{wb-sign}, \mathsf{sid}, \mathsf{sigid}, \mathbf{m}, \mathbf{J})$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{C}^*$; regardless, send $(\texttt{wb-pred}, \mathsf{sid}, \mathsf{sigid}, j, \phi_j)$ to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ on behalf of $\mathcal{C}^*$.

30. Run steps 19 through 25 of this simulator (from the **Signing, against a corrupt Client** $\mathcal{C}^*$ phase), but compute s_k in step 24c as

$$s_k \coloneqq s - s_0 - \sum_{i \in \mathbf{J} \setminus \{k\}} s_i$$

Let $\mathcal{H}_0$ denote the real-world experiment. That is, let

$$\mathcal{H}_0 \coloneqq \left\{ \text{REAL}_{\pi_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell),\mathcal{A},\mathcal{Z}}(\lambda, z) \right\}_{\substack{\lambda \in \mathbb{N}, n \in \mathbb{N}: n>1, t \in [2,n], \ell \in \mathbb{N}: \ell > 0,, \\ \mathcal{G} \leftarrow \mathsf{BilinGen}(1^\lambda), z \in \{0,1\}^{\mathsf{poly}(\lambda)}}}$$

Hybrid $\mathcal{H}_1$. In $\mathcal{H}_1$, we define an initial simulator $\mathcal{S}$ that replaces the honest parties and ideal functionalities in $\mathcal{H}_0$ in their interactions with all other entities. It simulates the honest parties and ideal functionalities by running their code, but learns any values that the corrupt parties send to functionalities in the protocol. After $\mathcal{A}$ announces which parties it would like to corrupt, $\mathcal{S}$ chooses $n - (t - 1)$ honest parties and treats *all* others as corrupt, even though they may not all be under the influence of $\mathcal{A}$. The difference between $\mathcal{H}_1$ and $\mathcal{H}_0$ is purely syntactic; $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In $\mathcal{H}_2$, we alter the code of $\mathcal{S}$ when a signature is requested for an *honest* client $\mathcal{C}$. Specifically, we add Steps 17a and 17b of $\mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ to define δ^x_j and δ^r_j for $j \in \overline{\mathbf{J}^*}$ as well as Δ^R, δ^u, and Δ^A; we also add an instruction to output a failure on behalf of $\mathcal{C}$ if $\delta^x_j \neq 0$ or $\delta^r_j \neq 0$ for any $j \in \mathbf{J}$. There are four potential values that a corrupt $\mathcal{P}_i$ could offset in its interaction with the honest parties in order to cheat. r_i could be used inconsistently in interactions with two different honest parties: we choose one honest party arbitrarily to define the "true" value; we use $r'_{i,j}$ to denote the value (masked by β_i) used in interactions with $\mathcal{P}_j$, and capture the cumulative additive offset induced relative to the true values via δ^r_j. x_i could be used inconsistently relative to the value $p(i)$ supplied by $\mathcal{S}$ on behalf of $\mathcal{F}_{\mathsf{BBS+}}$; we use $x'_{i,j}$ to denote the value (masked by α_i) used in interactions with $\mathcal{P}_j$ and capture the cumulative additive offset induced relative to $\sum_{i \in \mathbf{J}^*} \mathsf{lagrange}(\mathbf{J}, i, 0) \cdot p(i)$ via δ^x_j. Finally, R_i and u_i could be calculated inconsistently relative to the forgoing protocol; the values actually sent to the client are denoted R'_i and u'_i and the offsets that the corrupt parties cumulatively inject into these values are captured by Δ^R and δ^u respectively.

If $\mathcal{A}$ induces nonzero offsets in $\mathcal{H}_1$, then

$$\begin{aligned}\sum_{j\in\overline{\mathbf{J}^*}} u_j + \sum_{i\in\mathbf{J}^*} u'_i \\ &= r\cdot(x+e) + \sum_{j\in\overline{\mathbf{J}^*}} \left(\delta^r_j \cdot (\mathsf{lagrange}(\mathbf{J}, j, 0)\cdot p_j + \alpha_j) + (r_j+\beta_j)\cdot\delta^x_j\right) + \delta^u\end{aligned}$$

and

$$\sum_{j\in\overline{\mathbf{J}^*}} R_j + \sum_{i\in\mathbf{J}^*} R'_i = \sum_{k\in\mathbf{J}} r_k \cdot B + \Delta^R$$

where r_i for $i \in \mathbf{J}^*$ is defined to be the "true" r_i value of $\mathcal{P}_i$ (i.e. the one used when interacting with some arbitrarily chosen honest party).

Observe that fixing e and s fixes A, and so $A = B/(x+e)$ if and only if $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s)) = 1$. Because $\mathcal{C}$ computes

$$A := \frac{\sum_{j\in\overline{\mathbf{J}^*}} R_j + \sum_{i\in\mathbf{J}^*} R'_i}{\sum_{j\in\overline{\mathbf{J}^*}} u_j + \sum_{i\in\mathbf{J}^*} u'_i}$$

the adversary $\mathcal{A}$ avoids a failure on the part of the $\mathcal{C}$ in $\mathcal{H}_1$ if and only if

$$\left(\delta^u + \sum_{j\in\overline{\mathbf{J}^*}} \left(\delta^x_j\cdot(r_j+\beta_j) + \delta^r_j\cdot(\mathsf{lagrange}(\mathbf{J}, j, 0)\cdot p_j + \alpha_j)\right)\right)\cdot A = \Delta^R \quad (2)$$

Thus, $\mathcal{A}$ can distinguish $\mathcal{H}_2$ from $\mathcal{H}_1$ by setting $\delta^x_j \neq 0$ or $\delta^r_j \neq 0$ for any $j \in \overline{\mathbf{J}^*}$ and contriving to make the latter equality hold. However, at the time the adversary must commit to all of its offsets, β_j and α_j are information-theoretically hidden from it. They are both uniformly sampled, and so the adversary's chance of satisfying equation 2 is at most $1/q$. $1/q$ is also the statistical difference between $\mathcal{H}_2$ and $\mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. In $\mathcal{H}_3$, we again alter the code of $\mathcal{S}$ when a signature is requested for an honest client $\mathcal{C}$. Specifically, we remove the instruction to verify whether $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s)) \neq 1$ from step 12 in $\mathcal{C}$'s code in $\pi_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, and replace it with step 17c of $\mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, which produces a failure on behalf of $\mathcal{C}$ if only one of Δ^R and δ^u is nonzero, or if they are both nonzero and $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (\Delta^A, e, s)) \neq 1$.

Observe that fixing e and s fixes A, which $\mathcal{C}$ computes as

$$\frac{\sum_{j\in\overline{\mathbf{J}^*}} R_j + \sum_{i\in\mathbf{J}^*} R'_i}{\sum_{j\in\overline{\mathbf{J}^*}} u_j + \sum_{i\in\mathbf{J}^*} u'_i} = A$$

Rearranging and substituting the definitions of Δ^R and δ^u, this yields

$$\frac{\Delta^R + \sum\limits_{j\in\overline{\mathbf{J}^*}} R_j + \sum\limits_{i\in\mathbf{J}^*} r_i \cdot B}{\delta^u + \sum\limits_{j\in\overline{\mathbf{J}^*}} u_j + \sum\limits_{i\in\mathbf{J}^*} (r_i + \beta_i)\cdot \left(e + \sum\limits_{k\in\mathbf{J}^*}(x_k + \alpha_k)\right) + \sum\limits_{i\in\mathbf{J}^*}\sum\limits_{j\in\overline{\mathbf{J}^*}}(c_{i,j} + d_{i,j})} = A$$

where $x_i = \mathsf{lagrange}(\mathbf{J}, i, 0)\cdot p(i)$ for every $i \in \mathbf{J}^*$. If we define R_j and u_j to be the values a corrupt party $\mathcal{P}_j$ would compute if it were honest (as specified in sstep 11 of the protocol), this simplifies to

$$\frac{\Delta^R + \sum\limits_{i\in\mathbf{J}} R_i}{\delta^u + \sum\limits_{i\in\mathbf{J}} u_i} = A = \frac{\sum\limits_{i\in\mathbf{J}} R_i}{\sum\limits_{i\in\mathbf{J}} u_i}$$

where the second equality follows from the fact that we have defined R_j and u_j ideally.

We can finally conclude that $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s)) = 1$ in $\mathcal{H}_2$ if and only if there is some constant γ such that

$$\Delta^R = \gamma \cdot \sum_{i\in\mathbf{J}} R_i \qquad \text{and} \qquad \delta^u = \gamma \cdot \sum_{i\in\mathbf{J}} u_i$$

and thus that $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s)) = 1$ if and only if either $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (\Delta^A, e, s)) = 1$ or both Δ^R and δ^u are zero. The distribution of $\mathcal{H}_3$ is therefore identical to the distribution of $\mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. In $\mathcal{H}_4$, we implement Steps 17a and 17b of $\mathcal{S}_{\mathsf{BBS+}}$ when a *weak partially-blind* signature is requested for an *honest* client $\mathcal{C}$, and instruct $\mathcal{S}$ to output a failure on behalf of $\mathcal{C}$ if $\delta^x_j \neq 0$ or $\delta^r_j \neq 0$ for any $j \in \mathbf{J}$. $\mathcal{H}_4 \approx_{\mathrm{s}} \mathcal{H}_3$ by the same argument that we presented for $\mathcal{H}_2$.

Hybrid $\mathcal{H}_5$. In $\mathcal{H}_5$, we once again alter the code of $\mathcal{S}$ when a *weak partially-blind* signature is requested by an *honest* client $\mathcal{C}$. We remove the instruction verifying whether $\mathsf{BBS{+}Verify}(\mathsf{pk}, \mathbf{m}, (A, e, s')) = 1$ from step 12 in $\mathcal{C}$'s code in $\pi_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ and instead use the failure condition specified in step 28 of $\mathcal{S}_{\mathsf{BBS+}}$. Specifically, $\mathcal{S}$ now outputs a failure on behalf of $\mathcal{C}$ if only one of Δ^R and δ^u is nonzero, or if they are both nonzero and $\mathsf{e}(\Delta^A,\ X{+}e{\cdot}G_2) \neq \mathsf{e}(B,\ G_2)$. Note that this is equivalent to verifying the signature (Δ^A, e, s'). The distribution of $\mathcal{H}_5$ is therefore identical to the distribution of $\mathcal{H}_4$ under the same argument as we presented for $\mathcal{H}_3$.

Hybrid $\mathcal{H}_6$. In $\mathcal{H}_6$, we alter the code of $\mathcal{S}$ when a signature is requested for a *corrupt* Client $\mathcal{C}^*$ and at least two honest parties participate in the signing process. Specifically, when such a signing request is made, $\mathcal{S}$ calculates δ^x_j and δ^r_j for $j \in \overline{\mathbf{J}^*}$ as specified in steps 24a and 25b of $\mathcal{S}_{\mathsf{BBS+}}$, and if there exist some $j, j' \in \overline{\mathbf{J}^*}$ such that $\delta^x_j \neq \delta^x_{j'}$ or $\delta^r_j \neq \delta^r_{j'}$ then $\mathcal{S}$ samples $u_j \leftarrow \mathbb{Z}_q$ uniformly for $j \in \overline{\mathbf{J}^*}$, rather than calculating these values as the honest parties otherwise would.

In $\mathcal{H}_5$, u_j is calculated for every $j \in \overline{\mathbf{J}^*}$ per step 11 of $\pi_{\mathsf{BBS+}}$ as

$$u_j = (r_j + \beta_j) \cdot (e + x_j + \alpha_j) + \sum_{k \in \mathbf{J} \setminus \{j\}} (c_{j,k} + d_{j,k})$$

and by summing over all of the honest parties and substituting the definitions of $c_{j,k}$ and $d_{j,k}$ and then δ^x_j and δ^r_j this yields

$$\begin{aligned} \sum_{j \in \overline{\mathbf{J}^*}} u_j = & \sum_{j \in \overline{\mathbf{J}^*}} (r_j + \beta_j) \cdot \Big(e + \sum_{k \in \mathbf{J}} (x_k + \alpha_k) + \delta^x_j \Big) \\ & + \sum_{j \in \overline{\mathbf{J}^*}} \Big(\delta^r_j + \sum_{i \in \mathbf{J}^*} (r_i + \beta_i) \Big) \cdot (x_j + \alpha_j) - \sum_{i \in \mathbf{J}^*} \sum_{j \in \overline{\mathbf{J}^*}} (c_{i,j} + d_{i,j}) \end{aligned} \quad (3)$$

We now make three observations. First, in $\mathcal{H}_5$, the individual values u_j for $j \in \overline{\mathbf{J}^*}$ are effectively sampled uniformly subject to equation 3, because each pair of values $(c_{j,j'}, d_{j',j})$ for $j, j' \in \overline{\mathbf{J}^*}$ was sampled uniformly subject to satisfying a fixed relationship. Second, the adversary knows the sums of α_j and β_j over $j \in \overline{\mathbf{J}^*}$, but as before, the individual values are uniform (from the adversary's perspective) subject to these sums. Third, for any $j, j' \in \overline{\mathbf{J}^*}$, equation 3 depends upon $\beta_j \cdot \delta^x_j + \beta_{j'} \cdot \delta^x_j + \beta_{j'} \cdot (\delta^x_{j'} - \delta^x_j)$ and upon $\alpha_j \cdot \delta^\beta_j + \alpha_{j'} \cdot \delta^\beta_j + \alpha_{j'} \cdot (\delta^\beta_{j'} - \delta^\beta_j)$.

Combining our third observation with our second observation, we find that if there is any $j, j' \in \overline{\mathbf{J}^*}$ such that $\delta^\beta_j \neq \delta^\beta_{j'}$ or $\delta^\alpha_j \neq \delta^\alpha_{j'}$, then a uniform offset is induced into $\sum_{j \in \overline{\mathbf{J}^*}} u_j$ in $\mathcal{H}_5$, relative to the value that would be produced if the adversary behaved honestly. Combining this fact with our first observation leads us to the conclusion that if there is any $j, j' \in \overline{\mathbf{J}^*}$ such that $\delta^\beta_j \neq \delta^\beta_{j'}$ or $\delta^\alpha_j \neq \delta^\alpha_{j'}$, then the values u_j are uniform and independent from the adversary's perspective in $\mathcal{H}_5$. The same holds in $\mathcal{H}_6$, and so we have $\mathcal{H}_6 = \mathcal{H}_5$.

Hybrid $\mathcal{H}_7$. In $\mathcal{H}_7$, we alter the code of $\mathcal{S}$ when a signature is requested for a *corrupt* Client $\mathcal{C}^*$. Specifically, we delete the code of the honest parties (retaining only the uniform sampling of u_j under a certain condition that was introduced in $\mathcal{H}_6$), and instead create a signature $(A, e, s) \leftarrow \mathsf{BBS+Sign}(\mathsf{sk}, \mathbf{m})$ using the secret key sampled on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}$, and then use steps 18 through 25 of $\mathcal{S}_{\mathsf{BBS+}}$ to define δ^x and calculate the value $\hat{e}$, and the values e_j, s_j, u_j, and R_j that each $\mathcal{P}_j$ for $j \in \overline{\mathbf{J}^*}$ must transmit. Rather than querying $\mathcal{F}_{\mathsf{BBS+}}$, however, $\mathcal{S}$ generates a signature locally when appropriate using $\mathsf{BBS+Sign}$.

Consider the case that $\delta^r_j = \delta^r_{j'}$ and $\delta^x_j = \delta^x_{j'}$ for all $j, j' \in \overline{\mathbf{J}^*}$; in this case, the uniform sampling of u_j for $j \in \overline{\mathbf{J}^*}$ introduced in $\mathcal{H}_6$ is not done. Furthermore, since there is some j for which $\delta^r_j = 0$ *by definition*, we know that $\delta^r_j = 0$ for *every* $j \in \overline{\mathbf{J}^*}$. The δ^x_j values are interchangeable, and so we can choose an arbitrary $h \in \overline{\mathbf{J}^*}$ and replace all of them with δ^x_h.

If we define R_j and u_j to be the values a corrupt party $\mathcal{P}_j$ would compute if the computation were performed honestly (as specified in Step 11 of the protocol), then in $\mathcal{H}_6$, the honest parties $\mathcal{P}_j$ for $j \in \overline{\mathbf{J}^*}$ computed e_j, s_j, u_j, and R_j such that

$$\frac{\sum\limits_{i\in\mathbf{J}} R_i}{\sum\limits_{i\in\mathbf{J}} u_i} = \frac{r\cdot B}{r\cdot(x+e)+\delta^x_h\cdot\sum_{j\in\overline{\mathbf{J}^*}}(r_j+\beta_j)} \tag{4}$$

In $\mathcal{H}_7$, e is replaced by $\hat{e}$ in the view of $\mathcal{A}$, which implies that every corrupt party $\mathcal{P}_i$ will (if it acts honestly) compute u_i from $\hat{e}$ instead of from e via the equation in step 11 of $\pi_{\mathsf{BBS+}}$. Taking into account the value of u_j for $j \in \overline{\mathbf{J}^*}$ computed by $\mathcal{S}$, this implies that

$$\sum_{i\in\mathbf{J}} u_i = u - \sum_{i\in\mathbf{J}^*}(r_i+\beta_i)\cdot\delta^x_h = u + \delta^x_h\cdot\left(\sum_{j\in\overline{\mathbf{J}^*}} r_j - r - \sum_{i\in\mathbf{J}^*}\beta_i\right)$$

and since u is defined by $\mathcal{S}$ in $\mathcal{H}_7$ such that

$$A = \frac{B}{x+e} = \frac{r\cdot B}{u}$$

it must hold that $u = r\cdot(x+e)$. Recall that $e = \hat{e} + \delta^x_h$ and that

$$\sum_{i\in\mathbf{J}^*}\beta_i = -\sum_{j\in\overline{\mathbf{J}^*}}\beta_j$$

and so we know that if the adversary were to behave honestly when assembling the signature, then

$$\begin{aligned}\frac{\sum\limits_{i\in\mathbf{J}} R_i}{\sum\limits_{i\in\mathbf{J}} u_i} &= \frac{r\cdot B}{r\cdot(x+e)+\delta^x_h\cdot\left(\sum_{j\in\overline{\mathbf{J}^*}} r_j - r - \sum_{i\in\mathbf{J}^*}\beta_i\right)} \\ &= \frac{r\cdot B}{r\cdot(x+\hat{e})+\delta^x_h\cdot\sum_{j\in\overline{\mathbf{J}^*}}(r_j+\beta_j)}\end{aligned} \tag{5}$$

and by comparing equations 4 and 5 we can conclude that the relative distributions of $\hat{e}$, u_j, and R_j for $j \in \overline{\mathbf{J}^*}$ in $\mathcal{H}_7$ are identical to the relative distributions of e, u_j, and R_j for $j \in \overline{\mathbf{J}^*}$ in $\mathcal{H}_6$. Since e is uniformly-sampled but information-theoretically hidden from $\mathcal{A}$ in $\mathcal{H}_7$, $\hat{e}$ in $\mathcal{H}_7$ appears identically distributed to e in $\mathcal{H}_6$, and so the two hybrids are identically distributed overall.

Hybrid $\mathcal{H}_8$. In $\mathcal{H}_8$, we alter the code of $\mathcal{S}$ when a *weak parially-blind* signature is requested for a *corrupt* Client $\mathcal{C}^*$ and at least two honest parties participate in the signing process. Specifically, when such a signing request is made, $\mathcal{S}$ calculates δ^x_j and δ^r_j for $j \in \overline{\mathbf{J}^*}$ as specified in steps 24a and 25b of $\mathcal{S}_{\mathsf{BBS+}}$, and if there exist some $j, j' \in \overline{\mathbf{J}^*}$ such that $\delta^x_j \neq \delta^x_{j'}$ or $\delta^r_j \neq \delta^r_{j'}$ then $\mathcal{S}$ samples $u_j \leftarrow \mathbb{Z}_q$ uniformly for $j \in \overline{\mathbf{J}^*}$, rather than calculating these values as the honest parties otherwise would. $\mathcal{H}_8 = \mathcal{H}_7$ under the same argument as we presented for $\mathcal{H}_6$.

Hybrid $\mathcal{H}_9$. In $\mathcal{H}_9$, we fully implement $\mathcal{S}_{\mathsf{BBS+}}$ for *weak partially-blind signing* against a *corrupt* client $\mathcal{C}$. That is, we delete the honest parties code in such a case, and instead follow steps 29 and 30. Rather than querying $\mathcal{F}_{\mathsf{BBS+}}$, however, $\mathcal{S}$ generates a signature locally when appropriate using $\mathsf{BBS+Sign}$. $\mathcal{H}_9 = \mathcal{H}_8$ under the same argument as we presented for $\mathcal{H}_7$.

Hybrid $\mathcal{H}_{10}$. In $\mathcal{H}_{10}$, $\mathcal{Z}$ communicates with $\mathcal{S}$ instead of $\mathcal{A}$, and with dummy parties $\mathcal{P}_j$ for $j \in \overline{\mathbf{P}^*}$ instead of with $\mathcal{S}$. The dummy parties $\mathcal{P}_j$ forward their messages to $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, and $\mathcal{S}$ emulates the environment to $\mathcal{A}$ by forwarding all messages that are transmitted along its channel to $\mathcal{Z}$. The remaining honest protocol code of $\mathcal{P}_j$ for $j \in \overline{\mathbf{P}^*}$ is deleted from $\mathcal{S}$.

Instead, when the key generation phase of the protocol is initiated and the corrupt parties do not cheat, $\mathcal{S}$ initiates the key generation phase of $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, receives $(\mathbf{H}, X)$ as a consequence, and uses X to compute the shares $P(k)$ for $k \in [n]$ that it distributes on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}$ to $\mathcal{A}$. Likewise, when a signature (either plain, or weak partially-blind) is requested via a corrupt client $\mathcal{C}^*$, $\mathcal{S}$ does not compute a signature itself, as in $\mathcal{H}_9$, but instead requests a signature on behalf of $\mathcal{C}^*$ from $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$, and uses this as required to simulate the signing protocol. When a (plain or weak partially-blind) signature is requested via an honest client $\mathcal{C}$, $\mathcal{F}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ informs $\mathcal{S}$ of the request and leaks e (and s in the plain cases) to enable the protocol to be simulated, and $\mathcal{S}$ simply permits the functionality to release its output to the dummy $\mathcal{C}$ (or instructs it to output a failure), rather than running the code of $\mathcal{C}$ to reconstruct a signature and delivering it to the environment, as in $\mathcal{H}_9$.

These changes are purely syntactic, and thus $\mathcal{H}_{10}$ is distributed identically to $\mathcal{H}_9$. Furthermore, it is now the case that $\mathcal{S} = \mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n, t, \mathcal{G}, \ell)$ and that

$$\mathcal{H}_{10} = \left\{ \textsc{Ideal}_{\mathcal{F}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell), \mathcal{S}^{\mathcal{A}}_{\mathsf{BBS+}}(n,t,\mathcal{G},\ell), \mathcal{Z}}(\lambda, z) \right\}_{\substack{\lambda \in \mathbb{N}, n \in \mathbb{N}: n>1, t \in [2,n], \ell \in \mathbb{N}: \ell > 0, \\ \mathcal{G} \leftarrow \mathsf{BilinGen}(1^\lambda), z \in \{0,1\}^{\mathsf{poly}(\lambda)}}}$$

and so by transitivity equation 1 holds. □

9 Realizing Key Generation

In this section we describe how to realize the $\mathcal{F}_{\mathsf{DLKeyGen}}$ functionality that was given in section 3. We note that the protocol and proof given here are taken nearly-verbatim from the work of Doerner et al. [DKLs19]. We include them here in order to show that the protocol realizes our standalone key-sampling functionality, whereas Doerner et al. included this protocol as part of a larger protocol realizing a more complex functionality that included ECDSA signing.

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 $\mathcal{P}_i$ computes the sum $p(i)$ of their own point on their own polynomial and the points received, and takes this as their share of the secret key. To generate the public key corresponding to this secret key, parties perform a commit-and-release of $P(i) :=$

$p(i) \cdot G$ along with a proof of knowledge of discrete logarithm. Parties can compute $X = P(0) = p(0) \cdot G$ by interpolating a size t subset of $P(i)$ in the exponent. Malicious parties may however send malformed $P(i)$, so parties check that all received $P(i)$ lie on the same degree-$(t-1)$ polynomial.[7]

Protocol 9.1. $\pi_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$

This protocol is parameterized by the party count n, the threshold t, and a group $\mathcal{G} = (\mathbb{G}, G, q)$. It involves the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and the ideal functionality $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

Key Generation:

1. On receiving $(\mathtt{keygen}, \mathsf{sid})$ from the environment, each party $\mathcal{P}_i$ samples a random degree polynomial p_i of degree $t-1$.

2. For all pairs of parties $\mathcal{P}_i$ and $\mathcal{P}_j$, $\mathcal{P}_i$ sends $(\mathtt{poly\text{-}point}, \mathsf{sid}, p_i(j))$ to $\mathcal{P}_j$ (over a private channel) and receives $(\mathtt{poly\text{-}point}, \mathsf{sid}, p_j(i))$ in return.

3. Each party $\mathcal{P}_i$ computes
$$p(i) \coloneqq \sum_{j \in [n]} p_j(i) \qquad \text{and} \qquad P(i) \coloneqq p(i) \cdot G$$
and sends $(\mathtt{commit}, \mathsf{sid} \| \mathcal{P}_i, \{\mathcal{P}_j\}_{[n] \setminus \{i\}}, (P(i), G), p(i))$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

4. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ releases its proof by sending $(\mathtt{prove}, \mathsf{sid} \| \mathcal{P}_i)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

5. Each party $\mathcal{P}_i$ receives $(\mathtt{accepted}, \mathsf{sid} \| \mathcal{P}_j, (P(j), G))$ from $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ for each $j \in [n] \setminus \{i\}$ if $\mathcal{P}_j$'s proof of knowledge is valid. $\mathcal{P}_i$ aborts if it receives $(\mathtt{rejected}, \mathsf{sid} \| \mathcal{P}_j, *)$ instead for any proof, or if there exists an index $y \in [n-t-1]$ such that $\mathbf{J}^y = [y, y+t]$ and $\mathbf{J}^{y+1} = [y+1, y+t+1]$ and
$$\sum_{j \in \mathbf{J}^y} \mathsf{lagrange}(\mathbf{J}^y, j, 0) \cdot P(j) \neq \sum_{j \in \mathbf{J}^{y+1}} \mathsf{lagrange}(\mathbf{J}^{y+1}, j, 0) \cdot P(j)$$

6. The parties compute the shared public key using any subset $\mathbf{J} \subseteq [n]$ such that $|\mathbf{J}| = t$
$$X \coloneqq \sum_{j \in \mathbf{J}} \mathsf{lagrange}(\mathbf{J}, j, 0) \cdot P(j)$$
Each party $\mathcal{P}_i$ outputs $(\mathtt{key\text{-}pair}, \mathsf{sid}, X, p(i))$ to the environment.

[7] 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, and we use this strategy in the simulator.

Theorem 9.2 (Key Generation Security Theorem)**.** *Let $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, G_1, G_2, q)$ be the description of a bilinear group. $\pi_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ perfectly UC-realizes $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ in the $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$-hybrid model with selective abort against a malicious adversary that statically corrupts up to $n-1$ fixed parties.*

Proof. Formally, we show that for every group described by $\mathcal{G}$ and every malicious adversary $\mathcal{A}$ that statically corrupts up to $n-1$ parties, there exists a simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{DLKeyGen}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$ it holds that

$$\left\{\text{REAL}_{\pi_{\mathsf{DLKeyGen}}(n,t,\mathcal{G}),\mathcal{A},\mathcal{Z}}\left(\lambda, z\right)\right\}_{\lambda\in\mathbb{N},n\in\mathbb{N}:n>1,t\in[2,n],z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$$
$$= \left\{\text{IDEAL}_{\mathcal{F}_{\mathsf{DLKeyGen}}(n,t,\mathcal{G}),\mathcal{S}^{\mathcal{A}}_{\mathsf{DLKeyGen}}(n,t,\mathcal{G}),\mathcal{Z}}\left(\lambda, z\right)\right\}_{\lambda\in\mathbb{N},n\in\mathbb{N}:n>1,t\in[2,n],z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$$

Our proof is direct, without any hybrid experiments. The simulator is as follows:

Simulator 9.3. $\mathcal{S}^{\mathcal{A}}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$

This simulator is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The simulator has oracle access to the adversary $\mathcal{A}$, and emulates for it an instance of the protocol $\pi_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ involving the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. When the emulated protocol instance begins, $\mathcal{A}$ announces the identities of up to $t-1$ corrupt parties. Let the indices of these parties be given by $\mathbf{P}^* \subseteq [n]$. $\mathcal{S}^{\mathcal{A}}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ on behalf of every corrupt party, and in the experiment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and the corrupt parties on behalf of every honest party and on behalf of the ideal oracle $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

Key Generation:

1. On receiving $(\texttt{keygen-req}, \mathsf{sid}, j)$ from $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ such that $j \in [n] \setminus \mathbf{P}^*$, sample p_j to be a random polynomial of degree $t-1$ over $\mathbb{Z}_q$, and send $(\texttt{poly-point}, \mathsf{sid}, p_j(i))$ privately to party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{P}_j$.

2. On receiving $(\texttt{poly-point}, \mathsf{sid}, p_i(j))$ privately from every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{P}_j$ for some $j \in [n] \setminus \mathbf{P}^*$, and receiving $(\texttt{keygen-req}, \mathsf{sid}, j)$ from $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$, send $(\texttt{committed}, \mathsf{sid}\|\mathcal{P}_j, \mathcal{P}_j)$ to $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

3. On completing step 2 for every $j \in [n] \setminus \mathbf{P}^*$, and receiving $(\texttt{commit}, \mathsf{sid}\|\mathcal{P}_i, \{\mathcal{P}_j\}_{j\in[n]\setminus\{i\}}, (P(i), G), p(i))$ from $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, compute

$$p^*(i) := p(i) - \sum_{j\in[n]\setminus\mathbf{P}^*} p_j(i)$$

for $i \in \mathbf{P}^*$ and

$$p^*(j) := \sum_{i \in \mathbf{P}^*} p_i(j)$$

The values $p^*(i)$ for all $i \in [n]$ define a *consistent* polynomial of degree $t-1$ if and only if for all $y \in [n-t-1]$ defining $\mathbf{J}^y = [y, y+t]$ and $\mathbf{J}^{y+1} = [y+1, y+t+1]$,

$$\sum_{j \in \mathbf{J}^y} \mathsf{lagrange}(\mathbf{J}^y, j, 0) \cdot p^*(j) = \sum_{j \in \mathbf{J}^{y+1}} \mathsf{lagrange}(\mathbf{J}^{y+1}, j, 0) \cdot p^*(j)$$

4. If p^* is a consistent polynomial of degree $t-1$, and $p(i) \cdot G = P(i)$ for every $i \in \mathbf{P}^*$, then

   a. Send $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ on behalf of $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$.

   b. Send $(\texttt{poly-points}, \mathsf{sid}, \{p(i)\}_{i \in \mathbf{P}^*})$ directly to $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$, and receive receive $(\texttt{public-key}, \mathsf{sid}, X, \{P(1), \ldots, P(n)\})$ directly from $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ as a consequence.

   c. For every $j \in [n] \setminus \mathbf{P}^*$, send $(\texttt{accepted}, \mathsf{sid} \| \mathcal{P}_j)$ to $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

   d. For all $i \in \mathbf{P}^*$, wait to receive $(\texttt{prove}, \mathsf{sid} \| \mathcal{P}_i)$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, and instruct $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ to release its output to the honest parties.

5. If p^* is *not* a consistent polynomial of degree $t-1$, or there exists some $i \in \mathbf{P}^*$ such that $p(i) \cdot G \neq P(i)$, then

   a. For every $j \in [n] \setminus \mathbf{P}^*$, compute

   $$P(j) := \Big(p^*(j) + \sum_{k \in [n] \setminus \mathbf{P}^*} p_k(j)\Big) \cdot G$$

   b. For every $j \in [n] \setminus \mathbf{P}^*$, send $(\texttt{accepted}, \mathsf{sid} \| \mathcal{P}_j, P(j))$ to $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$.

   c. For all $i \in \mathbf{P}^*$, wait to receive $(\texttt{prove}, \mathsf{sid} \| \mathcal{P}_i)$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, and then instruct $\mathcal{F}_{\mathsf{DLKeyGen}}(n, t, \mathcal{G})$ to abort.

Following the reception of the corrupted parties' polynomial points in step 5 of π_{DLKeyGen}, there are two cases in the ideal experiment, defined by the abort conditions described in step 3 of $\mathcal{S}^{\mathcal{A}}_{\mathsf{DLKeyGen}}$. In the first case, there exists $y \in [n-t-1]$ such that

$$\sum_{j \in \mathbf{J}^y} \mathsf{lagrange}(\mathbf{J}^y, j, 0) \cdot P(j) \neq \sum_{j \in \mathbf{J}^{y+1}} \mathsf{lagrange}(\mathbf{J}^{y+1}, j, 0) \cdot P(j)$$

and the simulation aborts. It is trivially true that π_{DLKeyGen} also aborts in this case, since the honest parties perform exactly the same test in Step 5 of π_{DLKeyGen}, and because the simulator follows the protocol's instructions exactly in this case (sampling the honest parties' polynomials uniformly, just as they would), the messages it sends are identically distributed to their counterparts in the real experiment.

In the second case,

$$\sum_{j \in \mathbf{J}^y} \mathsf{lagrange}(\mathbf{J}^y, j, 0) \cdot P(j) = \sum_{j \in \mathbf{J}^{y+1}} \mathsf{lagrange}(\mathbf{J}^{y+1}, j, 0) \cdot P(j) \tag{6}$$

for all $y \in [n-t-1]$, no abort occurs in either the real experiment or the ideal one, and the views of the corrupted parties are characterized by the values $P(j)$ for $j \in [n] \setminus \mathbf{P}^*$. Note that in both experiments, equation 6 implies that these values are completely constrained by $P(i)$ for $i \in \mathbf{P}^*$ and pk. In the real experiment, we know that X is uniform by virtue of containing an additive, uniform contribution from at least one honest party. In the ideal experiment, $\mathcal{F}_{\mathsf{DLKeyGen}}$ generates a uniform X and and uses Lagrange interpolation to reconstruct the appropriate values of $P(j)$. Thus the experiments are identically distributed. $\square$

Acknowledgements

The authors of this work were supported by the NSF under grants 1646671, 1816028, and 2055568, by the ERC under projects NTSC (742754) and SPEC (803096), by ISF grant 2774/2, and by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

References

[AHS20] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ECDSA threshold signing. *IACR Cryptol. ePrint Arch.*, page 1390, 2020.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In Roberto De Prisco and Moti Yung, editors, *Proceedings of the 5th Conference on Security and Cryptography for Networks (SCN)*, pages 111–125, 2006.

[BB89] 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.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, pages 56–73, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, pages 41–55, 2004.

[BCC$^+$09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 108–125, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. pages 489–518, 2019.

[BCG$^+$20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1069–1080. IEEE, 2020.

[Bea95] Donald Beaver. Precomputing oblivious transfer. pages 97–109, 1995.

[BL09] Ernie Brickell and Jiangtao Li. Enhanced privacy id from bilinear pairing. *IACR Cryptol. ePrint Arch.*, 2009:95, 2009.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). pages 503–513, 1990.

[Bow17] Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. 2017.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CCL$^+$20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *Public-Key Cryptography - PKC 2020*, 2020.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In *Advances in Cryptology – ASIACRYPT 2015, part II*, pages 262–288, 2015.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016*, pages 1–20. Springer, 2016.

[CE87] David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 118–167, Berlin, Heidelberg, 1987. Springer-Verlag.

[CGG$^+$20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *CCS '20*. ACM, 2020.

[CGSB] Melissa Chase, Esha Ghosh, Srinath Setty, and Daniel Buchner. Zero-knowledge credentials with deferred revocation checks. https://github.com/decentralized-identity/snark-credentials/blob/master/whitepaper.pdf.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, oct 1985.

[Che96] Lidong Chen. Access with pseudonyms. In Ed Dawson and Jovan Golić, editors, *Cryptography: Policy and Algorithms*, pages 232–243, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[CKL$^+$16] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, pages 3–24, Cham, 2016. Springer International Publishing.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001*. Springer Berlin Heidelberg, 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, pages 78–96, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[CL19] Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 535–555, Cham, 2019. Springer International Publishing.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1205–1216, New York, NY, USA, 2014. Association for Computing Machinery.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. *The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption*, page 1445–1459. Association for Computing Machinery, New York, NY, USA, 2020.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *Proceedings of the 39th IEEE Symposium on Security and Privacy, (S&P)*, pages 980–997, 2018.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *Proceedings of the 40th IEEE Symposium on Security and Privacy, (S&P)*, 2019.

[DOK$^+$20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS 2020*, 2020.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*, page 643–662, Berlin, Heidelberg, 2012. Springer-Verlag.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, pages 416–431, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, jun 1985.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology – CRYPTO 2005*, pages 152–168, 2005.

[GGI19] Rosario Gennaro, Steven Goldfeder, and Bertrand Ithurburn. Fully distributed group signatures, 2019.

[Gil99] Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology – CRYPTO 1999*, pages 116–129, 1999.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[GMT20] Aurore Guillevic, Simon Masson, and Emmanuel Thomé. Cocks-pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Cryptogr.*, 88(6):1047–1081, 2020.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. 1987.

[GPS08] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. pages 339–352, 1995.

[HMRT22] Iftach Haitner, Nikolaos Makriyannis, Samuel Ranellucci, and Eliad Tsfadia. Highly efficient ot-based multiplication protocols. *EUROCRYPT '22*, 2022.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. 2003.

[KIK$^+$17] Yutaro Kiyomura, Akiko Inoue, Yuto Kawahara, Masaya Yasuda, Tsuyoshi Takagi, and Tetsutaro Kobayashi. Secure and efficient pairing at 256-bit security level. In *Applied Cryptography and Network Security*, pages 59–79, 2017.

[KMOS21] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer 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*, pages 608–625. IEEE, 2021.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. pages 830–842, 2016.

[KS22] Yashvanth Kondi and Abhi Shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In *Advances in Cryptology – ASIACRYPT 2022, part II*, pages 279–309, 2022.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *IACR Cryptol. ePrint Arch.*, page 374, 2022.

[LKWL22] Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-01, Internet Engineering Task Force, October 2022. Work in Progress.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. pages 1837–1854, 2018.

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, 1999.

[MP] Nikolaos Makriyannis and Udi Peled. A note on the security of gg18. https://info.fireblocks.com/hubfs/A_Note_on_the_Security_of_GG.pdf.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In *Proceedings of the 26th ACM Conference on Computer and Communications Security, (CCS)*, pages 309–326, 2019.

[Nym22] Nymtech, 2022. https://github.com/nymtech/nym/tree/develop/common/nymcoconut.

[Oka06a] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *Proceedings of the Third Theory of Cryptography Conference, TCC 2006*, pages 80–99, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Oka06b] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. Cryptology ePrint Archive, Report 2006/102, 2006. https://ia.cr/2006/102.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). pages 51–59, 1991.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*, page 111–126, Berlin, Heidelberg, 2016. Springer-Verlag.

[Roy22a] Lawrence Roy. personal communication, 2022.

[Roy22b] Lawrence Roy. Softspokenot:communication-computation tradeoffs in ot extension. In *Advances in Cryptology – CRYPTO 2022*, 2022.

[RP22] Alfredo Rial and Ania 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.

[SA19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In Martin Albrecht, editor, *IMACC 2019*, 2019.

[SAB$^+$19] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George 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.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology – CRYPTO 1989*, pages 239–252, 1989.

[SKSW20] Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby. Pairing-friendly curves. 2020.

[TAKS07] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: Blocking misbehaving users without ttps. In *Proceedings of the 14th ACM Conference on Computer and Communications Security, (CCS)*, page 72–81, 2007.

[TS21] Dmytro Tymokhanov and Omer Shlomovits. Alpha-rays: Key extraction attacks on threshold ecdsa implementations. Cryptology ePrint Archive, Paper 2021/1621, 2021.

[TZ23] Stefano Tessaro and Chenzhi Zhu. Revisiting bbs signatures. Cryptology ePrint Archive, Paper 2023/275, 2023. https://eprint.iacr.org/2023/275.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. pages 21–37, 2017.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). 1986.